# Message Authentication Codes On Ultra-Low SWaP Devices

Che-Hsien Liao

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Matthew Hicks, Chair
Chang Tien Lu
Taejoong (Tijay) Chung

May 11, 2022
Falls Church, Virginia

Keywords: Cryptography, HMAC, AES-CMAC, Ultra-Low SWaP Devices

# Message Authentication Codes On Ultra-Low SWaP Devices

Che-Hsien Liao

(ABSTRACT)

This thesis focuses on specific crypto algorithms, Message Authentication Codes (MACs), running on ultra-low SWaP devices. The type of MACs we used is hash-based message authentication codes (HMAC) and cipher-block-chaining message authentication code (CBC-MAC). The most important thing about ultra-low SWaP devices is their energy usage. This thesis measures different implementations' execution times on ultra-low SWaP devices. We could understand which implementation is suitable for a specific device. In order to understand the crypto algorithm we used, this thesis briefly introduces the concept of hash-based message authentication codes (HMAC) and cipher-block-chaining message authentication code (CBC-MAC) from a high level, including their usage and advantage. The research method is empirical research. This thesis determines the execution times of different implementations. These two algorithms (HMAC and CBC-MAC) contain three implementations. The result comes from those implementations running on the devices we used.

# Message Authentication Codes On Ultra-Low SWaP Devices

Che-Hsien Liao

(GENERAL AUDIENCE ABSTRACT)

The deployments of 5G cellular networks are now onboard. The demand increased due to consumers and the availability of more affordable devices. The amount of investment in 5G technology and infrastructure increases market interest in IoT. The 5G network security is essential. How to secure user privacy and their sensitive data while they use 5g network has become a big issue and needs to be solved. However, not all popular crypto algorithms are suited to all devices, especially in those resource-limited microcontrollers. In this thesis, we will deal with Message Authentication Codes that provide the data integrity check. With resource limit devices, energy usage is an important issue. We will identify which implementations have better energy usage depending on the device features. This thesis will use three implementations for each algorithm. The result of our experiment provide a straightforward way that helps people understand which implementation can run more efficiently on specific ultra-low devices.

# Dedication

*This thesis is dedicated to my parent and the people who have supported me at Virginia Tech.*

# Acknowledgments

I would like to express my sincerest gratitude to Dr. Matthew Hicks for his guidance, supporting, and funding me as GRA. I really learned a lot from the weekly meetings.

I would also like to thank Dr. Chang Tien Lu. Dr. Lu cares about our daily life and our progress on the dissertation through my Master's degree. Thank Dr. Lu, for sending a lot of helpful information for me.

I would like to thank my family and my close classmate for supporting me.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

Before the 90s, networks were relatively uncommon, and there were not a lot of internet users. People communicated using dial phone or physical mail in the past. However, In this digital age, more and more things rely on the Internet, wireless networks, and computer systems. As a result, network security is becoming significant because we sometimes transmit our sensitive data through those devices or computers. Although the Internet makes people have a more convenient way to communicate with each other, the amount of malware birthed simultaneously to steal users' data. In the report generated by the Check Point Research, we find that the number of network attacks per week on public networks increased 50 percent in 2021 compared to 2020.

**According to the Key Malware Statistics in 2022 [6]:**

- 560,000 new pieces of malicious software are detected every day.
- There are now more than 1 billion malicious software programs out there.
- Every minute, four companies fall sufferer from malicious software.
- Trojans account for 58% of all computer malicious software.

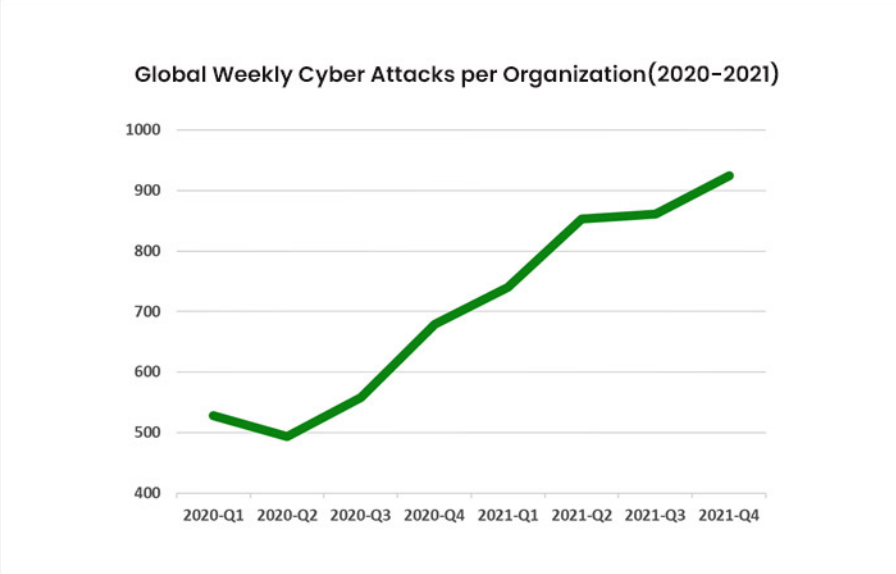


Figure 1.1: Global Weekly Attacks [12]

One essential aspect for secure communications is that of cryptography. But it is important to note that while cryptography is necessary for secure communications, it is not by itself sufficient [8]. Cryptography helps us achieve data integrity and message authentication.
As the number of IoT devices increases significantly, the data transmission through IoT devices also needs to be secure. Due to those IoT devices having different configurations and resources (CPU and Memory), choosing proper crypto algorithms is essential. This thesis aims to find an efficient implementation of specific crypto algorithms, "Message Authentication Codes" (MACs). We briefly introduce two types of MACs, Advanced Encryption Standard(AES) based MACs and Hash function based MACs. Then measure the execution time for those implementations. The target devices we choose are ultra-low SWaP devices. The implementations have been executed on those devices. The experiments and results section can find out the execution times and then determine The performance measurement section can find out the execution times and then determine which implementations are suitable for certain devices and have better resource usage.

## 1.2 Motivation

Tons of IoT devices contain different resources and different configurations. These differences will impact the execution time. This thesis focuses on ultra-low SWaP devices, which means their resources are relatively limited compared to regular microcontrollers or computers. Due to this condition, the usage of resources becomes essential. The ultra-low SWaP devices should have the ability to operate crypto algorithms while keeping lower resources usage. Message Authentication Codes (MACs) have simple operations, and it also plays a crucial role in network security. This simple bitwise operation makes it a good fit for ultra-low SWaP devices. On the other hand, RSA and Elliptic curves have a multiplication operation, which has a much slower execution time. The result of measurement is important for all ultra-low SWaP devices that help them find out the best resource usage implementation.

