

Research Article

Cryptographic Enhancement of Named Pipes for Secure Process Communication

Huseyin KARACALI^{1*}, Nevzat DONUM^{2*}, Efekan CEBEL^{3*}

¹ TTTechAuto Turkey, Software Architect, Orcid ID: <https://orcid.org/0000-0002-1433-4285>, E-mail: huseyin.karacali@tttech-auto.com

² TTTechAuto Turkey, Embedded Software Engineer, Orcid ID: <https://orcid.org/0000-0002-8293-8267>, E-mail: nevzat.donum@tttech-auto.com

³ TTTechAuto Turkey, Embedded Software Engineer, Orcid ID: <https://orcid.org/0000-0002-2027-0257>, E-mail: efekan.cebel@tttech-auto.com

* Correspondence: efekan.cebel@tttech-auto.com

Reference: Karacali, H., Donum., N., Cebel, E. Cryptographic Enhancement of Named Pipes for Secure Process Communication. The European Journal of Research and Development, 4(2), 1-18.

Abstract

This study aims to enhance cryptographic security within the "named pipe" Inter-Process Communication (IPC) method utilized in Unix and Unix-like systems. Addressing security vulnerabilities inherent in the named pipe structure, this research endeavors to augment IPC security by integrating a cryptographic layer using the Advanced Encryption Standard (AES) encryption algorithm with 128-bit length key. The named pipe structure allows all processes specified by the owner or group to access data within the pipe. This implies the potential for processes generated by malicious software to access this data. By integrating a cryptographic secure communication structure into this bidirectional, readily applicable method, this study seeks to fortify the protocol. Technical objectives of this research involve encrypting messages with the AES-128 algorithm, enabling meaningful decryption of messages written with the same algorithm, and preventing interpretation of messages within the pipe by third-party processes lacking this cryptographic structure. Experimental findings showcase that when monitored by a process lacking this cryptographic structure, encrypted and incomprehensible messages are displayed, indicating the resistance of the encrypted structure against external interventions. In conclusion, this study introduces a new method to enhance security in IPC by adding a cryptographic security layer to the named pipe. This research may have implications for IPC security in Unix systems and could be applicable to other IPC methods facing similar security vulnerabilities.

Keywords: inter-process, communication, network, security, cryptography, Linux, AES-128

1. Introduction

A named pipe, which is also recognized as a FIFO (First-In, First-Out), represents a unique file alike to a pipe, yet distinguished by its filesystem-assigned name. Numerous processes have the capability to interact with this distinct file for both reading and writing, functioning similarly to a regular file. Consequently, the name solely serves as a locator for processes requiring a named reference within the filesystem. In essence, a FIFO bears resemblances to other files in terms of its attributes; for instance, it possesses ownership, permissions, and metadata. Furthermore, a notable attribute of a FIFO is its facilitation of bidirectional communication [1].

The identified security flaw pertains to the inherent nature of named pipes as file-based communication channels, susceptible to unauthorized access and potential exploitation by malicious software.

One part of the problem is related to file access. Named pipes, despite serving as communication channels between processes, are implemented as special files within the file system. These files can be accessed and potentially manipulated by any software or user with appropriate permissions. Since the lack of access control mechanism for named pipes, once created, they may inherit permissions from the parent directory, potentially allowing unintended read or write access by unauthorized users or programs.

The problem is also that there is no inherit encryption method for inter-process communication. The data transmitted through named pipes is not encrypted by default. Any malicious software or unauthorized entity gaining access to the named pipe file can intercept, view, and potentially manipulate the transmitted information.

Moreover, the problem can be defined as a potential information leakage. As named pipes are accessible as regular files, any malware or unauthorized application with access to the pipe file can eavesdrop on the communication, leading to a severe breach of data confidentiality.

This problem can cause many impacts. Data exposure is one of the biggest impacts of the specified problem. Sensitive information transmitted through named pipes can be intercepted and exposed to unauthorized parties, compromising data confidentiality. Another one is the privacy breach. The vulnerability poses a significant risk to the privacy of the communicated data, potentially leading to unauthorized data tampering or misuse. Each of these poses a security risk. Malicious entities gaining access to named pipes can exploit this vulnerability to conduct espionage, extract sensitive information, or execute further attacks within the system.

To approach it from a sectoral perspective and give an example, in applications or software that run Unix-based operating systems used in the automotive industry, data about the vehicle or driver, depending on the type of application, is made through inter-process communication structures such as SOME/IP, Netlink and named pipes. Therefore, the problem of security vulnerabilities in IPC structures comes to the fore in critical issues such as theft of personal data.

There have been many approaches and developments regarding security in computer science and especially in networking. There has been no such direct security integration within the scope of named pipes, but some security-related work has been done in inter-process communication and using them.

In order to detect unauthorized access in the Android operating system, which has a Linux-based structure and is on billions of devices around the world, studies have been carried out that use inter-process communication and provide security in this system [2].

Almost all functions of the microkernel occur through inter-process communication, and certain security issues are encountered in the process. Transmission of an unsecured message is one of them. Therefore, the handshake mechanism is integrated into this structure and the communication process is secured [3].

Another secure inter-process communication system is implemented with a system called SkyBridge, which performs operations through virtual address spaces without kernel participation. IPC is provided with hardware called VMFUNC. Analyzes have been made against malicious EPT switching, DoS attacks and Malicious Server Calls [4].

One of the studies that is closest in content to our own work is to increase security by using FIFO in pipelines [5]. At this point, it should be noted that named pipes are also FIFOs. This work improves the avoidance of external attacks by adding FIFOs between two pipelines and changing their durations randomly. Additionally, it is similar to this study with its choice of AES encryption algorithm [5].

