The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM), 2024

Volume 27, Pages 214-227

IConTech 2024: International Conference on Technology

Performance Analysis of Encryption Algorithms in Named Pipe Communication for Linux Systems

Huseyin Karacali
TTTech Auto Turkey

Efecan Cebel
TTTech Auto Turkey

Nevzat Donum
TTTech Auto Turkey

Abstract: In modern computing environments, inter-process communication (IPC) plays a pivotal role in facilitating seamless interaction between software components. Named pipes, a form of IPC mechanism in Linux systems, offer a straightforward means of data exchange between processes. However, ensuring the confidentiality and integrity of data transmitted via named pipes is essential, particularly in environments where sensitive information is handled. This paper presents a comprehensive investigation into the performance characteristics of various encryption algorithms applied to named pipe communication within Linux systems. The efficiency of six encryption algorithms such as Caesar cipher, RSA, DES, AES-128, AES-192, and AES-256 is examined in terms of their impact on data throughput, latency, and resource utilization within the named pipe communication. Through a series of systematic experiments, encompassing diverse datasets and transmission scenarios, the trade-offs between security and performance inherent in each encryption algorithm are analyzed. Our findings shed light on the relative strengths and weaknesses of different encryption techniques, providing valuable insights for system administrators and developers in selecting appropriate encryption methods based on specific application requirements and security considerations. This study contributes to the broader understanding of secure IPC mechanisms in Linux environments, offering a nuanced perspective on the interplay between encryption algorithms and system performance in the context of named pipe communication.

Keywords: Inter-process communication, Encryption algorithms, Linux systems, Named pipe

Introduction

In the realm of digital security, cryptographic encryption methods serve as the cornerstone for safeguarding sensitive information and ensuring secure communication channels. This paper delves into a comparative analysis of encryption methods applied specifically to named pipes within the Linux environment. Named pipes, also known as FIFOs (First In, First Out), offer a means of inter-process communication (IPC), facilitating data exchange between unrelated processes.

In the past, the performance of encryption methods has been examined and compared with each other many times. There are dozens of studies on these in the literature. In particular, AES and DES are the 2 standards that are most compared. In the study conducted at Galgotias University in 2015, performance calculations and comparisons of AES and DES were made, and in this study, the variation and simulation time of these 2 encryption algorithms were focused on (Bhat et al., 2015). In another paper comparing AES and DES, the encryption times and CPU usage of these algorithms were discussed in the same way as this study (Rihan et al.,

2015). The originality of this study is that Caesar cipher is included in the comparison and the system is tested using named pipe in inter-process communication in Linux structure.

The study methodically investigates the performance of three encryption algorithms: Caesar Cipher, Data Encryption Standard (DES), and Advanced Encryption Standard (AES). These algorithms are scrutinized across three key phases: research and comprehension of cryptographic methods, implementation within the named pipe infrastructure, and comprehensive performance measurements. Additionally, the RSA algorithm has been chosen as the key determination method in all these encryption methods. In practical applications, all key generation is done using the RSA algorithm.

The research phase entails a thorough exploration of each encryption and decryption method, including key generation techniques. This foundational understanding forms the basis for subsequent algorithm development and integration into the Linux environment's named pipe structure.Subsequently, the study focuses on the practical implementation of the encryption algorithms within the named pipe framework. This phase involves the successful execution and evaluation of RSA, Caesar Cipher, DES, and AES algorithms, highlighting their applicability and efficiency in the Linux environment.

Finally, comprehensive measurements are conducted to assess the performance of the encryption methods. Time measurements provide insights into the encryption duration of each algorithm, while CPU utilization measurements offer a glimpse into the resource consumption during encryption operations. By systematically examining the performance of these encryption methods within the Linux environment's named pipe infrastructure, this study aims to provide valuable insights into selecting suitable encryption algorithms for ensuring secure communication channels in digital systems.

Material

Named Pipe

Inter-process communication (IPC) forms the cornerstone of effective multitasking in operating systems. Named pipes, also known as FIFOs (First-In-First-Out), offer a well-established method for IPC, particularly within Unix-like systems (Gaikwad,2023). They build upon the traditional pipe concept by introducing a crucial distinction: persistence.

Unlike standard, anonymous pipes that vanish with their creator process, named pipes exist as named entities within the file system. This characteristic allows them to transcend the lifespan of their originating process, enabling communication between processes that might not be running concurrently. Named pipes function as special files, accessible by processes through familiar file operations like opening, reading, writing, and closing (Adegeo, n.d.), A defining feature of named pipes is their adherence to the FIFO principle. Data written to the pipe by one process is retrieved by another process in the exact order it was written. This ensures the integrity of messages and streamlines communication flow.

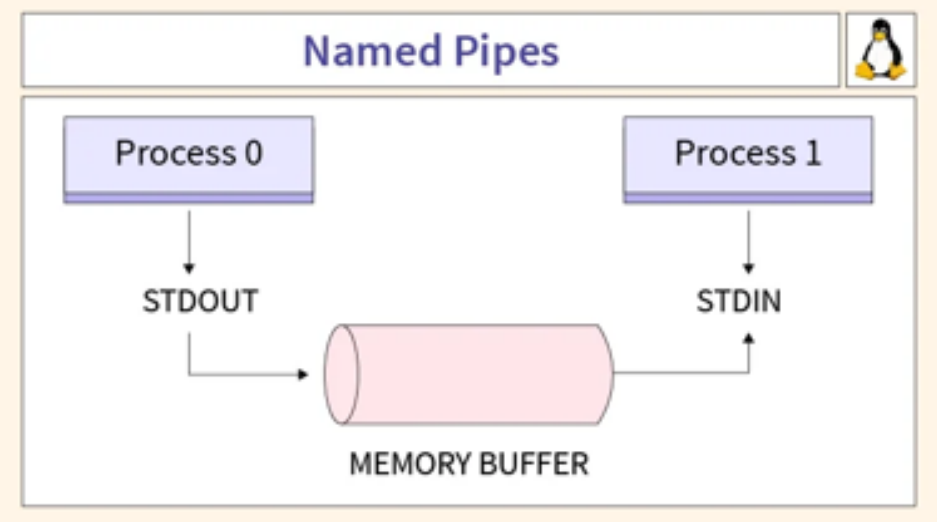

Figure 1. A representation of named pipe (Gaikwad, 2023)

The specific implementation details of named pipes vary across operating systems. In Unix-like environments, commands like mkfifo create named pipes, while functions like open and read/write handle data exchange (linux manual page, n.d). In conclusion, named pipes provide a robust and user-friendly mechanism for inter-process communication. Their persistence, adherence to FIFO principles, and ability to bridge communication between independent processes make them a valuable tool for various IPC applications.