## 1.3 Thesis Structure

- **Chapter 1:** Introduces the thesis topic and discuss the crypto algorithm and devices being used to measure the result.
- **Chapter 2:** Introduces the advantage, usage, and algorithm of Message Authentication Codes (MACs).
- **Chapter 3:** The methodology we use to measure the performance and then provide the ultra-low SWaP in this thesis.
- **Chapter 4:** Describe the result of the measurement based on those devices listed on chapter 3.
- **Chapter 5:** Summarizes the results of our research, followed by our future work.

## 1.4 Research method

This thesis uses two types of research method

- **Literature review**

  In order to implement accurate crypto algorithms, doing a literature review is to supply reliable and precise information for Message authentication codes. The literature review also contains Advanced Encryption Standard (AES) and Hash function (SHA) since this thesis focus on hash-based MACs and block-cipher-based MACs.

- **Empirical research**

  After having solid information about crypto algorithms, we pick three algorithms for hash-based MACs (HMAC) and block-cipher-based MACs (CMAC) from online open-source. The standard for choosing algorithms is that they should follow the correct operation and be relatively portable. We have two types of MACs, HMAC and CMAC. Each MACs is divided into two parts. The first part is MACs operation and the second part is its based crypto algorithms; the based crypto algorithm of HMAC is Hash function, and the based crypto algorithm of CMAC is AES. We choose one MACs implementation and three based implementations, respectively. Our target boards have executed those implementations and measured execution time.

# Chapter 2

# Cryptographic algorithms

This chapter introduces the standard of Message authentication codes (MACs). This thesis focus on HMAC and CMAC. We define MACs as the outer layer, and the inner layer is crypto algorithms of the outer layer based on; in this case, is AES and Hash function. The inner layer will be introduced first, and then the outer layer will be introduced in the rest of this chapter. Moreover, AES is family of block-cipher and symmetric cryptography, so we will discuss block-cipher and symmetric cryptography at the beginning.

## 2.1 Block Ciphers

Block cipher is the best-known cryptographic primitive and fundamental component of symmetric-key primitives. The plaintext is split into a fixed-length size. Block ciphers use the same key shared between communication parties to encrypt a whole block of data bits at its round. In practice, the prevalence of block ciphers either has a fixed size of 16 bytes (128 bits) such as advanced encryption standard (AES), or a fixed size of 8 bytes (64 bits) like the data encryption standard (DES) algorithm[2].

**Definition 2.1.1.** Block Cipher Encryption and Decryption

*The block cipher contains two types of algorithms, one for decryption D and one for encryption E. A block cipher with block size n-bits and key size k-bits. A block cipher with plaintext*

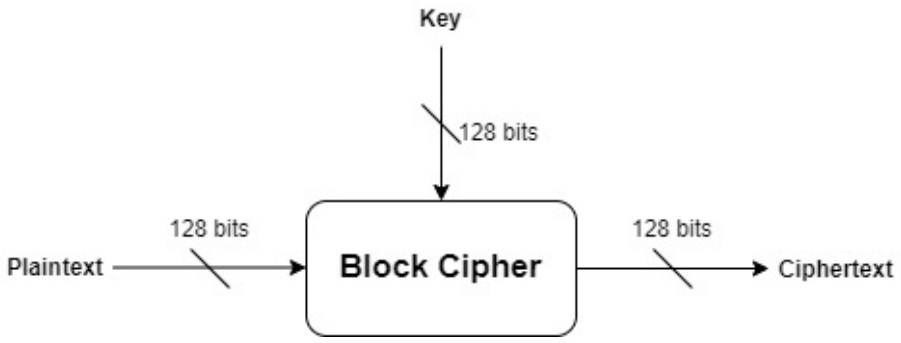


Figure 2.1: Principles of encrypting 128 bits with a block cipher

*P is specified by an encryption function* $E_K(P) := E(K, P) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ *and with ciphertext C decryption function* $D_K(C) := D(K, C) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$. *Moreover, it satisfies that for given P and fixed-length K,* $\forall K : D_K(E_K(P)) = P$.

## 2.2 Symmetric Cryptography

Before we discuss AES, this chapter introduces symmetric cryptography since AES is one of the most popular symmetric algorithms. Symmetric cryptographic algorithms are also known as symmetric key, single key, and secret key schemes. It satisfies diffusion and confusion. These two primitive operations are provided by the famous information theorist Claude Shannon in his 1945 classified report "A Mathematical Theory of Cryptography."[17] Diffusion and confusion are the way how good encryption should operate. The diffusion is that when we change a bit of the plaintext, the ciphertext will be very different, which means the several characters of ciphertext would be changed. It has the same meaning that if we change a bit of ciphertext, the plaintext will change a lot. Diffusion property can hind statistical properties of plaintext. Confusion is another property that obscures the relationship between key and ciphertext. There is no way to find correspondences between key and ciphertext. The common element from AES and DES to achieve confusion is substitution

operation.


Figure 2.2: Diffusion property: switch 1 bit of x should not only change 1 bit of result y.

In the Block Cipher chapter, Block cipher takes the equal length of key and plaintext and outputs the same ciphertext length. The size is usually 64 bits or 128 bits. Block ciphers can operate at a time. However, in the real world, we might want to encrypt our data more than just 64 bits or 128 bits block of plaintext, for example, when encrypting large PDF files or email. Several modes of operation make block cipher can encrypt large size of data, e.g Electronic Codebook (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB) and Cipher Feedback (CFB).

The electronic Code Book mode (ECB) is the simplest method of encrypting data. The plaintext would be split into multiple blocks with equal length. Let's assume the block cipher encrypted n-bits size. The plaintext that exceeds n-bits would partition into n-bits blocks. If the plaintext size is not a multiple of n-bits, it has to pad to multiple n-bits before doing an encryption operation.