A study was also carried out that prevents DDoS attacks and the system from going offline by using AES and XOR cipher separately and provides secure inter-process communication with a novel technique. In this study, different message encryption methods were used and illustrated in IPC [6].

Utilizing AES-128 encryption within the framework of named pipes communication represents a crucial step in addressing the inherent security vulnerability. By implementing this encryption standard, end-to-end security is fortified, ensuring data

confidentiality throughout transmission. AES-128 encryption safeguards against unauthorized access, thwarting potential attackers even if they gain access to the named pipe file. This approach not only aligns with regulatory compliance but also fortifies systems against evolving threats, enhancing trust, and credibility by demonstrating a commitment to robust data security practices. Overall, integrating AES-128 encryption stands as a pivotal measure, fortifying named pipes against exploitation and establishing a secure foundation for inter-process communication.

2. Materials

2.1. Ubuntu

Ubuntu, a widely used Linux distribution, has gained prominence in both desktop and server environments due to its robust architecture, security features, and open-source nature. This section provides a detailed examination of the technical aspects that make Ubuntu a preferred choice in the realm of operating systems [8]. Ubuntu is built upon the Linux kernel, which serves as the core component responsible for managing system resources, providing hardware abstraction, and facilitating communication between software and hardware. The modular and well-maintained nature of the Linux kernel ensures compatibility with a wide range of hardware, contributing to Ubuntu's versatility. One of the distinctive features of Ubuntu is its advanced package management system [8]. Utilizing the Debian package format, Ubuntu employs the Advanced Package Tool (APT) for streamlined installation, removal, and updating of software packages. The dependency resolution mechanism ensures a stable and coherent software ecosystem, enhancing the overall reliability of the operating system [9]. Ubuntu places a strong emphasis on security, implementing various measures to safeguard user data and system integrity [9]. The mandatory access control framework, AppArmor, restricts the capabilities of individual applications, reducing the potential impact of security breaches. Additionally, regular security updates through the unattended-upgrades mechanism contribute to maintaining a secure and up-to-date system. The default desktop environment for Ubuntu is GNOME, providing an intuitive and user-friendly interface [8]. Ubuntu's commitment to the Unity user interface in previous versions showcased its dedication to creating a cohesive and aesthetically pleasing user experience. The customization options available within the desktop environment allow users to tailor their workspace according to their preferences.

Ubuntu's open-source philosophy aligns with the academic ethos of knowledge sharing and collaboration. The active community and transparent development process provide students and researchers with opportunities to contribute to open-source projects, promoting a culture of continuous learning and skill development. In academic settings,

Ubuntu's accessibility, cost-effectiveness, and community support make it an ideal choice for educational institutions. The availability of a vast repository of open-source software and development tools facilitates teaching and research activities across various disciplines. Therefore, this study has been developed and implemented on Ubuntu 22.04.

2.2. Inter Process Communication (IPC)

IPC encompasses a set of mechanisms facilitating data, information, or command exchange between different processes within computer systems. These mechanisms are employed to facilitate interaction between processes, enable information sharing, and enhance coordination [9]. Forking, a fundamental method of IPC, involves a process creating a new one by duplicating the same program code. The created process may share the same memory space with the parent process but can have independent execution control. Pipes are a simple IPC mechanism used for transmitting data produced by one process to another [9]. Communication occurs between the output of one process and the input of another. Message passing, another prevalent IPC mechanism for data exchange between processes, enables them to send and receive messages following a specific protocol. Shared memory allows processes to share the same memory space, enabling them to read and write data in a common memory area. However, managing the security and synchronization of this mechanism is imperative [9]. Sockets represent another significant mechanism in IPC. They facilitate data communication between processes over a network or between processes on the same computer. This mechanism is particularly widespread in distributed systems.

IPC enables the development of collaborative applications through data sharing among different processes [10]. For instance, data exchange between a text editor and a graphic program. In the context of multi-processor systems, IPC is utilized for parallel processing and coordination [10]. Data exchange and synchronization among multiple processors are achieved through IPC mechanisms. In distributed systems, IPC is employed to establish communication between different computers. This is commonly accomplished through sockets or other network-based IPC mechanisms. IPC plays a crucial role in supporting collaboration, parallel processing, and communication in various computing environments.

In this study, Named Pipe has been utilized as the IPC mechanism. Named Pipe is a communication channel with a designated name in the file system, facilitating data transfer between processes [11]. It operates on the FIFO (First In, First Out) principle, meaning data is transmitted in the order of entry [11]. When one process writes data through a Named Pipe, another process can read the data from the same pipe. Named

Pipe is commonly employed in Unix and Unix-like operating systems, but it is also supported in other operating systems such as Windows.

The initial step in using Named Pipe involves its creation by a process and assigning a name. This name is defined within the file system, and other processes access the Named Pipe using this designated name [12]. The created Named Pipe enables data communication between one or multiple processes. While one process writes data to the pipe, another process can read this data. Following the FIFO logic, data is processed in the order of entry [13]. Upon completion of communication via Named Pipe, processes can terminate this interaction and delete the Named Pipe. This capability allows for effective resource management. Named Pipe is employed to facilitate data sharing among collaborative applications. In summary, Named Pipe serves as an IPC mechanism, providing a named communication channel for data exchange between processes [13]. Its implementation and utilization contribute to effective inter-process communication in various computing environments. The fundamental working principle of the named pipe is illustrated in Figure 1 [13].