Development and Test Environment

The encryption algorithms with named pipes were developed, compiled, and executed on a single computer to maintain consistent and precise comparisons. This computer operates on a Linux-based virtual machine running Ubuntu-20.04.1, utilizing a 64-bit architecture with x86_64 and a 5.15.0-101-generic kernel. All of the developments such as creating named pipe, RSA, DES, AES cryptosystems were developed with C programming language and compiled using gcc version 9.4.0.

Method

The methodology of this study, in which the performances of the encryption methods used on the named pipe are compared, is carried out in 3 main steps. The first of these is to research and learn these encryption and decryption methods in the field of cryptology. These also include generating keys. Following this research, we will develop and successfully run these encryption algorithms on the named pipe structure established in the Linux environment. Finally, several measurements are made on these structures for performance observation, which is the main purpose of the study.

Cryptographic Encryption Methods

Rivest, Shamir and Adleman (RSA) Cryptosystem

The RSA method is a frequently used asymmetric encryption technology for safe data transport. RSA, named after its founders Rivest, Shamir, and Adleman, encrypts data with a public key that can be freely transmitted, whereas decryption requires a private key that the receiver keeps secret. RSA's security is predicated on the practical difficulty of factoring the product of two large prime numbers, making it a key component of digital security in applications such as safe online browsing, email, and corporate data protection.

RSA encryption uses two keys: a public key for encryption and a private key for decryption. Its security arises from the difficulty of factoring huge numbers into primes, which is computationally expensive for large numbers. RSA is used to ensure secure data transmission, digital signatures, and key exchange. Its effectiveness is determined by key size, with longer keys providing higher security but needing more processing resources. RSA's use in SSL/TLS protocols for secure web connections demonstrates its importance in current cryptography.

Considering one letter sized plain text, letter B to be encrypted. The number representation of B is 2. Let public key for encryption is defined as (5, 14). Thus, cipher text is calculated $2^5 \pmod{14} \equiv 4$ in this case. Then, the letter equivalent of this is D. Also, private key for decryption is chosen as (11, 14). When back-propagation is applied, original text can be reconstructed. $4^{11} \pmod{14} \equiv 2$ (B).

In RSA, keys are generated by selecting two large prime numbers (p and q), calculating their product (n), and then determining a number (e) that is coprime with (n) and the product of the primes' decrements. The public key consists of (n) and (e), but the private key is composed of (n) and a number (d) that solves a certain modular equation involving (e). The procedure assures that public and private keys are mathematically connected, allowing for secure encryption and decryption processes.

Let us choose p= 2 and p= 7. n= p*q= 14 in this case. This value will be the modulo in encryption and decryption keys. Remainder numbers which are coprime with n (sharing no common factors with n) which are 1, 3, 5, 9, 11 and 13 in this case. In real scenario, p and q might be enormous, so calculating coprime would be difficult. The number of remainder numbers is equal to Ø(n)= 6. This term can be also obtained as Ø(n)= (p-1)*(q-1). Then, choosing number e under two conditions:

- $1 < e < Ø(n)$
- Coprime with n, $Ø(n)$

5 can be chosen from four options (2, 3, 4, 5), because only 5 is coprime with n and $Ø(n)$. Then the next step is determining number d for private key. Choosing d: $d*e \pmod{Ø(n)} \equiv 1$. In this case, 11 can be chosen from all options (4, 11, 18, 25…). Finally, public, and private keys for RSA cryptosystem is generated.

Caesar (Shift) Cipher

The Caesar cipher is one of the most simple and well-known encryption methods. It is a type of substitution cipher in which each letter in the plaintext is shifted a set number of positions down or up the alphabet. For example, with a shift of one, 'A' would be replaced by 'B', 'B' by 'C', and so on. Also, spaces and punctuations are reserved. This approach is named after Julius Caesar, who allegedly used it to communicate with his generals. The Caesar cipher's simplicity makes it easy to comprehend, but equally easy to break, restricting its practical relevance to modern security requirements.

For an n-letter alphabet; P, C, K ∈ Zn, encryption $EK(P) = P + K \pmod{n}$, decryption $DK(P) = C – K \pmod{n}$. Let consider the K (shift key) is 4, and plain text is “This is an encrypted message.”, cipher text can be created from the alphabet table in the figure above. For example, the letter T becomes X since the key is 4, so cipher text becomes: “XLMW MW ER IRGVCTXIH QIWWEKI.”.

The Caesar cipher's simplicity is also its primary drawback. It is vulnerable to frequency analysis, which involves an attacker comparing the frequency of letters or groups of letters in the ciphered text to known frequencies in the original message's language. Because the cipher does not dramatically modify these frequencies, it is quite simple to calculate the shift and decrypt the message. Furthermore, because there are only 25 potential shifts in the English alphabet, an attacker can easily try all combinations to decrypt the message.

The Caesar cipher is quite simple to decipher. The original data can be accessed by methods such as analysis of the most repeated characters, determining other characters by selecting one character as plaintext, or vice versa. Caesar cipher was chosen as the simplest encryption method in this study. The performance of Caesar encryption, which has a very simple structure compared to Advanced Encryption Standard (AES) and Data Encryption Standard (DES), has been observed.

Data Encryption Standard (DES)

The Data Encryption Standard (DES) is a symmetric-key block encryption algorithm created by IBM in the 1970s and later standardized by NIST. Its principal function is to encrypt and decrypt digital data with a common secret key. DES encrypts and decrypts plaintext blocks, which are typically 64 bits in size, using a 56-bit key. Although DES has been mostly replaced by more secure algorithms such as AES, knowing its operation provides insight into the fundamentals of modern encryption.

