*Research Article*

# Secure and Efficient NVM Usage for Embedded Systems Using AES-128 and Huffman Compression

**Huseyin KARACALI[1*], Nevzat DONUM[2*], Efecan 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: efecan.cebel@tttech-auto.com
[*] Correspondence: efecan.cebel@tttech-auto.com

(First received September 17, 2023 and in final form December 26, 2023)

**Reference:** Karacalı H., Donum, N., Cebel, N. Secure and Efficient NVM Usage for Embedded Systems Using AES-128 and Huffman Compression. The European Journal of Research and Development, 3(4), 333-356.

## Abstract

*Embedded systems are customized systems designed to meet functionality and specific usage purposes. These systems often grapple with challenges such as power interruptions, limited memory space, and the expectation of a long operational lifespan. It is in this context that Non-Volatile Memory (NVM) plays a vital role. NVM is a type of memory that ensures data persistence even in situations of temporary power loss or signal interruptions. Efficiently utilizing NVM is a critical necessity for the effectiveness and reliability of these systems. Efficient utilization of NVM necessitates the effective management of write and read cycles. Previous studies have demonstrated the efficient utilization of NVM sectors by ensuring that no garbage bytes remain unallocated. Previous studies have demonstrated the efficient utilization of NVM by compressing the data to be written into NVM using the Huffman compression algorithm, and filling sectors without any residual garbage bytes. This paper aspires to advance beyond prior research by integrating the Advanced Encryption Standard (AES-128) block encryption algorithm, aiming to optimize the utilization of NVM to a greater extent. The AES-128 algorithm provides data security with encryption and decryption operations. By applying AES-128 encryption to the data before writing it to the NVM, this work adds a data security layer to the developed software module. Data is divided into blocks using this algorithm. Each block is encrypted independently and then reassembled. The encryption process includes key management, substitution-permutation network layers, and multiple rounds, all of which make substantial contributions to high-level data security. The resulting encrypted data is compressed using the Huffman compression algorithm. This process enhances both the security and efficiency of data storage. The data is written to NVM with maximum efficiency, ensuring that no residual garbage bytes remain, upon the completion of the encryption and compression processes. This study surpasses previous research*

*by enhancing storage security and efficiency through these processes. Moreover, the developed secure and efficient NVM usage, as it fills the NVM with data without leaving any garbage bytes, results in slower processing speed compared to standard NVM usage. Therefore, in future research, interventions with code optimization techniques can be applied to the developed algorithms to enhance write and read speeds.*

**Keywords:** non-volatile memory, memory efficiency, Huffman compression algorithm, Advanced Encryption Standard, AES-128

## 1. Introduction

In recent years, the growing demand for data storage solutions has led to significant advancements in NVM technologies. As these technologies evolve, the need for robust security measures and enhanced data storage efficiency becomes paramount. This paper explores a novel approach to address these challenges by integrating the AES-128 encryption algorithm for data security and the Huffman compression algorithm for improved storage efficiency in NVM systems.

Although this study includes both the Huffman compression algorithm implementation and the fully efficient NVM usage system, it focuses deeply with the encryption of data to be stored in memory using AES-128. Both the study of using the Huffman compression algorithm to reduce the size of the data to be written to NVM [1] and the study of increasing the efficiency in NVM with the index table structure [2] were discussed in separate publications by the authors of this paper. Therefore, methodologically, it will be a paper in which the security part is more prominent. The reason for choosing AES as the data encryption method is that this method is efficient and strength, as well as the fact that AES has been preferred many times in projects where NVM was used before. A fast and energy-efficient AES implementation is mentioned in a paper published in 2015 [3]. In addition, in the study conducted by Xie et al., a technology called MIP based on the AES method is discussed [4]. The AES encryption system also has options such as 128, 192, and 256. What changes in these options is the length of the key used when encrypting the data. Since it is a focus of work that emphasizes data size as well as security, it is more appropriate to encrypt with the shortest length key.

Ensuring the confidentiality and integrity of stored data is a critical aspect of NVM systems. Prior to data storage, our proposed methodology employs the AES with a 128-bit key length. AES-128, known for its strength and efficiency, provides a secure cryptographic foundation, safeguarding sensitive information from unauthorized access or tampering [5]. This encryption process is a fundamental step towards establishing a robust security framework for data stored in NVM.

Beyond security concerns, efficient utilization of memory space is essential for maximizing the performance of NVM systems. In this study, we introduce the integration

of the Huffman compression algorithm to significantly reduce the size of stored data. Huffman coding, a widely adopted lossless compression technique, exploits the frequency distribution of data to represent frequently occurring symbols with shorter codes and less frequent symbols with longer codes [6]. This approach not only conserves memory space but also contributes to faster data retrieval and transmission.

The proposed methodology is not confined to independent application of encryption and compression; rather, it represents a synergistic approach. By combining AES-128 encryption with Huffman compression, we aim to establish a comprehensive solution that not only enhances data security but also optimizes memory space utilization. The symbiotic relationship between these two algorithms contributes to a balanced trade-off between security measures and storage efficiency in NVM systems. The problem of data size growth caused by encryption has been eliminated with the index table efficiency increase structure and Huffman compression algorithm, and even provides an advantage for large data.

## 2. Materials

### 2.1. Microcontroller Unit (MCU)

Microcontroller Unit (MCU) is a term used to describe a microcontroller unit designed as an integrated circuit that plays a critical role in electronic systems [7]. Commonly used in embedded systems, MCUs are capable of performing complex functions, processing sensitive data, and executing specialized tasks [8]. MCUs support automation by providing control for many electronic devices.

MCUs are essentially composed of the following components:

#### 2.1.1. Central Processing Unit (CPU)

The central processing unit (CPU) of an MCU serves as the core component responsible for fundamental computational and control tasks [9]. It typically employs Reduced Instruction Set Computing (RISC) or Complex Instruction Set Computing (CISC) architectures [10]. These processors operate at high speeds, facilitating the execution of complex mathematical operations, logical functions, and decision-making processes. The CPU executes program code, processes data, and applies specific algorithms. It may feature parallel processing capabilities, interrupt control, and low power consumption, enhancing the MCU's performance and energy efficiency [11].

#### 2.1.2. Memory

MCUs possess various types of memory units to store program code and data. Program memory, which can be Flash memory or Electrically Erasable Programmable Read-Only Memory (EEPROM), stores the MCU's operating code and persistent data. Flash memory is especially adept at ensuring the permanent storage of software, allowing programs to remain in memory without the need for an external power source [12]. EEPROM offers the capability to electrically erase and reprogram data, making it useful for storing permanent configuration information.