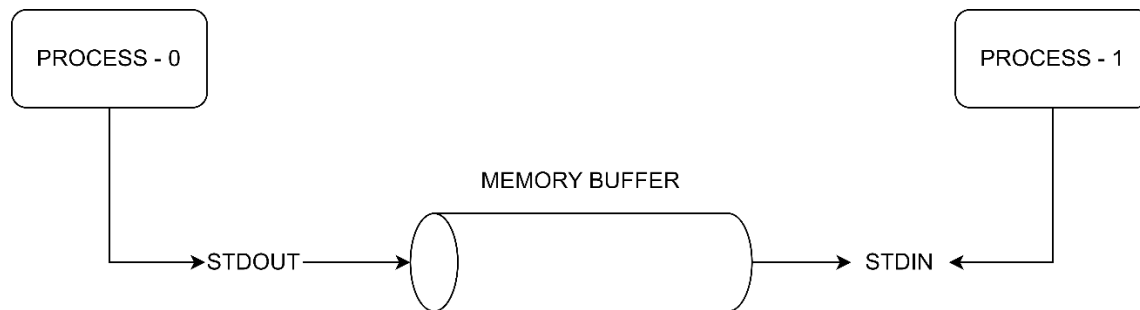


Figure 1: Fundamentals working principle of named pipe

2.3. Advanced Encryption Standard (AES)

AES is a cryptographic standard established by the National Institute of Standards and Technology (NIST) in the United States in 2001, replacing the Data Encryption Standard (DES). It serves a crucial role in ensuring the secure transmission and storage of data through symmetric encryption [14]. AES offers three versions based on key lengths: AES-128, AES-192, and AES-256, supporting 128, 192, and 256-bit key lengths, respectively. The security level of the algorithm increases with longer key lengths, with the number of rounds determined by the key length playing a pivotal role in encryption and decryption processes [14]. Fundamentally built upon the Rijndael algorithm, AES employs a block cipher approach to process data blocks of 128-bit size. The core process involves key expansion, SubBytes, ShiftRows, MixColumns, and AddRoundKey stages, ensuring

robust encryption through mathematical transformations [14]. The key expansion phase transforms the user-input key into round-specific subkeys, enhancing cryptographic security by increasing key complexity. Executed using specific constant values and functions of the Rijndael algorithm, the key expansion process is tailored to key length and the number of rounds. These stages unfold consecutively in each round. SubBytes performs a substitution operation using the S-box. ShiftRows transposes matrix elements by shifting them within rows. MixColumns shuffles columns using matrix multiplication, while AddRoundKey applies round keys to the data block. The combination of these stages ensures the secure encryption of data blocks [14]. AES is recognized for its resilience against various cryptanalysis methods, providing robust protection against advanced attacks such as differential and linear cryptanalysis. Additionally, the algorithm's structure ensures that an increase in key length directly influences the security level [15].

In this study, the utilization of the AES-128 algorithm is evident. AES-128 ensures a high level of security, boasting resilience against a multitude of cryptographic analysis methods due to its 128-bit key length, effectively protecting data from potential malicious attacks. In terms of speed and efficiency, AES-128 demonstrates remarkable effectiveness by swiftly executing data encryption and decryption processes with minimal computational overhead on the processor [15]. Additionally, AES-128 stands out for its ability to seamlessly integrate into various programming languages and cryptographic libraries, further enhancing its practicality. Figure 2 shows how the AES-128 encryption algorithm works in a simple way. Figure 3 shows how the AES-128 decryption algorithm works in a simple way.

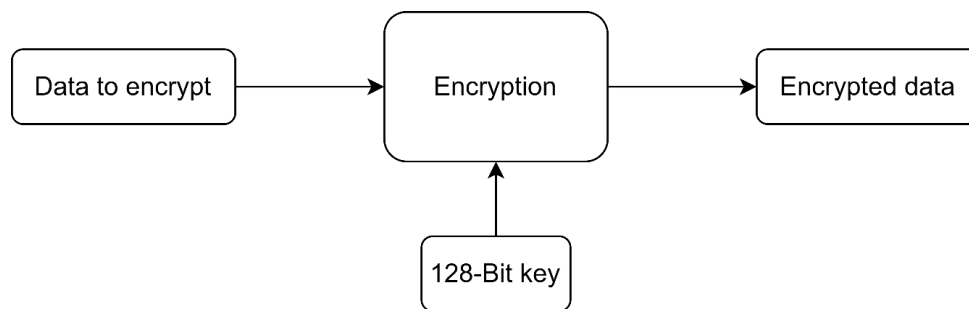


Figure 2: AES-128 encryption basic block diagram

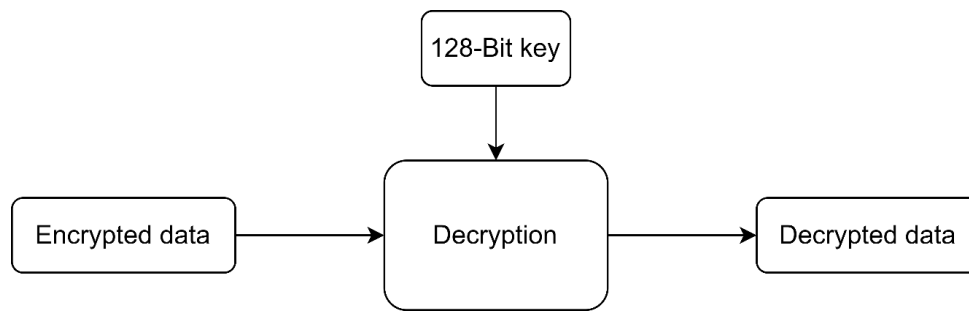


Figure 3: AES -128 decryption basic block diagram

2.4. OpenSSL

OpenSSL stands as an open-source cryptographic library and toolkit, functioning as a diverse set of utilities designed for securing communication and executing various security tasks [16]. These tasks encompass encryption, management of digital certificates, security protocols, and key management. OpenSSL's versatility extends across numerous platforms, and it provides support for various widely adopted security protocols.