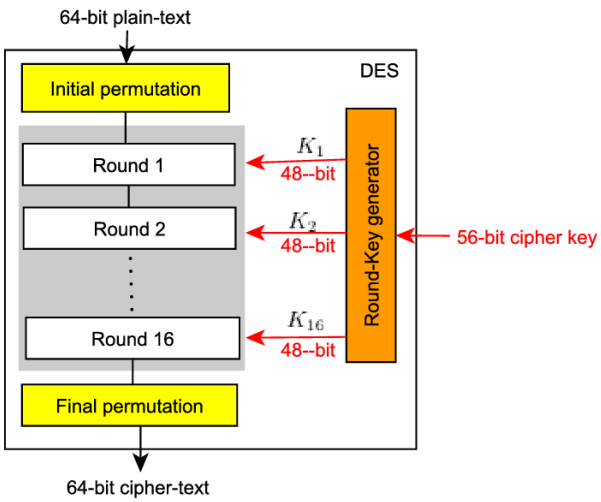

Figure 2. DES structure (Shorman & Qatawneh, 2018)

The first stage in DES is key generation, which converts the 56-bit key into 16 subkeys, each 48 bits long. The procedure starts with an initial permutation and then separates the key into two 28-bit halves. Circular left shifts are made to each half, and subkeys are formed by selecting specified groups of bits using a technique known as key scheduling. These subkeys are subsequently used for further encryption and decryption operations (Schneier & Diffie, 2015).

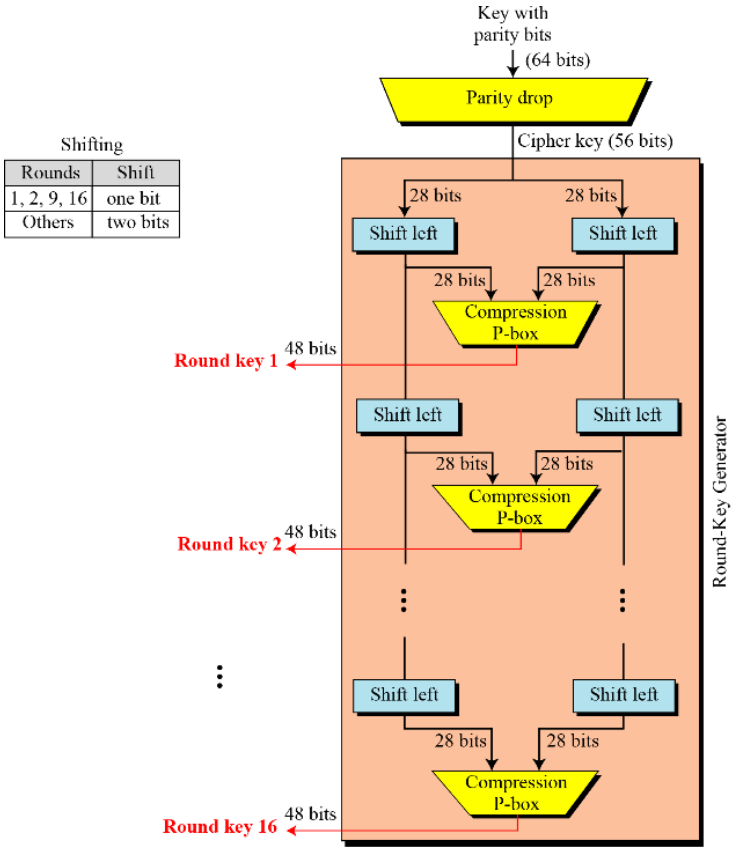

Figure 3. Key generation for DES algorithm (Sharmal & Garg, 2016)

DES encryption involves 16 rounds of processing for each plaintext block. Each round includes multiple operations such as substitution, permutation, and key mixing. The plaintext block is first permuted, then transformed in a sequence of rounds. Each round, the block is divided into two halves, expanded, XORed with a subkey, substituted using S-boxes, permuted, and XORed with the other half. This procedure scrambles plaintext into ciphertext in a reversible manner using the proper key.

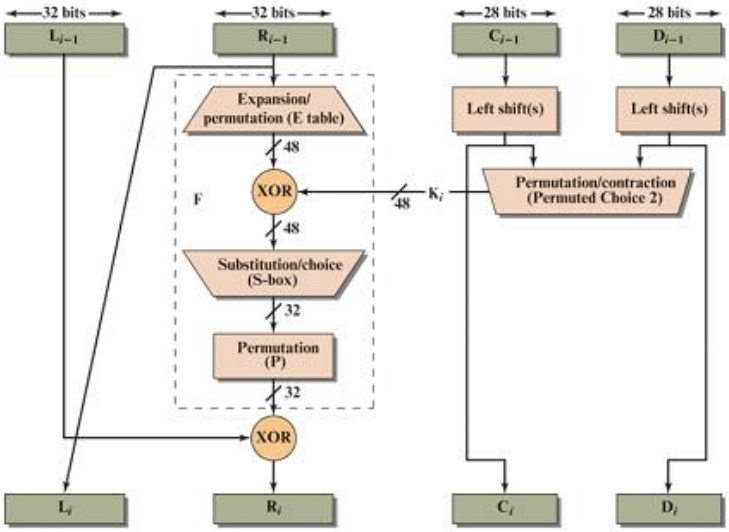

Figure 4. Single round of DES algorithm (Takieldeen et al., 2012)

DES decryption is the same as encryption, but in reverse. The ciphertext is initially permuted, then the subkeys are applied in reverse order across 16 rounds of processing. Each round involves the identical operations as encryption, except the subkeys are used in the reverse order. After the final round, the ciphertext block is treated to an inverse initial permutation, yielding the original plaintext block. Despite its historical relevance, DES has a number of flaws that make it unsuitable for modern cryptography applications. Its small key length renders it vulnerable to brute-force assaults, in which all potential keys can be checked within a reasonable timeframe. Furthermore, developments in cryptanalysis have revealed flaws in the DES algorithm, reducing its security. As a result, DES has been replaced by more powerful encryption protocols such as AES, which provide higher security assurances and improved performance.

Advanced Encryption Standard (AES)

AES (Advanced Encryption Standard) is a symmetric encryption algorithm that has become an essential component of modern cryptographic protocols. AES was developed to replace the outdated Data Encryption Standard (DES) and was adopted as a standard by the United States National Institute of Standards and Technology (NIST) in 2001 following a rigorous selection process. Unlike asymmetric encryption algorithms, which use separate keys for encryption and decryption, AES uses a single key for both operations, resulting in a symmetric encryption method. This key is shared by the communicating parties and must be kept secure in order to ensure the secrecy of the encrypted data (Hioureas, 2023).