Additionally, MCUs typically utilize Static Random Access Memory (SRAM) for storing temporary data and computation results. SRAM provides rapid access, enabling swift storage and retrieval of temporary data during processing. This facilitates the MCU's ability to process data quickly during operations [13].

Memory management in MCUs is vital for preserving data integrity and ensuring fast access during processing. The memory system may also incorporate specialized memory areas such as NVM. It is used for storing critical configuration data and information, even in the event of power interruptions, ensuring data retention. This enhances the MCU's reliability and durability, making it invaluable for critical applications.

### 2.1.3. Input/Output Ports

MCUs are equipped with various input and output ports to facilitate communication with the external world. Digital input/output pins have the capability to detect or generate low or high-level voltages. Analog input pins are capable of reading analog data from environmental sensors [14]. These ports are used for purposes such as reading data from sensors, controlling motor speed, sending data to displays, and interacting with other external devices. Furthermore, the MCU's ability to manage interrupts enables it to promptly respond to external events.

### 2.1.4. Processing Units

MCUs can have specialized processing units. These include counters, timers, Pulse Width Modulation (PWM) controllers, Analog to Digital (A/D) converters, and communication interfaces. These processing units facilitate the MCU's ability to perform specific tasks. For instance, PWM controllers enable precise control of motor speed or LED brightness by converting analog signals into digital format [15].

### 2.1.5. Clock

MCUs operate at a specific clock frequency, and the clock system ensures the synchronized operation of internal components. Internal oscillators or external clock

sources are responsible for accurately timing the MCU's processor, memory, and other components. This synchronization is vital for maintaining the MCU's stability and reliability, ensuring that it operates consistently and dependably [16].

In this manner, the fundamental components of the MCU come together to provide reliable and optimized control across a wide range of applications. The balanced design of these components ensures the efficient operation, energy savings, and longevity of the MCU. Therefore, MCUs are acknowledged as the cornerstone of modern electronic systems.

The study involved the use of the NXP S32K148 MCU platform. The NXP S32K148 is a 32-bit microcontroller based on the ARM Cortex-M4 core [17]. It is purpose-built for automotive applications, meeting stringent automotive-grade quality standards and delivering dependable performance in challenging automotive electronics environments while adhering to AEC-Q100 standards [18]. Moreover, it complies with ISO 26262 functional safety requirements, making it a suitable choice for safety-critical systems. The NXP S32K148 MCU distinguishes itself in automotive applications due to its robust features, particularly its NVM capabilities. The NVM of the NXP S32K148 MCU securely and permanently stores calibration data, configuration settings, user data, and other critical system information [19]. This not only ensures data security but also provides quick access and contributes to improved energy efficiency.

### 2.2. Non-Volatile Memory (NVM)

Non-volatile memory (NVM) or persistent memory is a type of memory that can retain stored data even after the system has lost all power. In contrast, volatile memory requires a constant power source to maintain data integrity [20].

NVM typically pertains to storage within semiconductor memory chips. These chips store data in floating-gate memory cells that consist of floating-gate Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs), including forms of flash memory storage such as NAND flash and solid-state drives (SSD).

Other examples of NVM include read-only memory (ROM), erasable programmable read-only memory (EPROM), and EEPROM, ferroelectric RAM, most types of computer data storage devices (e.g., disk storage, hard disk drives, optical discs, floppy disks, and magnetic tape), as well as early computer storage methods like punched tapes and cards [21].

The operation of NVM encompasses a broad family of various technologies. Different NVM types such as NAND Flash, NOR Flash, EEPROM, FRAM, and MRAM exist. Each

type offers distinct advantages in terms of storage capacity, speed, and durability. Therefore, selecting the one that aligns with application requirements is crucial [22].

The fundamental write and erase operations of NVM are carried out through specialized procedures applied to cells. For instance, in NAND Flash memory, cells are erased and written in block units, while in NOR Flash and EEPROM, each cell can be individually written and erased [23]. NVM programmatically stores data by applying electrical signals to the cells. Data is written to cells through changes in electrical current or voltage. This electrical process alters the physical structure of cells, thereby preserving the data.

NVM maintains data at the cell level. Each cell is represented by different states (such as 1 and 0), and data is stored through changes in these states. This ensures the permanent retention of data. Durability is a key attribute of NVM. NAND Flash memory, in particular, possesses the capability to withstand frequent write and erase operations, preventing cells from becoming unusable.

The speed and access times of NVM vary depending on the technology employed. For example, NAND Flash memory is well-suited for rapidly storing and retrieving large data blocks, although it may exhibit slower random access times. Figure 1 illustrates the diagram of the NXP S32K148 EVK NVM.

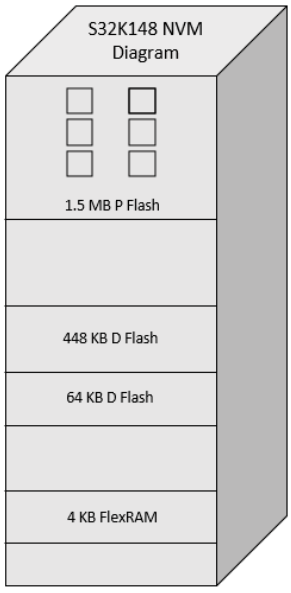


*Figure 1: NXP S32K148 EVK NVM diagram*

### 2.3. Huffman Compression Algorithm

The Huffman compression algorithm was developed by David A. Huffman in 1952 during his doctoral thesis work at MIT. Through this thesis, David A. Huffman laid the foundation for data compression algorithms, particularly for compressing symbols. The fundamental concept of the algorithm involves assigning different lengths of codes to symbols based on their frequencies, thus achieving compression. The Huffman compression algorithm is a lossless compression algorithm used for compressing data. This algorithm matches more frequent symbols with shorter bit sequences and less frequent symbols with longer bit sequences, taking into account the frequencies of different symbols in the data to be compressed [24].

In data compression, the Huffman compression algorithm proceeds through several key steps. In the first step, data analysis is performed on the content to be compressed. During this phase, the data's content is scanned, and the frequency of occurrence for each symbol, such as characters, is determined. The next step involves creating a priority queue based on the frequencies of symbols. The two symbols with the lowest frequencies are taken from the queue and merged. The total frequency of the merged symbols is considered, and this new symbol is added to the tree [25]. This process continues until only one symbol remains, resulting in the construction of a Huffman tree.