Being an open-source initiative, OpenSSL makes its source code available for examination, modification, and distribution by any interested party. This accessibility empowers security professionals and developers to thoroughly review the code and implement enhancements [16].

OpenSSL is primarily crafted for cryptographic operations and encryption. By employing various cryptographic algorithms, it facilitates the safe transfer of data [16]. This capability empowers users to encrypt their information to ensure confidentiality, enhance authentication procedures, and generate digital signatures.

Utilized for the implementation of secure communication protocols like SSL and TLS, OpenSSL guarantees the secure transmission of data over the internet. Specifically, TLS, widely employed in interactions between web browsers and servers, serves to encrypt and safeguard internet data [17].

OpenSSL is utilized in the generation and administration of digital certificates, a crucial element in authenticating and establishing the reliability of websites. OpenSSL streamlines the creation, modification, and interaction with certificate authorities.

OpenSSL is frequently employed in the generation and administration of secure key pairs, crucial components in performing encryption and digital signature functions. Additionally, OpenSSL facilitates the secure storage and exchange of these keys.

OpenSSL exhibits compatibility with a multitude of operating systems and platforms, granting developers the flexibility to deploy their applications in diverse environments. In the context of data security and encryption, the AES-128 library within OpenSSL stands out as a sturdy and dependable solution. OpenSSL has earned a distinguished reputation for its unwavering commitment to security, having undergone rigorous testing and widespread adoption in various security-critical applications over numerous years [18]. Notably efficient, OpenSSL's AES-128 implementation leverages hardware acceleration and optimized code, ensuring that encryption and decryption operations are executed with minimal computational overhead [18]. This swiftness proves crucial in scenarios demanding rapid data processing without compromising security. Additionally, OpenSSL's AES-128 library is cross-platform, offering versatility for a wide array of applications. Whether developing software for diverse operating systems or integrating encryption into a network device, OpenSSL ensures compatibility across different platforms [18]. The AES-128 library within OpenSSL empowers developers with the adaptability to customize encryption according to their specific requirements. Supporting various modes of operation (e.g., ECB, CBC, GCM) and key sizes, it enables tailoring the encryption process to align with the needs of individual applications.

3. Method

The initial phase of this study involved meticulously employing the 128-bit variant of the Advanced Encryption Standard (AES) to ensure the robust security and integrity of the input data. AES encryption stands out as an algorithm that utilizes a series of complex mathematical operations to transform plaintext data into ciphertext, rendering it virtually unrecognizable [19]. Decrypting this ciphertext without the appropriate decryption key poses an exceptionally challenging task. AES-128, representing a specific iteration within the AES encryption framework, is primarily distinguished by the key size employed during the encryption process. In this context, a 128-bit key was strategically chosen, striking a practical balance between computational efficiency and fortified data security. This key is generated from a cryptographically secure random source, ensuring its robustness against potential attacks [19].

The AES encryption procedure unfolds through a series of highly intricate stages. The initial stage, referred to as the sub-bytes step, intricately replaces each byte in the state matrix with a byte derived from a meticulously pre-defined 8-bit look-up table known as the Rijndael S-box. Subsequently, the rows shifting step ensues, involving cyclic shifts of bytes within each row of the state [19]. Following this, the mixing columns process intricately combines the four bytes within each column of the state, employing a linear transformation that replaces each byte with a value contingent upon all four bytes in the

column, utilizing arithmetic in a finite field. Finally, the addition of the rounded key step involves XOR operation the state with the round key—an elegantly simple operation of bitwise addition that is easily implementable in both hardware and software environments. The sophistication of the AES encryption process, characterized by these multifaceted stages, underscores its efficacy in providing robust data protection and confidentiality [19].

Comprising a singular iteration of operations, these stages are repeated tenfold within the AES-128 encryption process, excluding the column mixing operation specifically in the final round. Within the encryption domain, each iteration employs a unique round key, derived through the Key Expansion routine originating from the initial input key. This routine intricately integrates galois field multiplication and a central process involving byte-wise shift and substitution operations, ensuring the generation of essential round keys [19]. The intricacies of the operations collectively delineate the AES-128 encryption process. This systematic procedure systematically disrupts the arrangement of input data in a manner that is both methodical and highly entropic. Consequently, the data undergoes a transformation rendering it impervious to unauthorized access or breaches. Only an entity equipped with the precise decryption key possesses the capability to successfully restore the encrypted text to its original format.