AES's processing on data blocks is crucial to its functionality. Each block is made up of 128 bits, or 16 bytes, and if the plaintext is not a multiple of this block size, padding is used to ensure consistency. AES provides key sizes of 128, 192, and 256 bits, with bigger key sizes providing more security at the expense of computational complexity. The algorithm uses a substitution-permutation network (SPN) to perform its operations. These procedures take place over numerous rounds, with the number of rounds varied according to the key size: 10 rounds for AES-128, 12 rounds for AES-192, and 14 rounds for AES-256.

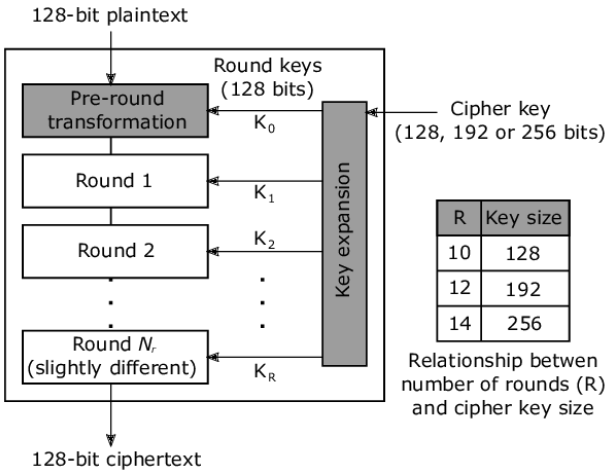

Figure 5. AES structure (Abdelrahman et al., 2017)

The AES encryption process begins with key expansion, which converts the initial key into a series of round keys, one for each round of encryption. Each round of AES encryption consists of four major steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey. In the SubBytes step, each byte in the input block is replaced with a corresponding byte from a substitution table, introducing non-linearity into the encryption process. ShiftRows is the process of cyclically shifting the bytes within each row of the block, which contributes to data diffusion. MixColumns treats the block's columns as polynomials and multiplies them with fixed polynomials modulo an irreducible polynomial. Finally, AddRoundKey applies bitwise XOR to merge the state and round key (Daemen & Rijmen, 2002).

SubBytes are the first stages in each round of AES encryption. It aims is to replace each byte in the input block with a corresponding byte from a prepared substitution table, known as the S-box. This replacement is a nonlinear process that causes confusion in the data. The S-box is a fixed 16x16 matrix with pre-computed values. Each byte in the input block is replaced with the value from the relevant row and column in the S-box. This transformation ensures that even minor changes in the input block cause huge changes in the output, making it difficult for attackers to identify patterns.

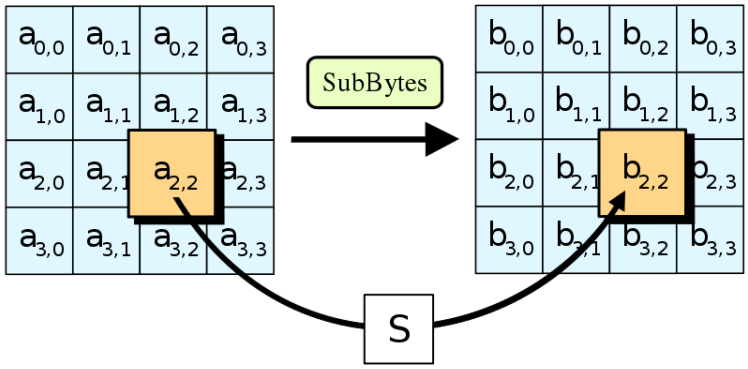

Figure 6. SubBytes act on the individual bytes of the state (“SubBytes”, 2024)

ShiftRows is the next stage in each round of AES encryption. This procedure consists of cyclically shifting the bytes within each row of the block. The bytes in the second row are shifted one position to the left; those in the third row are shifted two places to the left; and those in the fourth row are shifted three positions to the left. This phase adds to data diffusion by spreading each byte's influence across numerous columns. ShiftRows ensures that neighboring bytes interact with one another during successive encryption steps, hence improving the algorithm's overall security.

MixColumns follows ShiftRows with the exception of the final round of AES encryption. This procedure treats the block's columns as polynomials over the finite field GF(2^8). MixColumns multiplies each byte in a column by a fixed polynomial modulo an irreducible polynomial. The result replaces the original byte, yielding a linear transformation of the data. This process further distributes the data and ensures that each byte of the output is dependent on multiple bytes from the input. By creating this reliance, MixColumns improves the overall security of AES encryption, making it more resistant to cryptanalytic attacks.

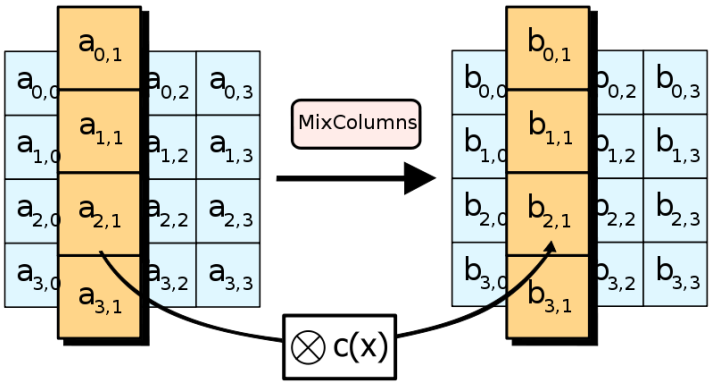

Figure 7. MixColumns operates on the columns of the state (“MixColumns”, 2024)

The last step in each round of AES encryption is AddRoundKey. In this stage, the block's current state is joined with a round key generated from the main encryption key. Each byte of the state is bitwise XORed with its matching byte from the round key. The round key created during key expansion is unique to the current round of encryption. AddRoundKey ensures that each round of encryption is separate and depends on the key by introducing the round key's unique effect into the state. This phase further obscures the relationship between the plaintext and the ciphertext, which improves the security of AES encryption.

AES is a symmetric key algorithm, which means it uses the same key for encryption and decoding. This differs from asymmetric key methods, which use two separate keys (public and private) for encryption and decryption. Symmetric key methods are often faster and more efficient for large volumes of data; nevertheless, key management can be difficult because securely communicating the key with the intended recipient is critical.