After constructing the Huffman tree, each symbol is assigned a bit sequence that represents its path in the tree. These codes are designed to be shorter for more frequent symbols and longer for less frequent symbols [26]. This coding scheme is what enables data compression. Each symbol in the data is then encoded using the Huffman tree, resulting in the representation of the data using shorter bit sequences instead of the original symbols, thereby achieving compression. Huffman-encoded data consumes less storage space and facilitates faster data transmission. Compressed data can be decompressed on the receiver's end using the Huffman tree, restoring it to its original form.

In summary, the Huffman compression algorithm involves data analysis to determine symbol frequencies, the construction of a Huffman tree to create symbol codes, and the subsequent compression of data using these codes, which can then be efficiently stored or transmitted.

In the prior research, the Huffman compression algorithm was employed to enhance data storage efficiency in NVM by reducing the data size [1].

### 2.4. Integrated Development Environment (IDE)

An Integrated Development Environment (IDE) is a potent software tool utilized to facilitate and expedite software development processes [27]. It empowers software developers to enhance their workflow in coding, debugging, compiling, and project management, thus fostering more efficient practices in the development lifecycle.

One of the core features of IDEs is their advanced code editing capabilities. These tools are tailored to meet the needs of developers working with text-based programming languages. They offer an array of features, including automatic code completion, syntax highlighting, code folding, and automated code formatting. These features collectively aid developers in crafting clean and efficient code while significantly speeding up the coding process.

IDEs are instrumental in simplifying the debugging process. Developers can execute their code step-by-step, inspect variable values, and pinpoint errors with precision. This systematic approach to debugging accelerates the identification and resolution of software flaws, resulting in more robust and reliable applications [28].

IDEs are equipped to compile and execute software programs seamlessly. This feature allows developers to rapidly test their applications within the IDE, promoting an efficient and convenient testing phase.

For complex software projects, IDEs provide essential project management tools. These tools assist developers in organizing their projects, managing resources, and collaborating effectively. The ability to handle large and intricate projects cohesively is a significant advantage of IDEs [29].

S32 Design Studio stands out as a specialized (IDE) designed for the development of applications in the automotive and industrial sectors. This IDE offers a comprehensive suite of tools and features aimed at simplifying the design, creation, and debugging processes for embedded systems based on NXP's S32 microcontroller platform. It accommodates widely-used programming languages like C and C++, and incorporates an extensive range of libraries and middleware components to expedite development tasks [30].

The user-friendly interface of S32 Design Studio provides an intuitive environment for developers to seamlessly navigate and manage their projects. Its robust code editor includes essential features like syntax highlighting, code completion, and refactoring capabilities, facilitating efficient and error-reduced code writing. Furthermore, the IDE seamlessly integrates with well-established debugging tools, enabling tasks such as code

step-through, breakpoint configuration, and variable inspection, thus simplifying the detection and resolution of issues. Additionally, it offers hardware debugging support through NXP's on-chip debuggers, providing real-time access to the target system for effective debugging and optimization [30].

With its rich set of features, S32 Design Studio empowers developers to create high-quality applications tailored for the automotive and industrial sectors. In this research, we utilized S32 Design Studio as the chosen IDE to implement the Huffman compression algorithm on sample data. Additionally, its compatibility played a crucial role in flashing the software onto the NXP platform for our testing purposes and monitoring the data.

### 2.5. Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is a symmetric encryption algorithm approved as a Federal Information Processing Standard (FIPS) by the National Institute of Standards and Technology (NIST) of the United States in 2001. AES is a block cipher encryption algorithm designed for the secure encryption and decryption of data, primarily used to ensure data privacy and security [31]. AES is included in the ISO/IEC 18033-3 standard.

AES can operate with various key lengths: 128-bit, 192-bit, and 256-bit. The key length chosen determines the strength of the encryption, with longer keys offering higher security but requiring increased computational power. AES is classified as a "symmetric" encryption algorithm because the same key is used for both encryption and decryption processes [32].

AES consists of the following fundamental stages:

#### 2.5.1. Key Expansion

This key schedule is responsible for producing subkeys that will be used in each round of the encryption process by generating them in the key expansion stage. These subkeys are created by extending the original key length and associating different portions of the key with each round.

#### 2.5.2. Initial Round

An XOR operation is performed between the plaintext block and the key in the initial round, marking the beginning of the encryption process.

#### 2.5.3. Rounds

Using the key schedule, the plaintext block undergoes encryption in a series of rounds. Each round consists of complex mathematical operations between the plaintext block and the subkeys. These operations progressively mix and relate the plaintext block with the key, enhancing the security of the encryption.

### 2.5.4. Final Round

In the final round, the transformations applied in previous rounds are executed without further mixing. This concludes the encryption process, yielding the ciphertext.

In this study, the AES-128 algorithm has been employed. AES-128 provides a robust level of security. With a 128-bit key length, it offers resilience against numerous cryptographic analysis methods, safeguarding data from malicious attacks. From the standpoint of speed and efficiency, AES-128 proves highly effective. It swiftly encrypts and decrypts data while imposing minimal computational overhead on the processor. Another notable advantage of AES-128 is its seamless integration capabilities. It is supported across various programming languages and cryptographic libraries. Figure 2 illustrates the encryption principle of the AES-128 encryption algorithm in a simple way.

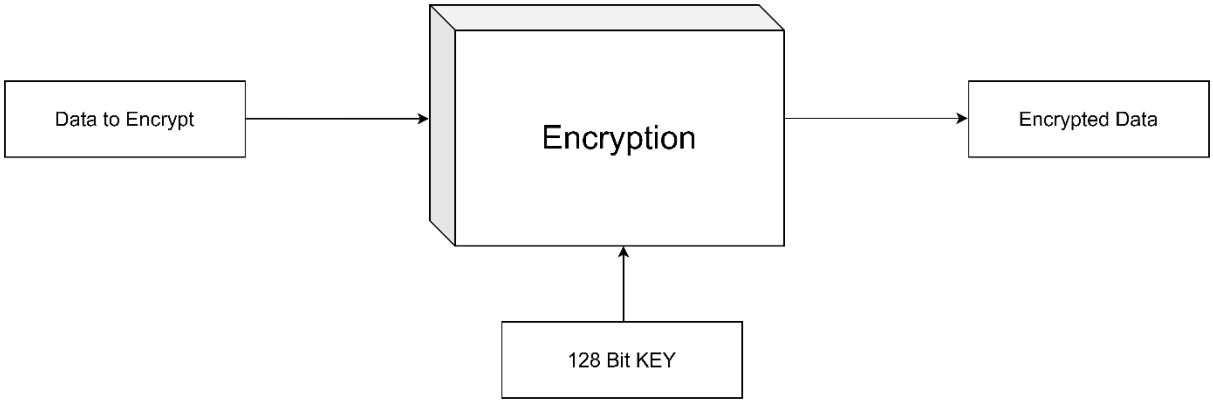


*Figure 2: AES-128 encryption block diagram*

Figure 3 illustrates the principle of decryption of the AES-128 encryption algorithm in a simple way.