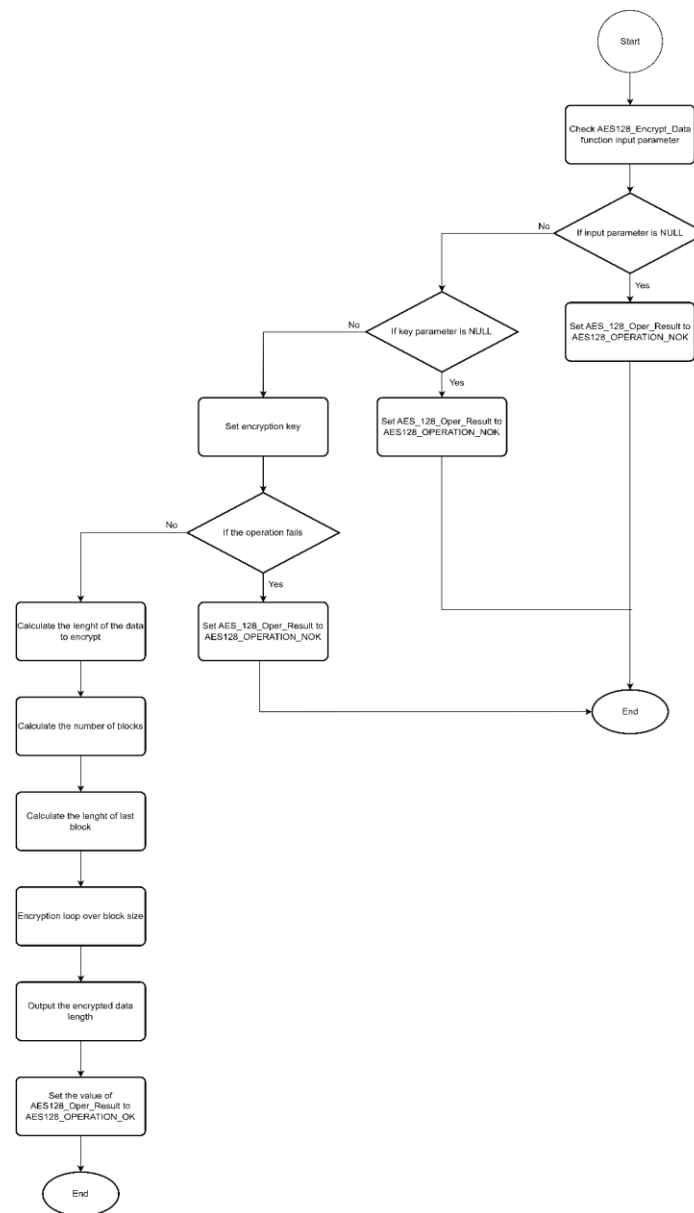


Figure 4: AES-128 encryption flowchart

Figure 4 illustrates a flowchart detailing the encryption procedure implemented in this research [19]. Irrespective of the input data's nature, size, or intended application, it is provided as an input parameter to the AES128_Encrypt_Data function [19]. Following a confirmation of the input's existence, the program proceeds to validate the designated key for encryption. If a valid key is identified, the encryption process initiates. Upon successful execution of the AES-128 algorithm steps without encountering errors, the encrypted data is extracted from the encryption block and presented as output for further

compression [19]. In the absence of a valid key or if errors arise during any stage of the process, the system responds with AES128_OPERATION_NOK [19].

Decryption, the inverse process of encryption, follows the same sequence of steps but in reverse order to transform the unreadable ciphertext back into its original plaintext [19]. In the context of the AES 128-bit decryption algorithm, it revisits each encryption round in a retrograde pattern [19]. When supplied with the same secret key used during encryption, the AES-128 decryption algorithm accurately reproduces the original data. The decryption process encompasses several key steps.

Initiating with the "adding rounded key" step, which involves the bitwise XOR operation of the ciphered data with the expanded round key, the AES decryption process mirrors the initial encryption step. However, the key schedule is executed in reverse. The subsequent step involves the inverse of the row-shifting process performed during encryption. Each row in the state undergoes a cyclic shift towards the right, with the number of places shifted varying for each row. This step effectively reverses the permutation conducted during encryption. The next decryption step is the inversion of sub bytes, entailing the transformation of bytes using the inverse of the S-box employed in encryption. An inverse substitution for each byte of the block is executed to derive the original data. The final step involves the inversion of mixed columns, undoing the effect of mixed columns during encryption. The outcome is a matrix whose column entries result from the inverse linear transformation of corresponding column entries in the input state.

These sequential processes collectively constitute a decryption phase, reiterated for the same number of rounds as initially applied during encryption. In the case of AES-128, specifically, ten rounds of decryption are required, with the crucial observation that the inversing mixed columns operation is omitted in the final round.

Upon completion of these intricate steps, the output materializes as the original plaintext data that underwent encryption. Consequently, the AES-128 decryption method adeptly recovers the initial data without compromising its integrity.