AES operates on fixed-size data blocks (128 bits). Different modes of operation can be used to encrypt data that does not fit into a single block, or to encrypt numerous blocks with increased security, such as Electronic Codebook (ECB), Cipher Block Chaining (CBC), or Galois/Counter Mode (GCM). These modes specify how the plaintext is divided into blocks and how the encryption process is performed on each block. Electronic Codebook (ECB) encrypts each block separately with the same key, which can reveal patterns in the ciphertext

if the plaintext contains repeated data. It is typically regarded as less secure and not recommended for most purposes. On the other hand, Cipher Block Chaining (CBC) creates a dependency between blocks by XORing the previous block's ciphertext with the current block's plaintext prior to encryption. The first block uses an Initialization Vector (IV) to add randomization. This mode offers more security than ECB, but it is vulnerable to certain attacks if the IV is predictable or overused.

AES is regarded extremely secure and is widely used in a variety of applications, including government, military, and commercial use. The algorithm's security stems mostly from its key size, which makes it resistant to brute-force attacks. Brute-force attacks attempt to decrypt the ciphertext by trying every conceivable key combination; however, with AES key sizes (128, 192, or 256 bits), the number of possible possibilities is so huge that it is currently deemed impossible to break AES encryption using this technique.

AES is a cornerstone of digital security, with use ranging from government communications to commercial transactions nowadays. Its exceptional resilience against brute-force attacks, combined with key size versatility (128, 192, and 256 bits), ensures strong defense mechanisms for sensitive information protection. AES's efficiency and security have won it a key role in global standards and protocols, making it a must have tool in the fight against cybersecurity threats. As we traverse the digital age, AES's importance grows, emphasizing its important role in protecting digital assets and communications around the world.

Development of the Named Pipe and Encryption Algorithms

Linux Inter-Process Communication with Named Pipe

In C programming on Linux systems, named pipes, often known as FIFOs (First In, First Out), provide a means for inter-process communication (IPC) that allows unrelated programs to share data. Unlike anonymous pipes, which are commonly used for parent-child process communication, named pipes exist independently of the processes that utilize them and persist in the file system, offering a path for communication between any processes that have access to the filesystem. To create a named pipe, use the mkfifo system function or command. This method creates a FIFO special file with the specified name in the filesystem. The required C libraries for mkfifo command are ‘types.h’ and ‘stat.h’ in the sys directory (Stevens & Rago, 2014).

The mkfifo command in the C programming language has an integer return value and 2 arguments. These arguments are the ‘pathname’ as a constant char pointer and ‘mode’ with the type of ‘mode_t’. The pathname represents the name of the directory of the FIFO to be created and “mode” specifies the permissions for the FIFO. The “mode” is specified as an octal (base-8) number and represents the file’s permission bits. It’s similar to the permissions used for regular files and directories. The “mode” is influenced by the process’s ‘umask’, which may restrict the permissions set during the creation of the FIFO. The “mode” parameter is composed of three groups of permissions:

- Owner permissions: What actions the owner of the file can perform.
- Group permissions: What actions users who are members of the file’s group can perform.
- Other permissions: What actions all other users can perform.

Each group can have permissions for reading (r), writing (w) and execution (x), represented by octal numbers:

- 4 (100 in binary) stands for read permission.
- 2 (010 in binary) stands for write permission.
- 1 (001 in binary) stands for execute permission.
- 0 stands for no permission.

These permissions are added to together to get the total permission value for each group. The final mode is a concatenation of these values for the owner, group and others in that order.

Opening the Named Pipe

Once created, processes can open the named pipe using open(.), just as they would with regular files. A process can open the FIFO in read-only (RDONLY) or write-only (WRONLY) mode, depending on its role. The writer

process, which sends data into the FIFO, opens it for writing. Also, the reader one opens it for reading. When a process is done with the FIFO, it can close it using close(.), similar to files.

Reading from and writing to a named pipe are accomplished with the read(.) and write(.) system methods, respectively. These calls halt the calling process: a read(.) call on an empty FIFO will block until there is data to read, and a write(.) call on a full FIFO will block until there is space to write new data. This blocking feature allows the producer and consumer processes to synchronize without the need for additional coordination code.

Caesar, RSA, DES, and AES Algorithm in C

Overview of the OpenSSL

The OpenSSL project, a powerful, commercial-grade, and feature-rich toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols, has expanded dramatically over time. One critical component of its evolution is the creation and improvement of its cryptography library, which includes a diverse set of cryptographic algorithms and capabilities like as AES, DES, and RSA. In subsequent releases, OpenSSL has continued to evolve, improving its support for these methods via its digital envelope library.

Digital Envelope Library

The digital envelope technique secures a communication by using asymmetric encryption to encrypt a symmetric key, which is then used to encrypt the message or data. This method combines the effectiveness of symmetric encryption algorithms (such as AES and DES) for large-scale data encryption with the security of asymmetric encryption algorithms (such as RSA) for safe key exchange (Viega et al., 2002).

The EVP (Envelope) interface serves as the foundation for OpenSSL's digital envelope capabilities. The EVP interface provides a higher-level abstraction of the different cryptographic methods. It provides a uniform method to encryption and decryption, hashing, and digital signature operations using diverse algorithms. By abstracting the complexities of each cryptographic technique, the EVP interface makes it easier to integrate encryption into programs while also ensuring that the algorithms are utilized appropriately and securely.

Usage in C and Named Pipes

When implementing cryptographic operations in C with OpenSSL, developers use the EVP interface to execute encryption and decryption. This method enables a seamless transition between multiple algorithms (AES, DES, RSA) without requiring significant changes to the codebase. For example, developers can encrypt data with AES for efficiency and then use RSA to encrypt the AES key, resulting in a secure digital envelope (Stallings, 2017).

As it mentioned before, cryptography is essential in scenarios involving inter-process communication (IPC), such as when employing named pipes (FIFOs), to ensure the confidentiality and integrity of the data being transmitted. In Unix-like operating systems, named pipes can be built and accessed using functions such as mkfifo to simplify communication between processes running on the same machine. Developers can use OpenSSL's cryptographic capabilities to encrypt data before sending it via the pipe and decrypt it upon receipt. This method assures that even if the data is intercepted while in transit over the designated pipe, it is protected and unreadable without the correct decryption key.