**Definition 2.2.1.** Electronic Codebook Mode (ECB)

*Define $e()$ as arbitary block cipher with block size $n$, set $p_i$ and $c_i$ as bit strings of size $n$.*

***Encryption operation:*** $c_i = e_k(p_i), i \geq 1.$

***Decryption operation:*** $p_i = e_k^{-1}(c_i) = e_k^{-1}(e_k(p_i)), i \geq 1.$

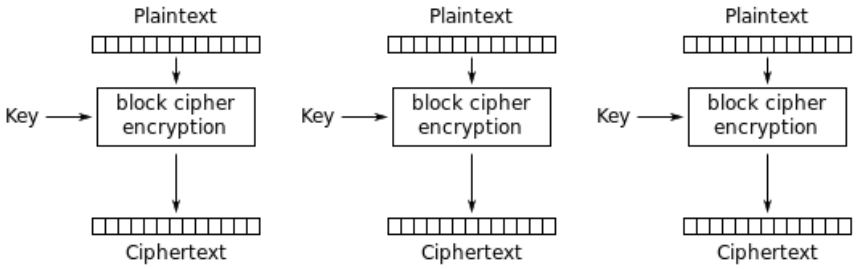


Figure 2.3: Electronic Codebook Mode encryption [wiki]

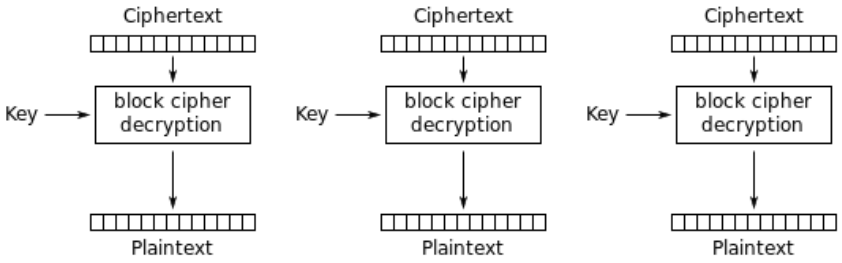


Figure 2.4: Electronic Codebook Mode decryption [16]

However, ECB has its weaknesses because the identical plaintext blocks will result in the same ciphertext block. So, using ECB is not recommended.

Cipher Block Chaining (CBC) is one of the most prevalent operation modes. It solves the problem of ECB that we mentioned previously. During CBC mode operation, the ciphertext block no longer depends on a certain plaintext block but all previous plaintext blocks. Moreover, the initialization vector (IV) is used to randomize.

**Definition 2.2.2.** Cipher Block Chaining (CBC)

*Define $e()$ as arbitary block cipher with block size $n$, set $p_i$, $c_i$ and $IV$ as bit strings of size $n$.*

***Encryption operation (First block):*** $c_1 = e_k(p_1 \oplus IV)$

***Encryption operation (Rest block):*** $c_i = e_k(p_i \oplus c_{i-1}), i \geq 2$

***Decryption operation (First block):*** $p_1 = e_k^{-1}(c_i) \oplus IV$

***Decryption operation (Rest block):*** $p_i = e_k^{-1}(c_i) \oplus c_{i-1}, i \geq 2$

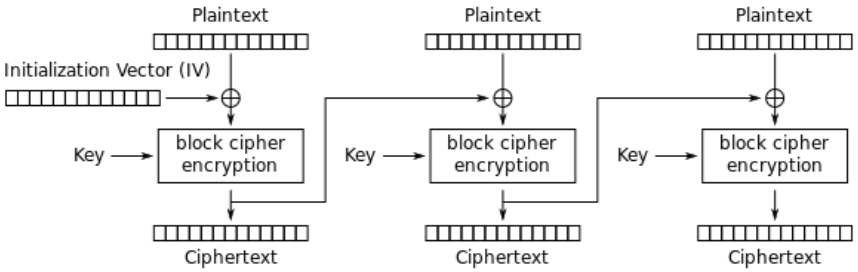


Figure 2.5: Cipher Block Chaining Mode Encryption [16]

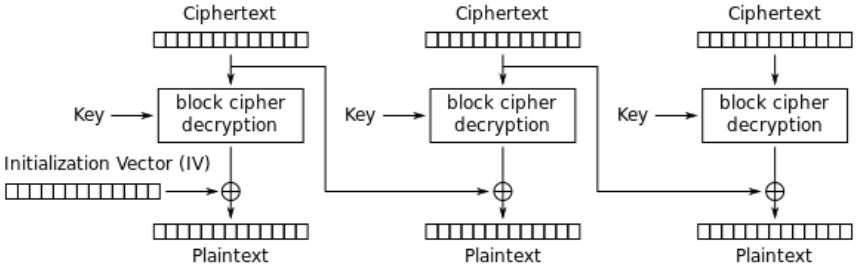


Figure 2.6: Cipher Block Chaining Mode decryption [16]

Output Feedback (OFB) transforms the block cipher into stream cipher. The plaintext itself is not used as an input; instead, the block cipher is used to generate the stream bits and then XORed with the plaintext to get the ciphertext.

**Definition 2.2.3.** Output Feedback (OFB)

*Define $e()$ as arbitary block cipher with block size $n$, set $p_i$, $c_i$, $block_i$ and $IV$ as bit strings of size $n$.*

***Encryption operation:*** $block_1 = e_k(IV)$ *and* $c_1 = block_1 \oplus p_1$

***Encryption operation:*** $block_i = e_k(block_{i-1})$ *and* $c_i = block_i \oplus p_i, i \geq 2$

***Decryption operation:*** $block_1 = e_k(IV)$ *and* $p_1 = block_1 \oplus c_1$

***Decryption operation:*** $block_i = e_k(block_{i-1})$ *and* $p_i = block_i \oplus y_i, i \geq 2$

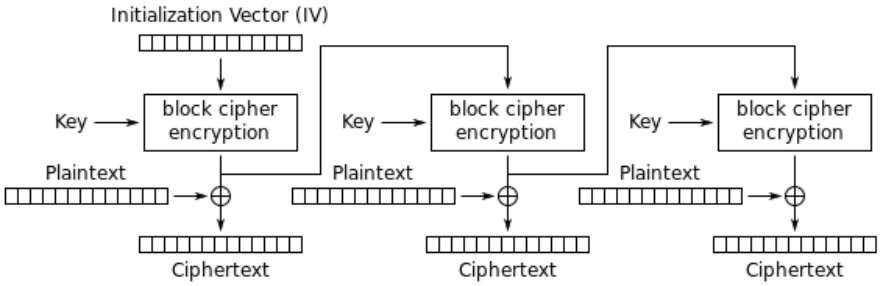


Figure 2.7: Output Feedback Mode Encryption [16]

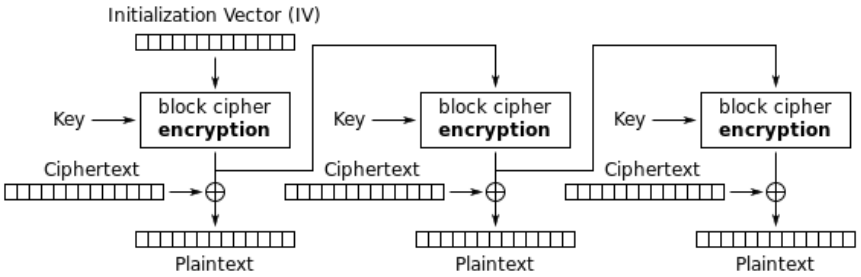


Figure 2.8: Output Feedback Mode Decryption [16]

Cipher Feedback (CFB) is the stream cipher mode like OFB, but the only difference between CFB and OFB is that in CFB, the plaintext influences encryption. Rather than feeding back the result of the block cipher, CFB uses ciphertext to feed back.