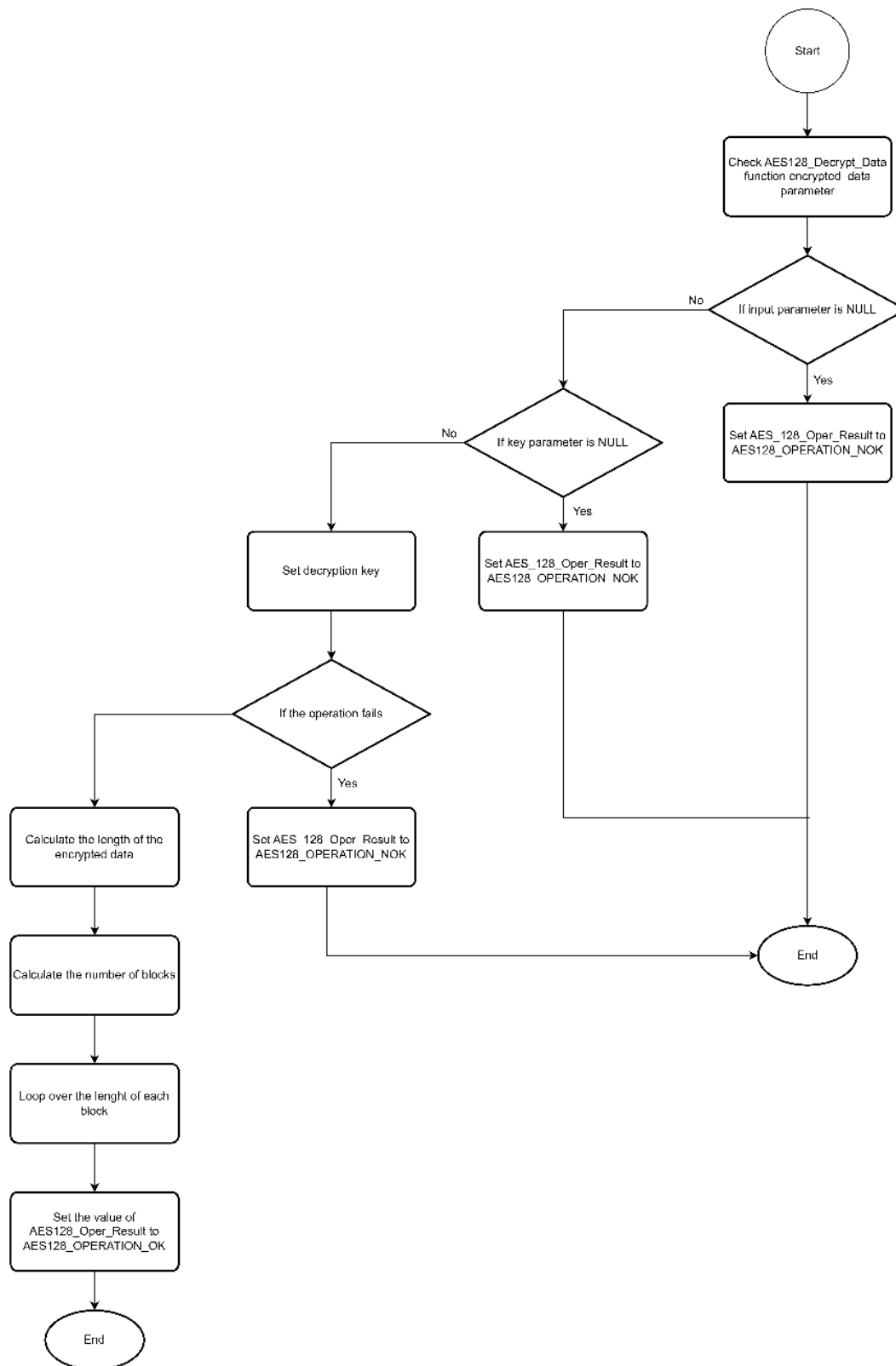
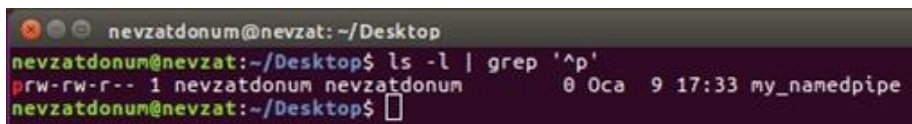


Figure 5: AES-128 decryption flowchart

Another key component of this work is to create the named pipe and ensure its integration with the processes that will use it. Named pipes stand as a crucial tool for facilitating inter-process communication within operating systems. However, unlike a regular file, a named pipe is identified by a unique name rather than being tied to a file system. This allows communication between different processes through a designated, pre-established name. While commonly found in Unix and Unix-like systems, they also provide similar functionality in Windows operating systems. Named pipes serve the purpose of easing data transmission between different processes and fostering interaction among them.

Named pipes offer bidirectional communication capabilities, allowing both reading and writing operations on the communication channel. This feature supports mutual data exchange between processes. Named pipes provide a reliable mechanism for data transmission. Data transfer between processes occurs over a reliable sequence, ensuring the integrity and accuracy of the transmitted data. Moreover, they can be used in distributed systems and collaboration.

In this study, creating a single named pipe is sufficient to implement and demonstrate the system. In total, there will be a named pipe, a sending process, a receiving process, and a malicious process. To create a named pipe, a FIFO, the 'mkfifo' command is used with a desired name for this pipe. In this study, it is called my_namedpipe during creation. It can be confirmed the successful creation of named pipe by using 'ls' command on the terminal whose current path is same directory where the named pipe was created as shown below in Figure 6.



```
nevzatdonum@nevzat: ~/Desktop
nevzatdonum@nevzat:~/Desktop$ ls -l | grep '^p'
prw-rw-r-- 1 nevzatdonum nevzatdonum 0 0ca 9 17:33 my_namedpipe
nevzatdonum@nevzat:~/Desktop$
```

Figure 6: Created named pipe in Unix system

By following the above steps, a named pipe is created in the Unix system. Any two or more processes running on a Unix system can communicate via this FIFO.

Another elements to be used in the study are processes. In order to both use and test FIFO, two processes were created using the C programming language. In order to better convey the operation, one is designed as a sender and the other as a receiver. Figure 7 represent the processes that are running on the Unix system.


```
nevzatdonum@nevzat:~$ ps aux | grep _process
nevzatd+ 9871  0.0  0.0  10796  1224 pts/20  S+   18:04   0:00  ./transmitter_process
nevzatd+ 9915  0.0  0.0  10664  1272 pts/21  S+   18:04   0:00  ./receiver_process
nevzatd+ 10041 0.0  0.0  15628   928 pts/22  S+   18:05   0:00  grep --color=auto _process
nevzatdonum@nevzat:~$
```

Figure 7: Running transmitter and receiver processes on Unix-based system

These processes are executables of codes written in the C programming language. The common feature of these programs is that the named pipes are a file in the file system and they open this pipe (my_namedpipe) in order to read and write as a file. Later, the above-mentioned AES-128 cryptography algorithm was integrated in all processes. Both the transmitter, receiver and malicious process have AES-128 encryption and decryption features. The difference here is that the receiving process has the correct key to access the original message.