This combination of OpenSSL with named pipes for safe IPC is especially useful in applications that require secure transport of sensitive information between various components or services running on the same system. Using OpenSSL's digital envelope features ensures that data enclosed within a secure envelope is efficiently encrypted and securely sent, combining the strengths of symmetric and asymmetric cryptography.

OpenSSL's digital envelope library, which supports the AES, DES, and RSA algorithms, is a powerful and versatile toolset for performing cryptographic operations in C, including secure inter-process communication via named pipes. By abstracting the intricacies of cryptographic operations and assuring secure key and data handling, OpenSSL allows developers to create more secure applications that can confidently protect sensitive information from interception and unwanted access.

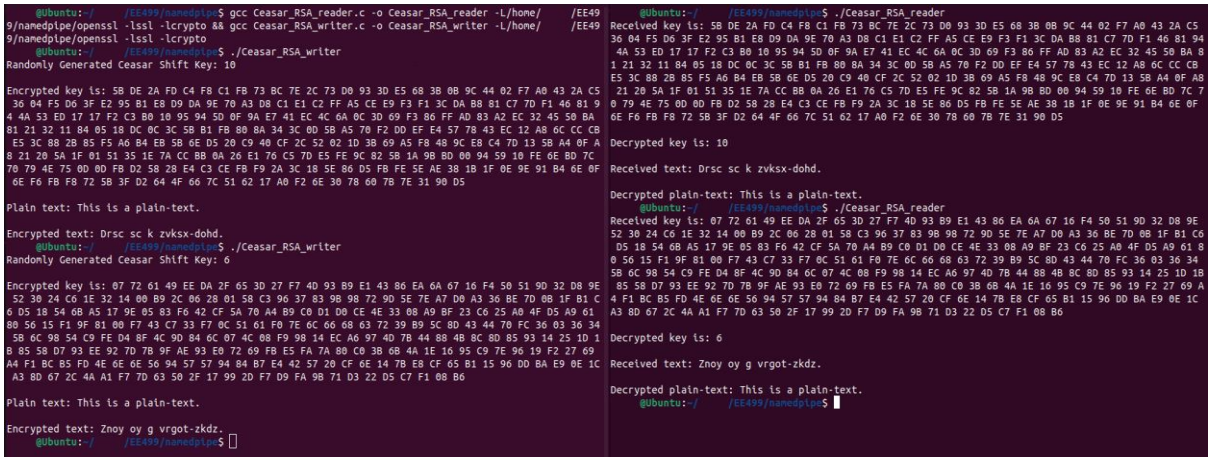

Figure 8. Caesar encryption with RSA secured key

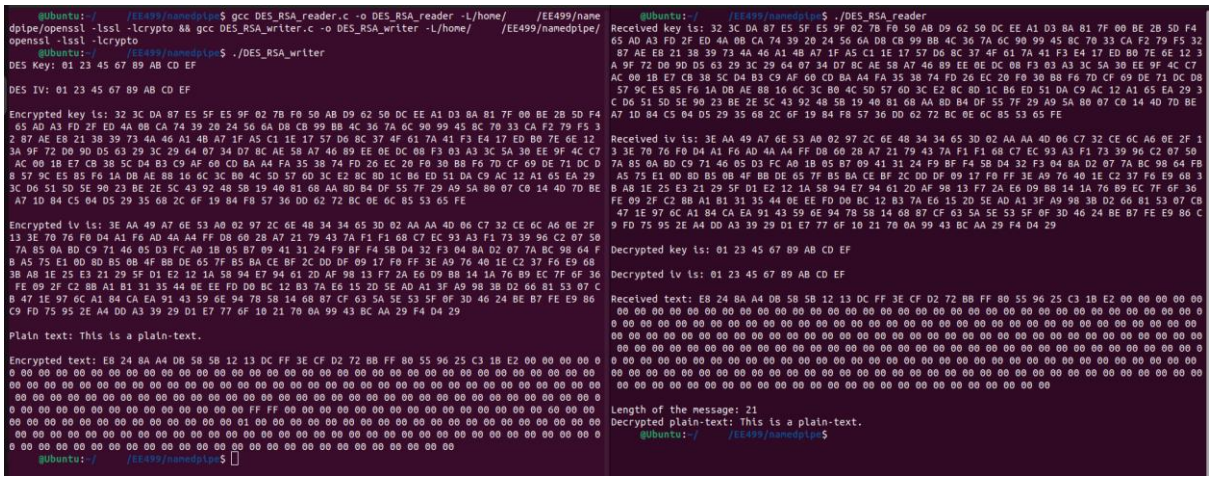

Figure 9. DES encryption with RSA secured key

Figure 10. AES encryption with RSA secured key

Measurements and Comparison of Algorithms

After the encryption algorithms are researched, developed and compiled, the last stage in the study is to run the programs with these algorithms in a Linux environment and make the necessary measurements. The running times of the developments in this study and the resource consumption at run time are recorded and a comparison is made between the encryption algorithms.

Time Measurement

The first measurement is to measure the time each cryptographic method spends on encryption. In order to measure this, 'time', a built-in function in the Linux environment, was used. The 'time' command expresses the time, in seconds, from the start to completion of the application run with it. On the other hand, although measuring an encryption process with the time command can give a value, a single value for comparison will be weak in terms of consistency. During this process, operations such as other applications running in the background, I/O waits, and network states cause deviations in the current time and performance. For this reason, a script was developed and run 1000 times for each encryption algorithm separately and averaged after saving each time output value in a file. In this way, the effect of deviations due to external factors is significantly reduced and consistency is increased. This script runs 1000 times for Caesar Cipher, DES and AES and transfers the results to a csv file. Then, a separate application called 'averager' presents the mean of these values as an output.

CPU Utilization Measurement

Another factor to be evaluated in the study is the CPU usage of the algorithms while they are running. Although all the algorithms in the study use RSA for key generation, the operations and structures they perform in each cycle are completely different, as explained above. For this reason, observing and comparing the use of resources will have an important role in the use of these algorithms. Just like in time measurement, data taken for a single trip will be insufficient in terms of consistency and accuracy. Both the difference in the resources the device allocates and uses for other processes at that moment and the difficulty of obtaining CPU values in a very short time have led to the need to take measurements during repetition many times. Each encryption algorithm was run 1000 times respectively with a prepared script and the values were saved in a csv file. The built-in function that this script uses to monitor CPU utilization is 'ps'. This command shows the list of processes actively running on that computer. By running the 'ps -p <pid> -o %cpu' command, the CPU usage of the process whose 'pid', that is, the process ID, is given among the processes listed with 'ps' can be observed.