**Definition 2.2.4.** Cipher Feedback (CFB)

*Define $e()$ as arbitary block cipher with block size $n$, set $p_i$, $c_i$, and IV as bit strings of length $n$.*

***Encryption operation:*** $c_1 = e_k(IV) \oplus p_1$

***Encryption operation:*** $c_i = e_k(c_{i-1}) \oplus p_i, i \geq 2$

***Decryption operation:*** $p_1 = e_k(IV) \oplus c_i$

***Decryption operation:*** $p_i = e_k(c_{i-1}) \oplus c_i, i \geq 2$

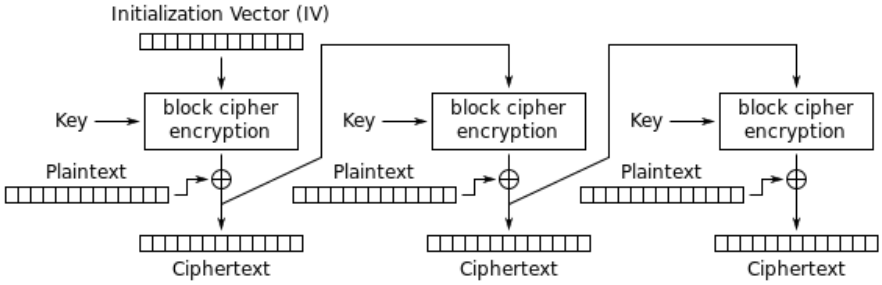


Figure 2.9: Cipher Feedback Mode Encryption [16]

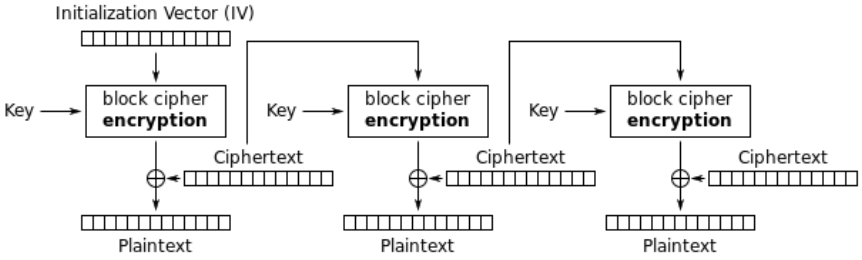


Figure 2.10: Cipher Feedback Mode Decryption [16]

## 2.3 Advanced Encryption Standard (AES)

In 1997 US National Institute of Standards and Technology (NIST) called for proposals for the Advanced Encryption Standard (AES). The main goal is to replace the Data Encryption Standard (DES) since DES is not efficient for software implementation. DES has the disadvantage of the block length of 64 bits which is a weakness in particular applications. After NIST evaluated the proposal, they narrowed down the number of potential candidates. The proposal called Rijndael won the competition. The NIST mentioned that block cipher Rijndael as AES final standard and was published in 2001. (FIPS PUB 197)

On a high overview of AES, AES operates block cipher with 128 bits with a specific key length, which is 128, 192, and 256 bits. These key length also known as AES-128, AES-192, and AES-256. We have mentioned that AES contains three key lengths. Those key lengths

must be supported by Rijndael due to the NIST design requirement. The total internal rounds can be different by key length. With 128 bits key lengths, AES has 10 rounds of the cipher. AES with 192 bits contains 12 rounds, and 256 bits contains 14 rounds. Some parts of AES can be done in parallel, making it possible to implement efficient performance. Low memory requirements make AES appropriate for restricted environments.

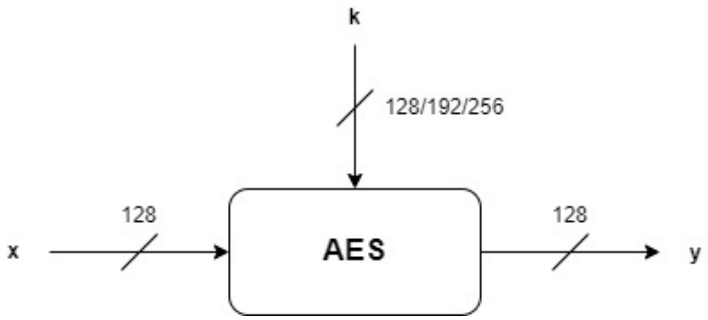


Figure 2.11: AES overview

AES contains three different types of operation. Each operation uses whole 128 bits of plaintext. Every result of data is also referred to as state. Every round except the first contains all three operations.

- **Key Addition operation:** Every round has a 128-bit round key or subkey; the round key is created from the primary key in the key expansion. The round key is used to XORed to the state.
- **Byte Substitution operation (S-Box):** Each state uses a lookup table with mathematical properties. This step provides confusion to the data.
- **Diffusion Layer:** There are two linear operations in this operation. Both provide diffusion over all state. The ShiftRows sublayer permutes the state of data, and the MixColumn sublayer is a matrix operation that combines every block of 4 bytes.

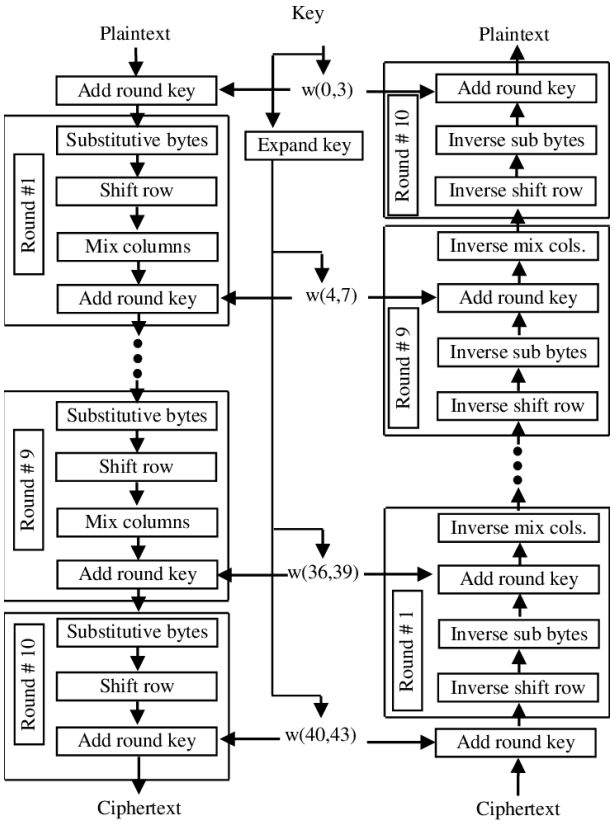


Figure 2.12: AES implementation [15]