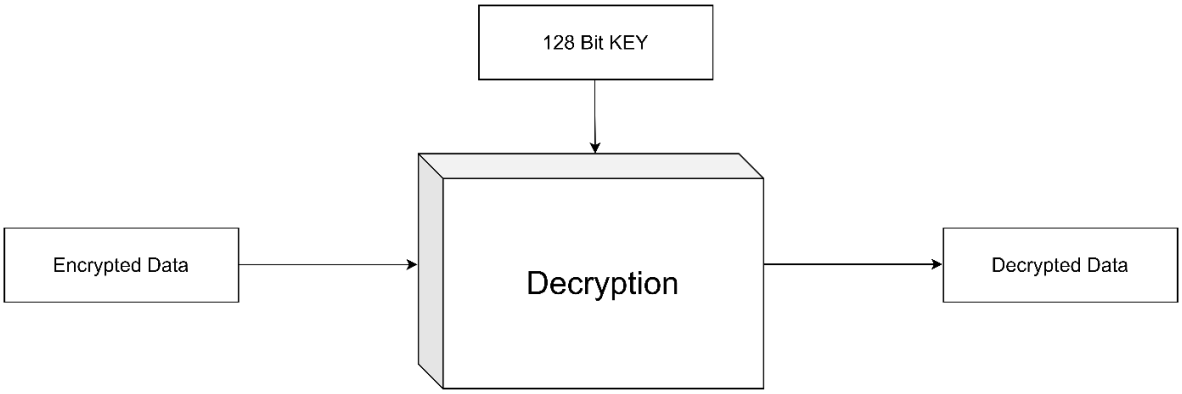


*Figure 3: AES-128 decryption block diagram*

### 2.6. OpenSSL

OpenSSL is an open-source cryptographic library and toolkit. It serves as a multifaceted collection of tools used to ensure secure communication and perform various security-related operations, including encryption, digital certificates, security protocols, and key management. OpenSSL is accessible across a wide range of platforms and supports many widely used security protocols [33].

Below is a more detailed overview:

#### 2.6.1. Open-Source Software

OpenSSL is an open-source project, which means that its source code is open to inspection, modification, and distribution by anyone. This allows security experts and developers to scrutinize the code and make improvements.

#### 2.6.2. Cryptography

OpenSSL is fundamentally designed for encryption and cryptographic operations. Through its utilization of cryptographic algorithms, it enables the secure transmission of data. This empowers users to encrypt their data for confidentiality, strengthen authentication processes, and create digital signatures.

#### 2.6.3. Secure Sockets Layer/Transport Layer Security Protocols

OpenSSL is used to implement secure communication protocols such as SSL and TLS [34]. These protocols ensure secure data transmission over the internet. Particularly, TLS, used extensively between web browsers and servers, encrypts and secures data on the internet.

#### 2.6.4. Digital Certificates

OpenSSL is employed to create and manage digital certificates. Digital certificates are essential for authenticating and ensuring the trustworthiness of websites. OpenSSL facilitates the production, editing, and communication with certificate authorities.

#### 2.6.5. Key Management

OpenSSL is often used for creating and managing secure key pairs. These keys are utilized in encryption and digital signature operations. OpenSSL also supports the secure storage and sharing of keys.

#### 2.6.6. Diverse Platform

OpenSSL can function on numerous operating systems and platforms, offering developers the flexibility to use their applications in various environments.

In this study, OpenSSL's AES library has been employed. In the realm of data security and encryption, OpenSSL's AES-128 library emerges as a robust and reliable solution. OpenSSL is renowned for its strong emphasis on security. It has been rigorously tested and widely used in various security-critical applications for many years. AES-128, a subset of the OpenSSL library, follows the NIST standards and is considered highly secure against known attacks [35]. OpenSSL's AES-128 implementation is known for its efficiency. It takes advantage of hardware acceleration and optimized code, ensuring that encryption and decryption processes do not introduce significant computational overhead. This speed is particularly important in scenarios where data must be processed rapidly without compromising security. OpenSSL's AES-128 library is cross-platform, making it a versatile choice for a wide range of applications. Whether you are developing software for various operating systems or embedding encryption in a network device, OpenSSL offers compatibility across different platforms. OpenSSL's AES-128 library provides developers with the flexibility to tailor encryption to their specific needs. It supports various modes of operation (e.g., ECB, CBC, GCM) and key sizes, allowing you to adapt the encryption process to your application's requirements.

## 3. Method

This study consists of three basic stages. Compressing and writing the data to memory based on the index table was carried out based on previous studies. The main part is to encrypt the data before writing it to memory in order to ensure its security. Therefore, the method in this paper concentrates on the encryption process. Although these two

processes have been carried out before, the encryption process is not included as the last step. Since encryption with AES-128 leads to an increase in the size of the data, it was included in the process as the first step in order not to reduce efficiency. In this way, the data whose size is enlarged is then compressed, aiming to save on memory size and increase efficiency.

### 3.1. Security Process

#### 3.1.1. Encryption Process

The initial phase of this study involved employing the Advanced Encryption Standard type 128-bit encryption method. This was utilized to ensure the security and integrity of the input data.

AES encryption uses a series of complex mathematical operations to transform readable plaintext data into a ciphered and encrypted format of ciphertext. This encrypted text is essentially unrecognizable, and efforts to decipher it without the requisite decryption key are considered obscenely difficult to achieve.

AES-128 represents a specific variant of the AES encryption, mainly defined by the size of the key utilized in the encryption process. In this case, a key of 128 bits was employed. A 128-bit key generated from a cryptographically secure random source offers a practical balance between computational efficiency and robust data security.

The AES encryption process is highly intricate and involves several stages. The first step, subbytes step, is the stage in which each byte in the state matrix is replaced with a byte from a statically defined 8-bit look-up table, the Rijndael S-box. Then, rows shifting step follows it. Here, bytes in each row of the state are cyclically shifted. The first row is left untouched while each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. The next process is the mixing columns. This operation combines the four bytes in each column of the state using a linear transformation. Using arithmetic in a finite field, each byte of a column is replaced by a value dependent on all four bytes in the column. At the end, the addition of the rounded key step is performed. This process involves the state being XOR'd with the round key. This operation is elegantly simple, bitwise addition of the state with a round key, and is easy to implement whether in hardware or software.

These stages constitute one round of operations, and with AES-128, ten rounds of such operations are conducted. As a point of note, the column mixing operation is excluded from the final round.