Results and Discussion

Time Measurement Results

Time measurements were noted through scripts as described in the method section. Afterwards, the averages of each 100 iterations were taken and graphed using a Python script. On a single graph, the time spent by AES, DES and Caesar encryption methods during encryption can be observed. Values in this measurement are in milliseconds.

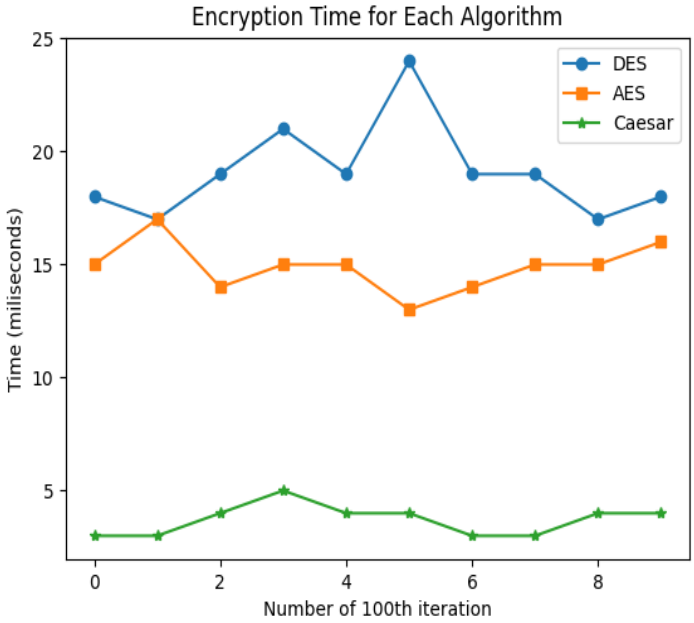

Figure 11. Time measurement graph

Table 1. Time measurement result recordings

	Caesar Cipher	DES	AES
Average of 1st 100 runs	3 ms	18 ms	15 ms
Average of 2nd 100 runs	3 ms	17 ms	17 ms
Average of 3rd 100 runs	4 ms	19 ms	14 ms
Average of 4th 100 runs	5 ms	21 ms	15 ms
Average of 5th 100 runs	4 ms	19 ms	15 ms
Average of 6th 100 runs	4 ms	24 ms	13 ms
Average of 7th 100 runs	3 ms	19 ms	14 ms
Average of 8th 100 runs	3 ms	19 ms	15 ms
Average of 9th 100 runs	4 ms	17 ms	15 ms
Average of 10th 100 runs	4 ms	18 ms	16 ms

According to the observed results, the Caesar encryption algorithm, which has a very simple working structure, completes transactions in 4-5 times shorter time compared to AES and DES. Data that can be easily encrypted can also be decrypted very easily through cyber-attacks. On the other hand, although the AES algorithm makes encryptions that are more difficult to decipher than DES, the running time of the DES algorithm is longer than AES, although there are not big differences. The small difference also varies depending on the size of the data to be encrypted. If very large data will be encrypted, it is appropriate to choose the AES algorithm.

CPU Utilization Measurement Results

CPU usage measurements were noted through scripts as described in the method section. Afterwards, 10 different measurements are done and the results are taken notes. Then, a Python script is used to visualize them. On a single graph, the utilized CPU by AES, DES, and Caesar encryption methods during encryption can be observed.

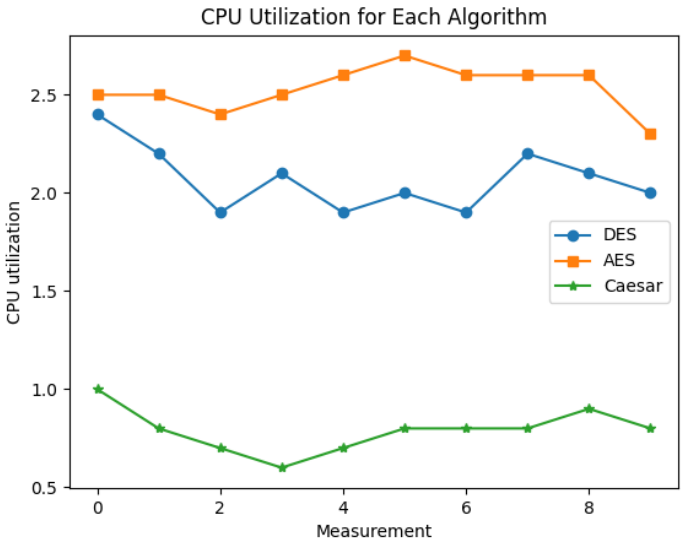

Figure 12. CPU utilization measurement graph

Table 2. CPU utilization measurement result recordings

	Caesar Cipher	DES	AES
1st measurement	1.0%	2.4%	2.5%
2nd measurement	0.8%	2.2%	2.5%
3rd measurement	0.7%	1.9%	2.4%
4th measurement	0.6%	2.1%	2.5%
5th measurement	0.7%	1.9%	2.6%
6th measurement	0.8%	2.0%	2.7%
7th measurement	0.8%	1.9%	2.6%
8th measurement	0.8%	2.2%	2.6%
9th measurement	0.9%	2.1%	2.6%
10th measurement	0.8%	2.0%	2.3%

According to the CPU usage measurement results, the Caesar encryption method uses minimum resources with its very simple structure, just like time measurements. It completes operations using less than half the CPU of AES and DES algorithms. To compare in terms of CPU usage, there is a difference of almost similar proportions to that in time measurements. The AES algorithm consumes slightly higher CPU during operations than DES, with a slight difference. This may determine the preference for platforms with very limited processing power.

To summarize the results, these algorithms that encrypt communication on named pipes responded as expected. The Caesar algorithm, which can be solved very easily, has the highest performance in terms of resource consumption and time, but provides a very low level of security. AES and DES can be preferred in different ways according to different restrictions when they want to be used in inter-process communication.