The state is a two-dimensional byte array. The array contains four rows and four columns. Each element of the array represents 1 byte. The 16-byte input is fed byte-wise to the S-Box. The 16-byte output is permuted in the ShiftRows layer and then combined by the MixColumn. The last step is that the 128-bit subkey is XORed with the result of the state. All these steps form a round.The ShiftRow sublayer shifts all the rows except the first row. The second row of state shift one position to the left, the third row of state shift two positions to the left, and the last row of state shift three positions to the left. The MixColumn sublayer also provides diffusion. This step mixes each column of the state using multiplication. The last step of the round is AddRoundKey which operates XORed for the result of MixColumn with a round key created from an encryption key.

| | | y | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| x | 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| | 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| | 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| | 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| | 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| | 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| | 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| | 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| | 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| | 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| | a | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| | b | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| | c | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| | d | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| | e | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| | f | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

Figure 2.13: S-box: substitution values mapping in hexadecimal value of a byte.[3]

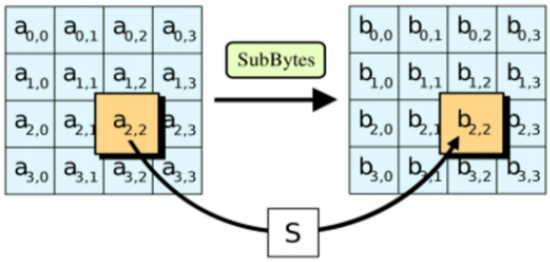


Figure 2.14: Byte Substitution operation [11]

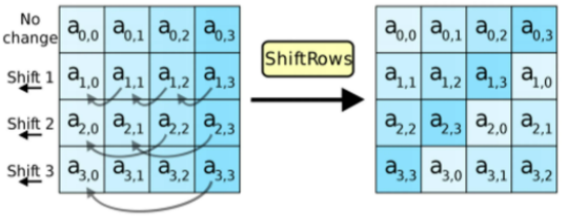


Figure 2.15: ShiftRow operation [11]

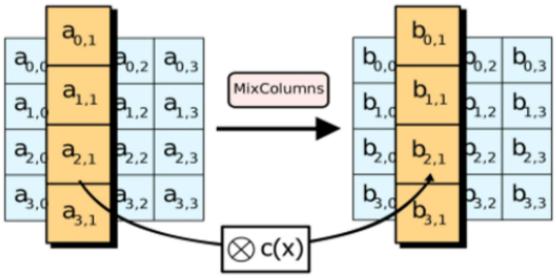


Figure 2.16: MixColumn operation[11]

## 2.4 Hash Function

Hash functions are widely used and are essential cryptographic primitive. They play an important role in the digital signature. The asymmetric algorithms RSA also can do signature. However, the data size is usually between 1024 and 3072 bits because it can not exceed the modulus value; a lot of emails are longer than this size. So how can we sign a large message? The easy solution is to break the message into fit length and sign the block individually. But it will be three problems [2].

- **High Computational Load**
- **Message OverHead**
- **Security Limitation**

The Hash function can solve these problems. Hash functions are one-way algorithms. Hash functions take an arbitrary-length message and return a fixed-length bit-string. It is also called a digest. Unlike RSA based signature, hash can save computing resources due to using a fixed length. There are some security requirements of Hash function. First, it must be one-way, which means if we have a hash output y, it must be hard to find input data x such that

| Algorithm | Output [bit] | Input [bit] | Rounds | Collisions found |
|---|---|---|---|---|
| **MD5** | 128 | 512 | 64 | yes |
| **SHA-1** | 160 | 512 | 80 | not yet |
| **SHA-256** | 256 | 512 | 64 | no |
| **SHA-384** | 384 | 1024 | 80 | no |
| **SHA-512** | 512 | 1024 | 80 | no |

Table 2.1: The MD4 hash functions.

$y = h(x)$. It is impossible to find the original message through the result. Second, collision-free and collision-resistant are essential. It is impossible to find two different messages to get the same hash value. SHA family is based on MD4. MD4 was developed by Ronald Rivest, and it is the most popular hash function, also referred to a message digest algorithm. MD4 is a software-friendly algorithm that all operations are bitwise like OR, AND, and XOR. In 1996, many experts recommended SHA-1 as a replacement for the MD5, the strengthened version of MD4. SHA-1 produces a 160-bit digest. The maximum collision resistance of SHA-1 is $2^{80}$, which is also the weakest point. SHA-1 is the most famous and widely used in many protocols and applications. NIST raised three variant of SHA-1 in 2001, which is SHA-256, SHA-384, and SHA-512. All of These variants are also called SHA-2. SHA-1 and SHA-2 have two main steps, preprocessing and hash computation. The step of preprocessing contains three stages, padding, dividing the message, and initial hash value. Before hash operation, the data has to be padded to fit the length of a multiple of 512 bits. In order to get an overall data length of a multiple of 512 bits, It append a ”1” followed by the numbers of zero bits. The rest of the part is added binary 64-bit representation in the block. We use one example to outline the padding stage. Assume the message ”abc” were to be encoded using SHA-1. The message ”abc” equals the length of 24 bits in size.

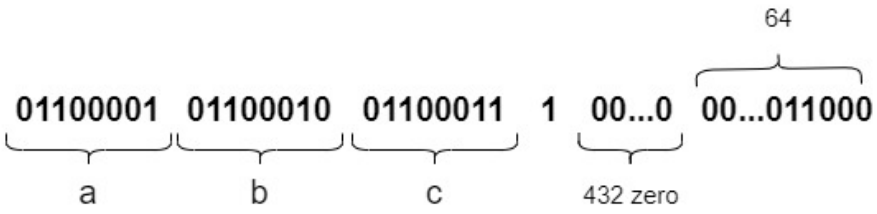


Figure 2.17: Padding SHA-1 message.