4. Results

The aim of this study is to prevent the data transmitted between processes from being viewed meaningfully by a malware or out-of-scope process. To ensure security within the scope of both integrity and confidentiality concepts. In this case, an image representing the situation in which this structure is provided is presented. Figure 6 shows three different terminals open on a Unix-based computer. These three different terminals contain the processes required to define and demo the system. The sending process that will write the data to the pipe is in the terminal on the left side of the screen, the receiving process that can read the data from the pipe and decrypt it successfully is in the terminal on the upper right side, and finally the malicious process that will not be able to make sense of the data after receiving it from the pipe is in the terminal on the lower right side. is shown.

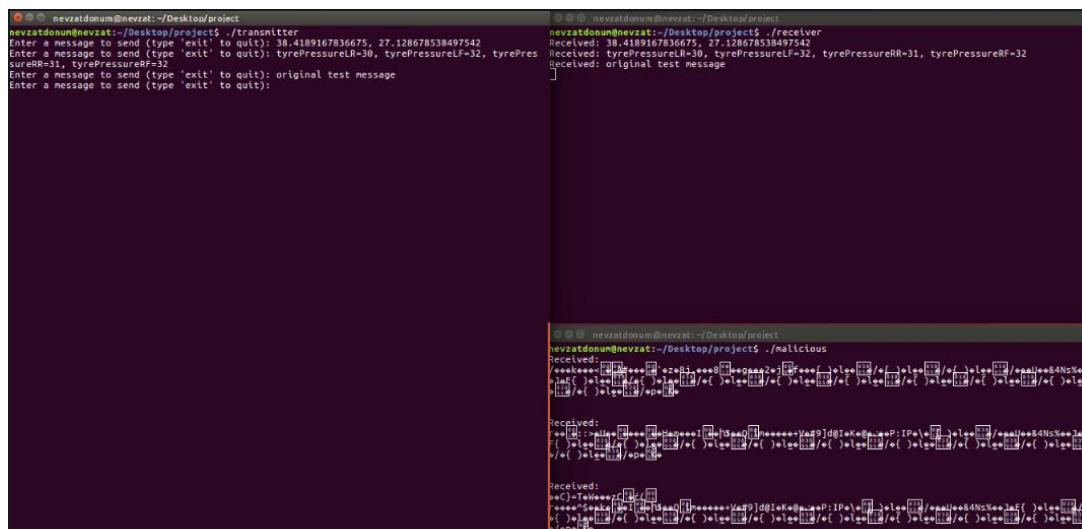


Figure 6: A snapshot of the integrated secure named pipe system

In the demo, three different messages were written to the created pipe by the sender. These data are defined as a coordinate address (Izmir Konak Clock Tower coordinates), four sample tire pressure values on the vehicle and a test string message. These three different posts were written separately on pipe.

As can be seen in Figure 6, the values written to the pipe in an encrypted form by the transmitter process can be read and decrypted by the receiver process. The receiver can interpret this data exactly as it is transmitted from the sending terminal. In this case, the system works as expected. However, although the malicious process can read this data encrypted from the relevant pipe, it cannot decipher this message because it does not have the appropriate password. It is designed assuming that the malicious process in the demo does not have any password data, so the malicious process prints the data it reads to the screen as it is. In this case, encrypted data is also observed. Since this is completely meaningless and cannot be processed by the process, the security of the data is ensured between processes.

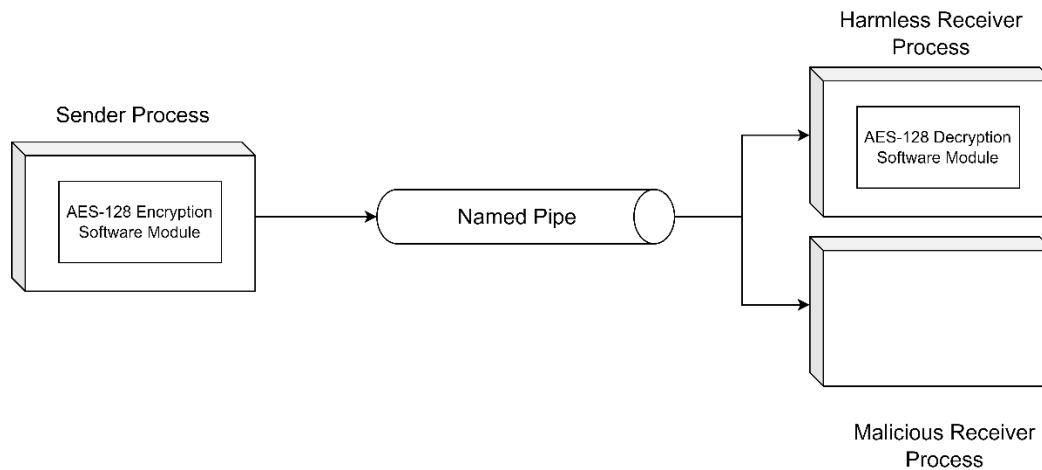


Figure 7: General overview of secured named pipe and processes structure

Figure 7 represents the general structure of the work carried out within the scope of the project. The system that is being implemented and the components of this system are presented in Figure 7 as an overview. All processes have the encryption and decryption algorithm with the AES-128 algorithm, but not all of them have the correct key. In this figure, only the software parts of the modules used are indicated.