In encryption, each round makes use of a round key. These round keys are derived from the original input key by means of the Key Expansion routine. The Key Expansion routine employs galois field multiplication and a core routine that involves a byte-wise shift and a byte substitution operation to derive round keys.

These sequences of intricate operations constitute the AES-128 encryption process. It ensures that the input data is encrypted effectively through scrambling data in a systematic, yet highly entropic manner. As a result, data becomes impervious to unauthorized threats and breaches, and only someone in the possession of the correct decryption key can revert the encrypted text back to its original format.

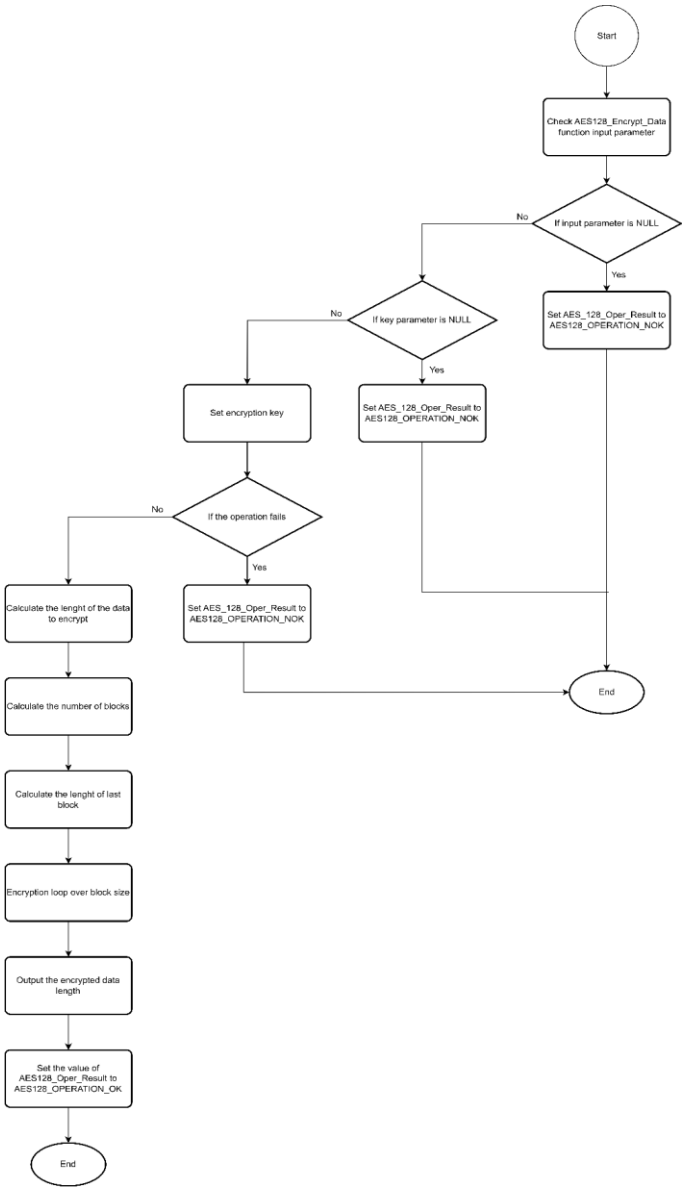


*Figure 4: AES-128 encryption process flowchart*

Figure 4 represents a flow diagram of the encryption process carried out in this study. Regardless of the type of data received as input, its size or the purpose for which it will be used, it is given as an input argument to the AES128_Encrypt_Data function. After verifying the existence of this input, the program checks the key value to be used during encryption. If there is a valid key, it starts the encryption process. If there is no error during the process, the steps of the AES-128 algorithm are run and the encrypted data is separated from the encryption block as output for compression. If a valid value is not found or an error is received in any of these steps, the system returns AES128_OPERATION_NOK.

### 3.1.2. Decryption Process

Decryption is the reverse process of encryption and utilizes the same sequence of steps but in reverse order to convert the unreadable ciphertext back into its original plaintext.

For the Advanced Encryption Standard (AES) 128-bit decryption algorithm, it revisits each round of encryption but in a retrograde pattern. If the same secret key used during the encryption is supplied, the AES-128 decryption algorithm will reproduce the original data. The decryption process involves the following main steps. The AES decryption process begins with the adding rounded key step, which involves the bitwise XOR operation of the ciphered data with the expanded round key. This step is the same as in the encryption process. However, the key schedule is performed in reverse. The second step is the inverse of shifted rows performed during the encryption. Each row in the state is shifted cyclically towards the right, with each row varying in the number of places it is shifted. The first row is left in place, the second row moves one space to the right, and so on. This step reverses the permutation done during encryption. Inversing sub bytes is the next step for decryption. This step involves the transformation of the bytes using the inverse of the S-box used during encryption. An inverse substitution for each byte of the block is performed to derive the original data, and the last step is inversing mixed columns. 6This step undoes the effect of mixed columns during encryption. The result is a matrix whose column entries are derived from the inverse linear transformation of corresponding column entries in the input state.

These steps together represent a stage of decryption, and this process is repeated for as many rounds of encryption that were initially applied. For the case of AES-128, there are ten rounds of decryption that need to be applied. However, it is important to note that the inversing mixed columns operation is removed from the final round.

At the end of these steps, the output is the original plaintext data that was encrypted. Thus, AES-128 decryption successfully retrieves the original data without compromising the data's integrity.