In Figure 2.17, we can see that number of bits left in the block is 512–64 = 448 bit. So the number of bits to pad with zeroes is 448–(24 + 1) = 423 bit. After doing this padding, it divides the message into 16 words of the size of 32 bits. SHA-1 initials five random strings of hex characters that will serve in the hash function. The second step is hash computation. SHA-1 and SHA-2 have the same principles. The hash algorithm takes the first block and the initial value, and the result of the hash of the first block is the input of the next round with the next block. The last round result is the actual hash value.

## 2.5 Message Authentication Codes

A Message Authentication Code (MACs) is a cryptographic checksum and is also known as a keyed hash function. MACs have some same properties as digital signatures, both of them provide message integrity and message authentication. MACs do not provide nonrepudiation because MACs are symmetric-key schemes. Even so, MACs are important protocol for applications and they have been widely utilized for over many years because they are based on the hash function and are much faster than digital signatures. MACs use a symmetric-key $k$ which is known for both receiver and sender. In MACs, the sender will compute the MACs value, $m = MAC(x, k)$, and add an authentication tag at the end of the message. The sender will send not only the message $x$ but also the check tag $m$ to the receiver.

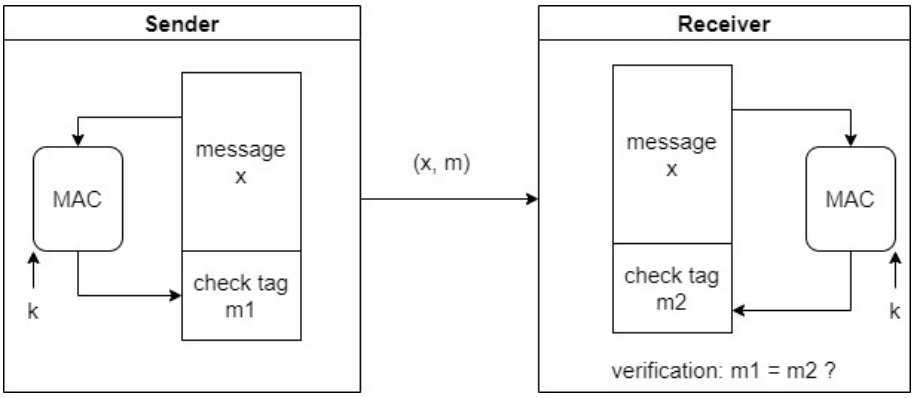


Figure 2.18: MAC principles.

### 2.5.1 HMAC

The HMAC structure was proposed by Mihir Bellare, Ran Canetti, and Hugo Krawczyk in 1996 [1]. HMAC contains an inner and outer hash. We can see the HMAC structure in Figure 2.19.

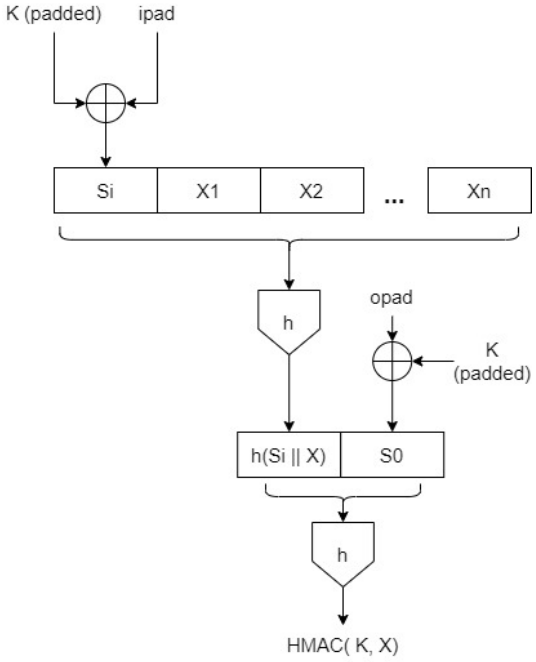


Figure 2.19: HMAC construction.

To compute the HMAC, It can be defined as below.

$$HMAC(k, x) = h[(k \oplus opad)||h[(k \oplus ipad)||x]] \tag{2.1}$$

The h represents the hash function. The k represents the secret key, and The x represents the message. If the length of the secret key is shorter than the block size, it will pad with zero. When the length of the secret key is longer than the block size, it will be truncated. The symbol  stand for concatenation. The inner pad (ipad) and outer pad (opad) are the repetitions of the bit pattern[7].

$$ipad = 00110110, 00110110, ..., 00110110 \tag{2.2}$$

$$opad = 01011100, 01011100, ..., 01011100 \tag{2.3}$$

### 2.5.2 MACs from Block Ciphers: CBC-MAC

An alternative way to construct MACs is to use AES cipher block chaining(CBC), which we mentioned in chapter 2.3. Figure 2.20 describes the construction of CBC-MAC. $e$ denotes as a block cipher, that is AES in this thesis.

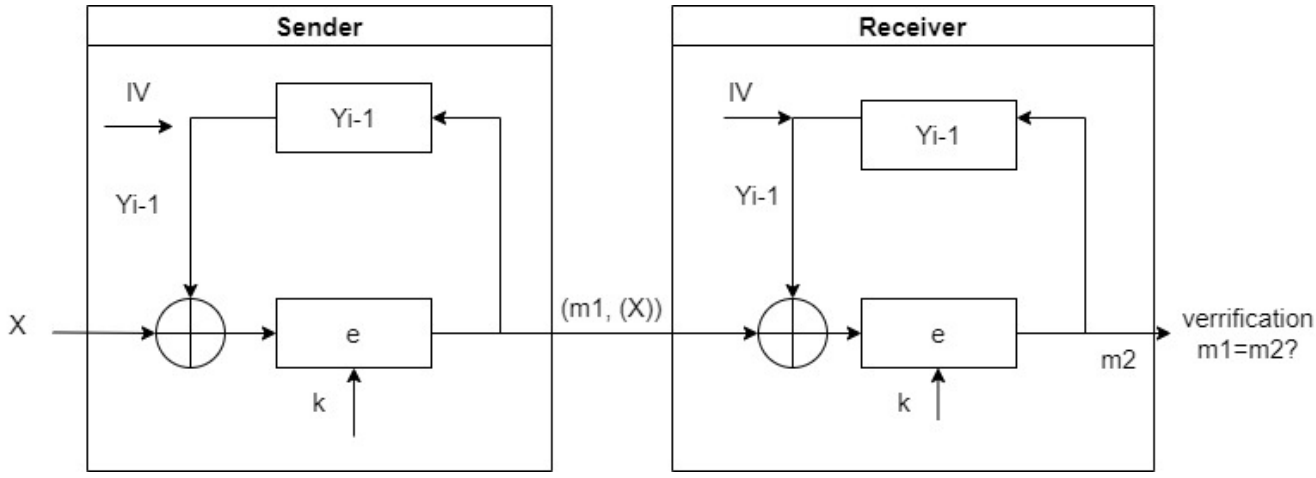


Figure 2.20: CBC-MAC construction.