5. Discussion and Conclusion

When the outcomes described in this study are observed, it can be concluded that the system created aligns with the anticipated objectives set at the study's outset and proves to be suitable for its intended purpose. This communication, which occurs without any bit loss between the processes in which the AES-128 cryptography system is integrated, cannot be decrypted by other processes that do not have this encryption or the same key value. This is completely in line with the idea of ensuring confidentiality, which is the aim of the project.

In the messages sent in the test part, it was highlighted that encryption can be done in different variable types by sending both numerical values (coordination data) and written expressions. Again, at this point, a sample content was selected based on Unix systems and inter-process communication, which are used in a sector such as automotive, where security is at the forefront, and referring to GPS location data, which is frequently shared in the automotive industry.

In literary terms, it supports studies that have previously worked on security with hardware or other studies. It contributes to process security that protects the kernel or tries to be provided on Android devices.

It is a study that is very suitable for development both in terms of literature and practice. This system prevents malicious processes from leaking data from the system and manipulating data. If an acknowledgement mechanism is created in future studies, malicious processes can not only access messages but also be detected and automatically terminated by the system. Processes that understand the content and those that cannot be distinguished through expected acknowledgment messages sent at regular intervals during messaging.

6. Acknowledge

We want to express our deepest gratitude to Huseyin Karacali, the Software Architect, for his outstanding guidance and motivating impact. Additionally, we wish to recognize the invaluable support provided by TTTech Auto Turkey during the various phases of project development.

References

- [1] baeldung, W. by: (2020, October 20). Anonymous and named pipes in linux. Baeldung on Linux. <https://www.baeldung.com/linux/anonymous-named-pipes>
- [2] R. Lemos, T. Heinrich, C. A. Maziero and N. C. Will, "Is It Safe? Identifying Malicious Apps Through the Use of Metadata and Inter-Process Communication", 2022 IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 2022, pp. 1-8, doi: 10.1109/SysCon53536.2022.9773881.

- [3] M. Asif, M. M. Iqbal, M. U. Khalid, Y. Saleem, "SECURING THE MESSAGE PASSING IN INTER PROCESS COMMUNICATION OF A MICROKERNEL", *Sci.Int(Lahore)*, 26(5), 2103-2106, 2014
- [4] Zeyu Mi, Dingji Li, Zihan Yang, Xinran Wang, and Haibo Chen. 2019. SkyBridge: Fast and Secure Inter-Process Communication for Microkernels. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 9, 1–15. <https://doi.org/10.1145/3302424.3303946>
- [5] K. J. Lin, C. P. Weng and T. K. Hou, "Enhance hardware security using FIFO in pipelines," 2011 7th International Conference on Information Assurance and Security (IAS), Melacca, Malaysia, 2011, pp. 344-349, doi: 10.1109/ISIAS.2011.6122844.
- [6] A. E. M. Eljaily, Sultan Ahmad, "A Novel Technique to Secure Inter-Process Communication," *IJCSNS International Journal of Computer Science and Network Security*, VOL.22 No.9, September 2022
- [7] "Ubuntu PC operating system | Ubuntu," *Ubuntu*. <https://ubuntu.com/desktop>
- [8] Educative, "Educative Answers - trusted answers to developer questions," *Educative*. <https://www.educative.io/answers/what-is-ubuntu-linux>
- [9] H. Dinari, "Inter-Process Communication (IPC) in Distributed Environments: An Investigation and Performance Analysis of Some Middleware Technologies," *International Journal of Modern Education and Computer Science*, vol. 12, no. 2, pp. 36–52, Apr. 2020, doi: 10.5815/ijmecs.2020.02.05.
- [10] "The Interprocess Communication (IPC) overview." <https://www.ibm.com/support/pages/interprocess-communication-ipc-overview>
- [11] I. Clough and N. Bergmann, "Using Linux FIFOs to allow Flexible Hardware/Software Communications on Reconfigurable Systems-on-Chip," *FSP Workshop 2018; Fifth International Workshop on FPGAs for Software Programmers*, Dublin, Ireland, 2018, pp. 1-2.
- [12] "Introduction to named Pipes | Linux Journal." <https://www.linuxjournal.com/article/2156>
- [13] H. Gaikwad, "What are Named Pipes in Linux? - Scaler Topics," *Scaler Topics*, Aug. 17, 2023. <https://www.scaler.com/topics/linux-named-pipe/>
- [14] Daemen, J., & Rijmen, V. (2000). The block Cipher Rijndael. In *Lecture Notes in Computer Science* (pp. 277–284). https://doi.org/10.1007/10721064_26
- [15] Dworkin, M. J. (2023). Advanced Encryption Standard. <https://doi.org/10.6028/nist.fips.197-upd1>
- [16] OpenSSL Foundation, Inc. (n.d.). [/index.html](https://www.openssl.org/). <https://www.openssl.org/>
- [17] OpenSSL Foundation, Inc. (n.d.-a). [/docs/man3.1/man7/crypto.html](https://www.openssl.org/docs/man3.1/man7/crypto.html). <https://www.openssl.org/docs/man3.1/man7/crypto.html>
- [18] Kekayan. (2018, July 7). Encrypt files using AES with OPENSSL - Kekayan - Medium. <https://kekayan.medium.com/encrypt-files-using-aes-with-openssl-dabb86d5b748>
- [19] H. Karacali, N. Dönüm, and E. Cebel, "Secure and efficient NVM usage for embedded systems using AES-128 and Huffman Compression," *The European Journal of Research and Development*, vol. 3, no. 4, pp. 333–356, Dec. 2023, doi: 10.56038/ejrmd.v3i4.281.