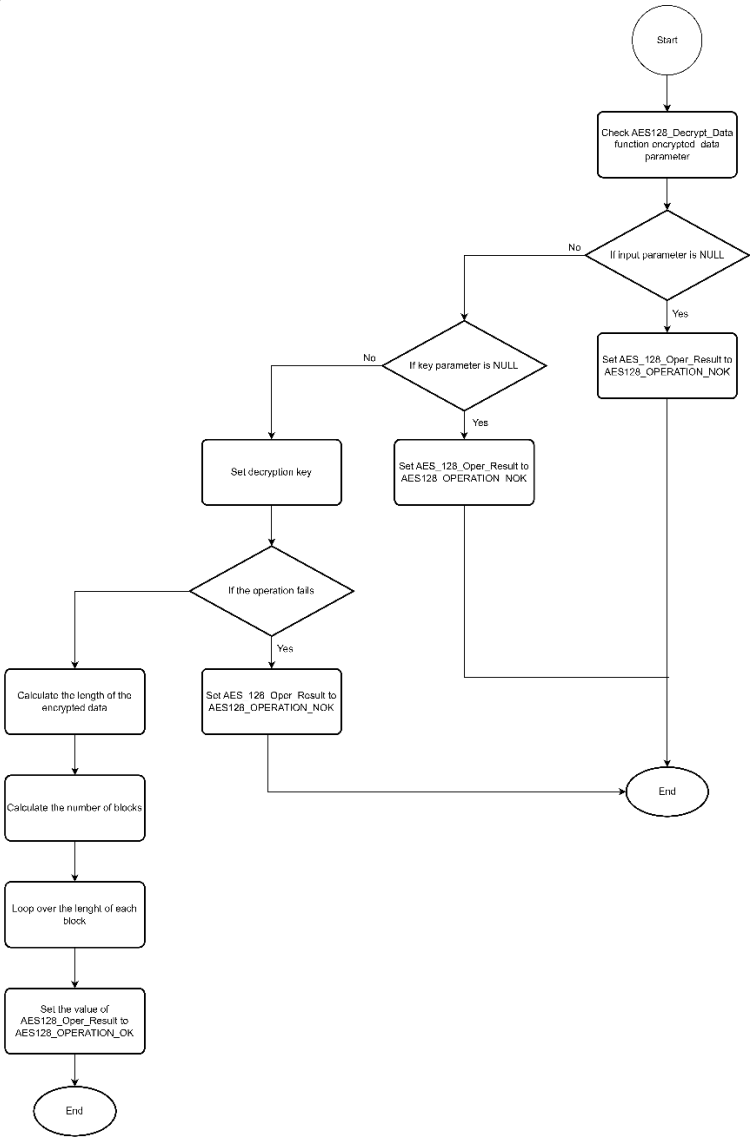


*Figure 5: AES-128 decryption process flowchart*

### 3.2. Comperssion Process

This stage of the study is based on the work done by Karacali, Cebel and Donum (2023). The methodology in that publication was followed exactly. The data encrypted in the previous block is given as input to the compression block to reduce the size. The data compressed with the Huffman compression algorithm passes to the next process to be written to NVM [1]. Figure 6 represents the flow during the compression structure on full efficient NVM usage.

### 3.3. Writing Data to NVM Process

This part of the work is also based on the study done by Karacali, Cebel and Donum (2023). Data address tracking, data addition, removal and updating related to the index file were all carried out by following the methodology in that study. The data, which is first encrypted with the AES-128 algorithm and then reduced with the Huffman compression algorithm, comes to the block in this process and is written to NVM. Figure 6 presents the steps that the data taken as input goes through in this process.

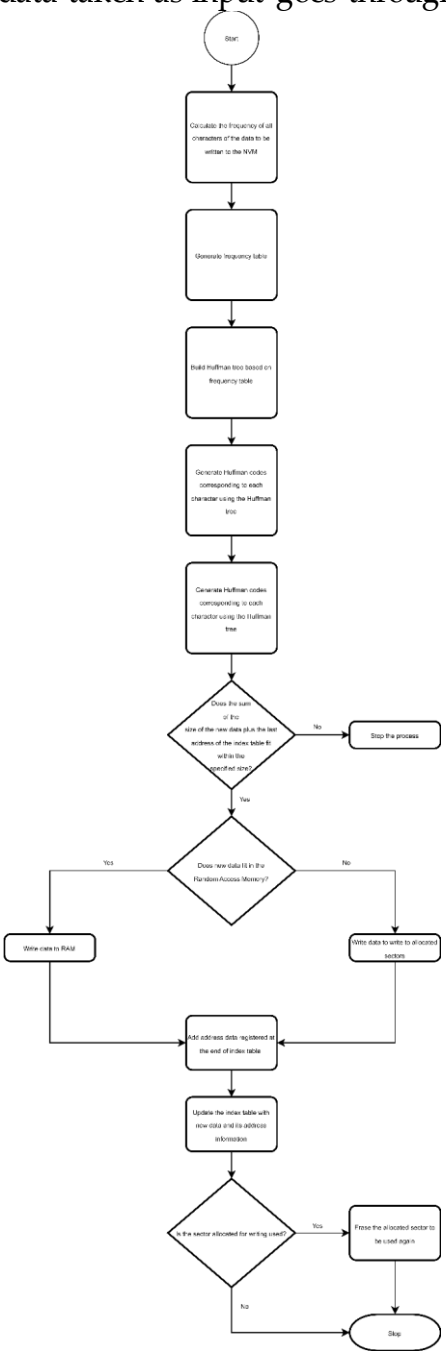


*Figure 6: Huffman compression process flowchart [1]*

## 4. Result

Figure 7 shows the encryption, compression and writing to NVM parts of the developed secure and efficient NVM structure, respectively.

Additionally, Figure 8 represents the decompressing and parsing of secure and compressed data after reading it from NVM.

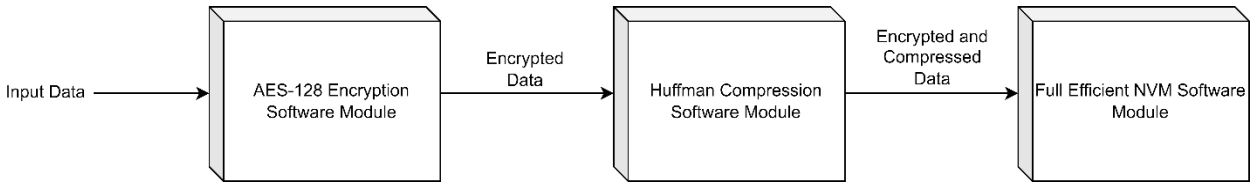


*Figure 7: Encryption, compression, and data writing block diagram*

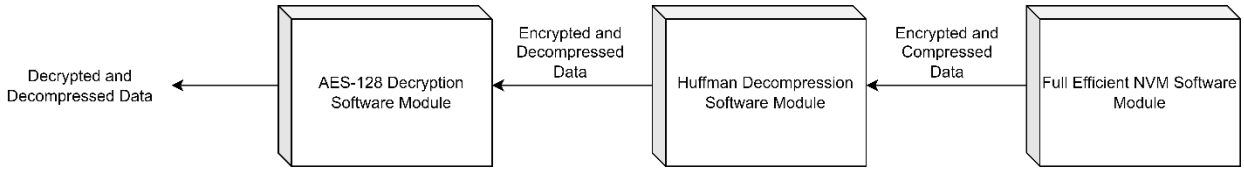


*Figure 8: Data reading, decompression, and decryption block diagram*

In this study, as mentioned in the method section, studies were carried out on 2 sample data. Firstly, it is the GPS point of the 'Clock Tower', the symbol of the city of Izmir, and secondly, it is the numerical mileage value that should be kept as secret data in the memory of a car.