# Chapter 3

# Problem Statement & Methodology

## 3.1 Problem Statement

The number of embedded devices is increasing, and the attack techniques are evolving simultaneously. More and more sensitive information from embedded devices is transmitted via the public internet. Therefore, embedded devices must provide different levels of security features. Implementing cryptographic features on embedded devices might have several challenges. The most important part is the performance. An Embedded system is known for limited resources and low battery power. In this thesis, our lowest resource device is the MSP430G2553, which only has a 16 mHz MCU with 16KB flash memory and 512B SRAM. There are several factors that affect the design of cryptographic algorithms, which is computational power, memory size, and power consumption. Using asymmetric algorithms like Elliptic curve (ECC) or RSA might not be suitable for these resource-limit devices due to the requirement of computational power. MACs scheme is ideal for resource-limit devices. We are going to find the implementation of MACs that have better performance for those devices with their resource constraints.

## 3.2 Methodology

This chapter will go over the methodology of time measurement. We are going to discuss the code implementation, test environment, and method of time measurement.

### 3.2.1 Code Implementation

Our HMAC and CMAC implementation code are selected from open sources. The MAC implementation is the same, and we have three different implementations for AES-128 and SHA256, respectively. All the implementation should be relatively lightweight and suitable for our target devices. We customize our main file, which can easily switch the implementation through the config file. Table 3.1 details our chosen AES and SHA implementations.

| **MACs Type** | **Crypto algorithm** | **Source** | **License** |
|---|---|---|---|
| HMAC | SHA | Allan Saddi[13] | Copyright |
| | | Brian Gladman[5] | Copyright |
| | | Mbedtls[10] | Apache License |
| CMAC | AES | Tiny-AES[14] | Copyright |
| | | Brian Gladman[4] | Copyright |
| | | Mbedtls[9] | Apache License |

Table 3.1: MACs implementations.

### 3.2.2 Test Environment

We have four target devices, and their main feature are listed in table 3.1. The test environment contains two hardware devices, a personal PC and a target microprocessor. They communicate with micro USB. To compile the code for the target devices, HiFive1 rev B use The Freedom Studio, and the rest of the target devices use Code Composer Studio. Both

of these IDE contains their building process and toolchain. Moreover, the debugger tool is also included. We can easily get our results via IDE.

| Device | Memory | Coremark/ HHz |
|---|---|---|
| **HiFive1 Rev b** | 4 MB Flash, 16 KB SRAM | 3.17 |
| **MSP432P401R** | 256 KB Flash, 64 KB RAM | 3.42 |
| **MSP430FR5994** | 256 KB FRAM, 8 KB SRAM | 1.11 |
| **MSP430G2553** | 16 KB Flash, 512 B SRAM | 1.11 |

Table 3.2: Target Devices.

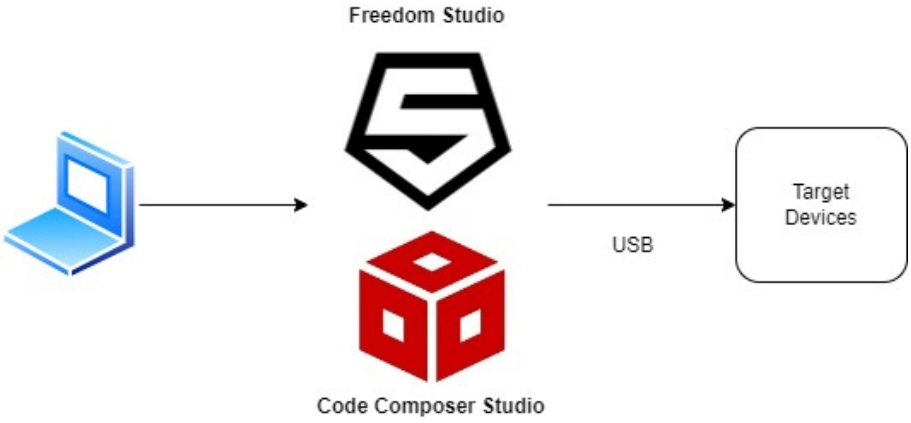


Figure 3.1: Test process.

### 3.2.3 Method of Time Measurement

To find the better energy consumption of MACs running on different resource-constrained devices, we assume less executing time might have lower energy consumption. Our four target devices have their own setting for time measurement.

- **HiFive1 Rev B**

  We use cycle ticks to get the execution time. We can get CPU frequency by calling metal_clock_get_rate_hz function on HiFive1 which value is 16MHz, which refers to

1 second. The total ticks of execution time are equal to the difference between the end and start cycles. The result will be the total ticks divided by the CPU frequency.

```
  // Sample Timer for HiFive1 Rev B
#define CPU_FREQ
    metal_clock_get_rate_hz(&__metal_dt_clock_4.clock)
#define read_csr(reg) ({ unsigned long __tmp; \
  asm volatile ("csrr %0, " #reg : "=r"(__tmp)); \
  __tmp; })

int main() {
   uint64_t start_ticks, mcycleh0, end_ticks, mcycleh1;
   long total_ticks;
   int a = 0;
   do {
      mcycleh0 = read_csr(mcycleh);
      start_ticks = read_csr(mcycle);
      test_function();
      end_ticks = read_csr(mcycle);
      mcycleh1 = read_csr(mcycleh);
      total_ticks = (long)(mcycle1 - mcycle0);
   } while (mcycleh0 != mcycleh1);
   return 0;
}
```

- **MSP432P401R**

  This device is relatively simple. We use The interrupt to increment our ticks. The

built-in function "MAP_SysTick_setPeriod" provided by Texas Instruments can set the period to 1 millisecond per tick.

```
void SysTick_Handler(void) {
    ticks++;
}
#define startTimer() MAP_SysTick_setPeriod(3000); /* 1ms resolution */ \
                     MAP_SysTick_enableInterrupt();                  \
                     /* Enabling MASTER interrupts */                \
                     MAP_Interrupt_enableMaster();                   \
                     MAP_SysTick_enableModule(); /* Start timer */ \
                     unsigned int start = ticks;
```