Conclusion

In conclusion, this study has provided a comprehensive analysis of encryption methods applied to named pipes within a Linux environment, focusing on RSA, Caesar Cipher, DES, and AES algorithms. Through meticulous examination and performance measurements, several key findings have emerged.The RSA algorithm, renowned for its asymmetric encryption capabilities, demonstrates robust security features suitable for various applications, including secure data transmission and digital signatures. Its reliance on complex mathematical operations, such as prime number factorization, ensures high levels of encryption strength. However, RSA's computational demands are notable, impacting performance metrics such as encryption time and CPU utilization.

Contrastingly, the Caesar Cipher, while simple and easy to implement, lacks the sophistication necessary for modern security standards. Its straightforward substitution method makes it vulnerable to frequency analysis and brute-force attacks. Nevertheless, the Caesar Cipher exhibits the lowest resource consumption and fastest encryption times among the algorithms studied, albeit at the expense of security.

The Data Encryption Standard (DES), a symmetric-key block encryption algorithm, offers historical insights into encryption fundamentals but falls short in contemporary security contexts. Its small key size and susceptibility to brute-force attacks render it obsolete compared to more robust alternatives like AES. Advanced Encryption Standard (AES), heralded as a cornerstone of modern cryptography, emerges as the preferred choice for secure communication over named pipes. AES balances strong encryption with efficient performance, making it suitable for diverse applications ranging from government to commercial use. With key sizes of 128, 192, and 256 bits, AES provides flexible security options tailored to specific needs while maintaining resistance against brute-force attacks. Performance measurements confirm AES's superiority in both encryption time and CPU utilization, albeit with marginal differences compared to DES. The Caesar Cipher, while significantly faster and less resource-intensive, lacks the requisite security for most applications.

The measurements in the study overlap with the inferences found in similar studies in the literature. Very similar results were obtained with the results in the study comparing AES and DES, not only on named pipes. In this study, which considers both time and resource consumption, it was concluded that AES is faster and uses more CPU than DES (Rihan et al., 2015). Obtaining similar results with this study conducted without a named pipe indicates that the named pipe has no relative impact on the algorithm performance.

In summary, the selection of encryption algorithm for named pipe communication hinges on a nuanced evaluation of security requirements, computational resources, and performance considerations. While the RSA algorithm offers unparalleled security, its computational overhead may limit practical applications. Conversely, the Caesar Cipher, while efficient, lacks the requisite security for modern encryption needs. DES, though historically significant, is overshadowed by AES's superior performance and security. Ultimately, AES emerges as the optimal choice, striking a balance between robust encryption and efficient resource utilization, ensuring secure communication over named pipes in Linux environments.

Scientific Ethics Declaration

The authors declare that the scientific ethical and legal responsibility of this article published in EPSTEM Journal belongs to the authors.

Acknowledgements or Notes

* This article was presented as an oral presentation at the International Conference on Technology (www.icontechno.net) held in Alanya/Turkey on May 02-05, 2024.

* We would like to express our sincere gratitude to Software Architect Hüseyin Karacalı for his extraordinary mentorship and inspiring influence. We would also like to thank TTTech Auto Turkey for their invaluable assistance in the development stages of this project.

References

Abdelrahman, A. A., Fouad, M. M., & Dahshan, H. (2017). Analysis on the AES implementation with various granularities on different GPU architectures. *Advances in Electrical and Electronic Engineering, 15*(3),526-535.

Adegeo. (n.d.). *How to use named pipes for network interprocess communication* Retrieved from https://learn.microsoft.com/en-us/dotnet/standard/io/how-to-use-named-pipes-for-network interprocess-communication

Bhat, B., Ali, A. W., & Gupta, A. (2015). DES and AES performance evaluation. *International Conference on Computing, Communication & Automation* (pp. 887-890). IEEE.

Daemen, J., & Rijmen, V. (2002). *The design of Rijndael: AES - the advanced encryption standard*. Springer.

Gaikwad, H. (2023, August 17). *What are named pipes in linux?.* Retrieved from https://www.scaler.com/topics/linux-named-pipe/

Hioureas, V. (2023, October 13). *Encryption 101: How to break encryption: Malwarebytes labs*. Retrieved from https://www.malwarebytes.com/blog/news/2018/03/encryption-101-how-to-break-encryption

Linux Manual Page. (n.d.). *MKFIFO (1*). Retrieved from https://man7.org/linux/man-pages/man1/mkfifo.1.html

Rihan, S. D., Khalid, A., & Osman, S. E. F. (2015). A performance comparison of encryption algorithms AES and DES. *International Journal of Engineering Research & Technology (IJERT), 4*(12), 151-154.

Schneier, B., & Diffie, W. (2015). *Applied Cryptography Protocols, algorithms, and source code in C.* John Wiley & Sons.

Sharmal, M., & Garg, D.R. (2016). DES: The oldest symmetric block key encryption algorithm. *International Conference System Modeling & Advancement in Research Trends (SMART),* 53-58.

Shorman, A., & Qatawneh, M. (2018). Performance Improvement of double data encryption standard algorithm using parallel computation. *International Journal of Computer Applications, 179*(25), 1–6.

Stallings, W. (2017). *Cryptography and network security: Principles and practice.* Pearson.

Stevens, W. R., & Rago, S. A. (2014). *Advanced programming in the unix environment.* Addison-Wesley.

Takieldeen, A., Awade, R., & Mahmoud, A. (2012). *Modern encryption techniqes of communication on networks* .(Master's thesis).Faculty of Engineering, Electronics &Communication Department.

Viega, J., Messier, M., & Chandra, P. (2002). *Network security with openSSL.* O'Reilly.

Wikimedia Foundation. (2022, April 15). *Named pipe.* Retrieved from https://en.wikipedia.org/wiki/Named_pipe

Wikimedia Foundation. (2024a, January 9). *AES*. Retrieved from https://tr.wikipedia.org/wiki/AES

Wikimedia Foundation. (2024b, March 25). *Advanced encryption standard.* Retrieved from https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Author Information

Huseyin Karacali
TTTech Auto Turkey,
Türkiye

Efecan Cebel
TTTech Auto Turkey,
Türkiye
Contact e-mail: e*fecan.cebel@tttech-auto.com*

Nevzat Donum
TTTech Auto Turkey,
Türkiye

To cite this article:

Karacali H. & Cebel E. & Donum N. (2024). Performance analysis of encryption algorithms in named pipe communication for Linux systems. *The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM), 27,* 214-227.