For both sample data, first the raw version (without any encryption or compression process) was presented, then only the encrypted version, and finally the compressed version after being encrypted was written to memory.

In the represented figures below, the left side of the memory browser is set to the hexadecimal values, and the right side is text.

### GPS Data Result

The GPS location in terms of latitude and longitude of the selected place is 38.41887263488455, 27.128669775180803.

Firstly, the raw version of this data is written into the memory as given below.

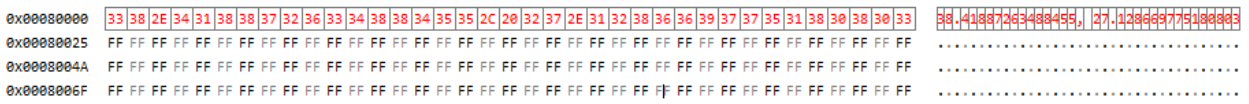

*Figure 9: Raw GPS location data*

GPS location data written in its raw form takes up 37 bytes as given in Figure 9.

Then, the AES-128 encrypted version of the data is written to memory as given below.

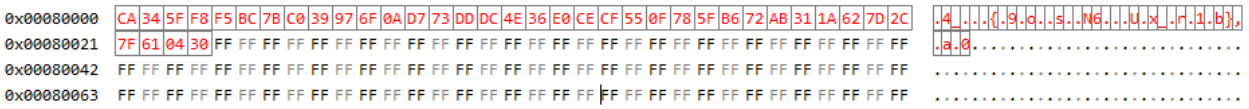

*Figure 10: Encrypted GPS location data*

The encrypted version of the same data becomes 41 bytes after this operation as shown in Figure 10. It can be clearly seen the encryption is inserted 4 more bytes to the raw data and the size that is allocated for this data is increased. Lastly, Huffman algorithm is applied to the encrypted data and the size is expected to be smaller as given below.

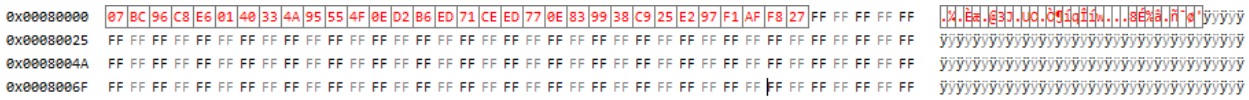

*Figure 11: Compressed GPS location data*

Data that is compressed with the Huffman algorithm takes up 32 bytes in memory as shown in Figure 11. This means that it is 5 bytes smaller than its raw form and 9 bytes smaller than its encrypted form. Security is ensured by encryption, and efficiency is increased by increasing the available space in memory. This GPS location example shows that the study is successfully progressing towards its initial purpose.

**Kilometer Data Result**

The kilometer in terms of kilometer per hour of a vehicle is 1059872.

Firstly, the raw version of this data is written into the memory as given below.

*Figure 12: Raw kilometer data*

Kilometer data written in its raw form takes up 4 bytes since it is an integer value as given in Figure 12.

Then, the AES-128 encrypted version of the data is written to memory as given below.

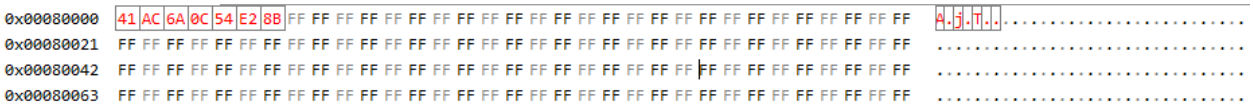


*Figure 13: Encrypted kilometer data*

The encrypted version of the same data becomes 7 bytes after this operation as shown in Figure 13. It can be clearly seen the encryption is inserted 3 more bytes to the raw data and the size that is allocated for this data is increased. Lastly, Huffman algorithm is applied to the encrypted data and the size is expected to be smaller as given below.


*Figure 14: Compressed kilometer data*

Data that is compressed with the Huffman algorithm takes up 6 bytes in memory as shown in Figure 14. In this case, while 1 byte is gained compared to the encrypted version, there is a total increase of 2 bytes compared to the raw version. At this point, the trade-off between security and efficiency must be considered and determined according to the requirements and priorities of the system to be used. In addition, from this example, it can be inferred that the Huffman compression algorithm works at a higher rate on data with larger raw sizes.

## 5. Discussion and Conclusion

In conclusion, this paper has presented a comprehensive exploration of the integration of the AES-128 encryption algorithm and the Huffman compression algorithm to enhance security and efficiency in non-volatile memory (NVM) systems. The synergistic approach employed in this study aims to strike a balance between safeguarding data integrity and optimizing memory space utilization.

The implementation of AES-128 encryption ensures a robust security foundation, protecting sensitive information from unauthorized access or tampering. By encrypting data prior to storage in NVM, we contribute to the establishment of a secure framework crucial for modern computing environments where data privacy is of utmost importance.

Furthermore, the introduction of Huffman compression as a complementary measure addresses the imperative need for efficient memory space utilization. The compression algorithm, based on the frequency distribution of data, allows for a significant reduction

in the size of stored information. This not only conserves memory space but also facilitates quicker data retrieval and transmission, contributing to the overall performance enhancement of NVM systems.

The discussion surrounding the implementation of AES-128 encryption and Huffman compression in NVM systems prompts considerations for future research and practical applications. One notable aspect is the trade-off between security and computational overhead. While AES-128 provides a high level of security, the computational resources required for encryption may impact the overall system performance. Investigating optimized implementations of AES-128 for NVM systems could offer insights into mitigating this potential trade-off.

Moreover, the efficiency of Huffman compression is influenced by the nature of the data being processed. Future research could explore adaptive compression techniques that dynamically adjust to varying data characteristics, potentially further improving compression ratios and system performance.

The scalability of the proposed approach across different types of NVM technologies is another avenue for exploration. As emerging NVM technologies, such as resistive random-access memory (RRAM) and phase-change memory (PCM), gain prominence, adapting the proposed security and efficiency measures to these technologies will be crucial for their successful integration into diverse computing architectures.

In summary, the integration of AES-128 encryption and Huffman compression in NVM systems presents a promising avenue for addressing the dual challenges of data security and storage efficiency. This research opens the door for further investigations and practical implementations that can contribute to the continual evolution of secure and efficient non-volatile memory solutions in the ever-expanding landscape of information technology.

## 6. Acknowledge

We would like to convey our gratitude to Huseyin Karacali, the Software Architect, for his skillful mentorship and motivating leadership. Additionally, we acknowledge the invaluable assistance provided by TTTech Auto Turkey throughout the progression of the project.

## References

[1] H. Karacali, E. Cebel and N. Donum, "Optimizing NVM Utilization Efficiency with Huffman Compression Algorithm in MCU Based System," 2023 8th International Conference on Computer Science and Engineering (UBMK), Burdur, Turkiye, 2023, pp. 75-80, doi: 10.1109/UBMK59864.2023.10286571.

[2] Karacali, H., Dönüm, N., & Cebel, E. (2023). Full Efficient NVM Usage For MCU. The European Journal of Research and Development, 3(1), 115–128. https://doi.org/10.56038/ejrnd.v3i1.245

[3] J. Clement, B. Mussard, D. Naccache and L. Torres, "Implementation of AES Using NVM Memories Based on Comparison Function," 2015 IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 2015, pp. 356-361, doi: 10.1109/ISVLSI.2015.37.

[4] M. Xie, S. Li, A. O. Glova, J. Hu and Y. Xie, "Securing Emerging Nonvolatile Main Memory With Fast and Energy-Efficient AES In-Memory Implementation," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 11, pp. 2443-2455, Nov. 2018, doi: 10.1109/TVLSI.2018.2865133.

[5] Mehta, J. (2023, July 14). What is AES 128 bit encryption and how secure AES 128 bit is?. CheapSSLWeb.com Blog. https://cheapsslweb.com/blog/what-is-aes-128-bit-encryption

[6] Wikimedia Foundation. (2023, September 18). Huffman coding. Wikipedia. https://en.wikipedia.org/wiki/Huffman_coding

[7] Embedded Artistry LLC. (2021, June 9). Microcontroller Unit - Embedded Artistry. Embedded Artistry. https://embeddedartistry.com/fieldmanual-terms/microcontroller-unit/

[8] Lutkevich, B. (2019, November 7). microcontroller (MCU). IoT Agenda. https://www.techtarget.com/iotagenda/definition/microcontroller#:~:text=A%20microcontroller%20is%20a%20compact,peripherals%20on%20a%20single%20chip.

[9] Ag, I. T. (n.d.). Microcontroller - Infineon Technologies. Copyright Infineon Technologies AG - All Rights Reserved. https://www.infineon.com/cms/en/product/microcontroller/

[10] Microcontrollers - Overview. (n.d.). https://www.tutorialspoint.com/microprocessor/microcontrollers_overview.htm

[11] Utmel. (2020, May 28). What is a Microcontroller? Utmel All Rights Reserved. https://www.utmel.com/blog/categories/microcontrollers/what-is-a-microcontroller

[12] GeeksforGeeks. (2023, May 5). Microcontroller and its Types. https://www.geeksforgeeks.org/microcontroller-and-its-types/

[13] Inc, L. T. (2022, October 25). What is a Microcontroller? — LOOMIA Soft Electronics | E-textiles. LOOMIA Soft Electronics | E-textiles. https://www.loomia.com/blog/what-is-a-microcontroller

[14] Thornton, S. (2022, February 12). microcontrollers vs microprocessors what's the difference. Microcontroller Tips. https://www.microcontrollertips.com/microcontrollers-vs-microprocessors-whats-difference/

[15] Administrator. (2023, May 26). Basics of Microcontrollers – History, structure and Applications. ElectronicsHub. https://www.electronicshub.org/microcontrollers-basics-structure-applications/

[16] What is a microcontroller? The defining characteristics and architecture of a common component - technical articles. (2019, April 4). https://www.allaboutcircuits.com/technical-articles/what-is-a-microcontroller-introduction-component-characteristics-component/

[17] General Purpose Microcontrollers | NXP semiconductors. (n.d.). https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus:GENERAL-PURPOSE-MCUS

[18] S32K148-Q176 Automotive General-Purpose Evaluation Board. (n.d.-b). https://www.nxp.com/design/development-boards/automotive-development-platforms/s32k-mcu-platforms/s32k148-q176-evaluation-board-for-automotive-general-purpose:S32K148EVB

[19] S32K1 General-Purpose MCUS. (n.d.). NXP Semiconductors. https://www.nxp.com/products/processors-and-microcontrollers/s32-automotive-platform/s32k-auto-general-purpose-mcus/s32k1-microcontrollers-for-automotive-general-purpose:S32K1

[20] Wikipedia contributors. (2023, September 7). Non-volatile memory. Wikipedia. https://en.wikipedia.org/wiki/Non-volatile_memory

[21] Patterson, D. A., & Hennessy, J. L. (2004). Computer Organization and design: The Hardware/Software Interface, Third Edition. Elsevier.

[22] GeeksforGeeks. (2023a, February 21). Difference between Volatile Memory and Non Volatile Memory. https://www.geeksforgeeks.org/difference-between-volatile-memory-and-non-volatile-memory/

[23] What is Non-Volatile Memory? (2021, December 2). Fierce Electronics. https://www.fierceelectronics.com/electronics/what-non-volatile-memory

[24] Huffman, D. A. (1952). A method for the construction of Minimum-Redundancy Codes. Proceedings of the IRE, 40(9), 1098–1101. https://doi.org/10.1109/jrproc.1952.273898

[25] Data structures. (n.d.). https://cgi.luddy.indiana.edu/~yye/c343-2019/huffman.php

[26] Huffman Coding. (n.d.). https://engineering.purdue.edu/ece264/17au/hw/HW13?alt=huffman

[27] Why is Integrated Development Environment (IDE) Important? (n.d.). Spiceworks. https://www.spiceworks.com/tech/devops/articles/what-is-ide/

[28] Oshana, R. (2006). Managing the DSP software development effort. In Elsevier eBooks (pp. 351–387). https://doi.org/10.1016/b978-075067759-2/50012-0

[29] Bird, C., Menzies, T., & Zimmermann, T. (2015). The art and science of analyzing software data. In Elsevier eBooks. https://doi.org/10.1016/c2012-0-07289-4

[30] S32 Design Studio IDE. (n.d.). NXP Semiconductors. https://www.nxp.com/design/software/development-software/s32-design-studio-ide:S32-DESIGN-STUDIO-IDE

[31] 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

[32] Dworkin, M. J. (2023). Advanced Encryption Standard. https://doi.org/10.6028/nist.fips.197-upd1

[33] OpenSSL Foundation, Inc. (n.d.). /index.html. https://www.openssl.org/

[34] OpenSSL Foundation, Inc. (n.d.-a). /docs/man3.1/man7/crypto.html. https://www.openssl.org/docs/man3.1/man7/crypto.html

[35] Kekayan. (2018, July 7). Encrypt files using AES with OPENSSL - Kekayan - Medium. Medium. https://kekayan.medium.com/encrypt-files-using-aes-with-openssl-dabb86d5b748