- **MSP430FR5994 & MSP430G2553**

  These devices use timer capture to evaluate performance. We use ALCK(32.767 kHz). Since we set the divider as eight, ALCK is divided by 8. The frequency of the timer is 4.096 kHz. The start timestamp is before the encrypting function, and the end timestamp is after encrypting function. The elapsed time is equal to the difference between the start timestamp and end timestamp multiplied by one over 4.096 kHz.

```
// Sample Timer for MSP430 family
int main() {
    TA0CTL |= TACLR;      // reset timer
    TA0CTL |= MC_2;       // put into continuous mode
    TA0CTL |= TASSEL_1;   // choose ALCK
    TA0CTL |= ID_3;       // divided by 8
```

```
    TA0CCTL0 |= CAP;       // put CCR0 to capture mode
    TA0CCTL0 |= CM_3;      // sensitive to both edge
    TA0CCTL0 |= CCIS_2;    // GND
    int clock_start = 0;
    int clock_end = 0;
    TA0CCTL0 ^= CCIS0;
    clock_start = TA0CCR0; // get start timestamp
    // test function
    test_function();
    TA0CCTL0 ^= CCIS0;
    clock_end = TA0CCR0;  // get end timestamp
    return 1;
}
```

# Chapter 4

# Experiments & Results

This chapter shows the result of the measurement. The AES-128, SHA256, HMAC, and CMAC will test, respectively. The test focuses on execution time which is essential for knowing the better implementation of cryptographic algorithms for target devices.

## 4.1 AES-128 Result

Three implementations are executed. The key size is 128-bit. All this implementation will test in a single block without any chaining mode.

| Algorithm | Key size | Source | Target Devices | Execution Time |
|---|---|---|---|---|
| AES | 128 bit | Tiny-AES | MSP432P401R | 4.7 ms |
| | | | HiFive1 rev B | 5.7 ms |
| | | | MSP430FR5994 | 40 ms |
| | | | MSP430G2553 | 59.8 ms |
| | | Brian Gladman | MSP432P401R | 1.6 ms |
| | | | HiFive1 rev B | 22.7 ms |
| | | | MSP430FR5994 | 36.6 ms |
| | | | MSP430G2553 | Out of Memory |
| | | Mbedtls | MSP432P401R | 16.5 ms |
| | | | HiFive1 rev B | 12.5 ms |
| | | | MSP430FR5994 | Out of Memory |
| | | | MSP430G2553 | Out of Memory |

Table 4.1: AES-128 results.

Table 4.1 shows the overall results. The AES-128 has good performance compared to other crypto algorithms. For MSP432, The Gladman AES is more suitable for MSP432. But the time difference between Gladman AES and Tiny AES is very small. For HiFive1 rev B, The Tiny AES has the best performance. The MbedTLS is in second place, but MbedTLS is more suitable for HiFive1 rev B. We can see that other implementations running on HiFive1 rev B are slower than MSP432 except for MbedTLS. The advantage of MbedTLS is that the performance is not slow, and it is open source and has its license. The MSP430FR5994 and MSP430G2553 have limited resources, so the performance is slower than HiFive1 rev B and MSP432. The MbedTLS implementation is both out of memory for MSP430FR5994 and MSP430G2553. For MSP430G2553, only one implementation is suitable, which is Tiny AES. For MSP430FR5994, like the MSP432, the Gladman AES is faster than the Tiny AES.

## 4.2 SHA256 Result

| Algorithm | Key size | Source | Target Devices | Execution Time |
|---|---|---|---|---|
| SHA | 256 bit | Allan Saddi | MSP432P401R | 3.1 ms |
| | | | HiFive1 rev B | 8.1 ms |
| | | | MSP430FR5994 | 209 ms |
| | | | MSP430G2553 | 227.2 ms |
| | | Brian Gladman | MSP432P401R | 2.3 ms |
| | | | HiFive1 rev B | 63.39 ms |
| | | | MSP430FR5994 | 229 ms |
| | | | MSP430G2553 | Out of Memory |
| | | Mbedtls | MSP432P401R | 3.6 ms |
| | | | HiFive1 rev B | 18.12 ms |
| | | | MSP430FR5994 | 166.9 ms |
| | | | MSP430G2553 | 276.3 ms |

Table 4.2: SHA256 results.

SHA256 also has simple operations. The overall performance is acceptable. All three implementations running on MSP432 have a tiny difference, so they are all suitable for MSP432.

For HiFive1 rev B, The Allan Saddi implementation is the fastest. Although Gladman is relatively slow, the performance result is also acceptable. It is reasonable that the performance of MSP430FR5994 and MSP430G2553 is slower than the other two devices. Only Gladman is out of memory when it executes on MSP430G2553. Execution times are very similar in the rest of implementations.

## 4.3 MACs

### 4.3.1 HMAC Result

| MACs Type | Algorithm | Source | Target Devices | Execution Time |
|---|---|---|---|---|
| HMAC | SHA | Allan Saddi | MSP432P401R | 7.1 ms |
| | | | HiFive1 rev B | 14.17 ms |
| | | | MSP430FR5994 | 408.9 ms |
| | | | MSP430G2553 | Out of Memory |
| | | Brian Gladman | MSP432P401R | 5.7 ms |
| | | | HiFive1 rev B | 110.39 ms |
| | | | MSP430FR5994 | 435.5 ms |
| | | | MSP430G2553 | Out of Memory |
| | | Mbedtls | MSP432P401R | 8.1 ms |
| | | | HiFive1 rev B | 24.68 ms |
| | | | MSP430FR5994 | 323.5 ms |
| | | | MSP430G2553 | Out of Memory |

Table 4.3: HMAC results.

The HMAC implementation is MAC operation and SHA256 operation. So comparing the three implementations with each other, the results are very similar. With the same devices, the only time difference between SHA and HMAC implementation is MAC operation. On the MSP430G2553, most of the SHA implementation work on it, but HMAC is all out of memory due to its resource.

### 4.3.2 CMAC Result

| MACs Type | Algorithm | Source | Target Devices | Execution Time |
|---|---|---|---|---|
| CMAC | AES | Tiny-AES | MSP432P401R | 165.9 ms |
| | | | HiFive1 rev B | 109.82 ms |
| | | | MSP430FR5994 | Out of Memory |
| | | | MSP430G2553 | Out of Memory |
| | | Brian Gladman | MSP432P401R | 73.7 ms |
| | | | HiFive1 rev B | 160.32 ms |
| | | | MSP430FR5994 | Out of Memory |
| | | | MSP430G2553 | Out of Memory |
| | | Mbedtls | MSP432P401R | 74.4 ms |
| | | | HiFive1 rev B | 33.11 ms |
| | | | MSP430FR5994 | Out of Memory |
| | | | MSP430G2553 | Out of Memory |

Table 4.4: CMAC results.

CMAC uses AES-128 CBC mode, so the performance is slower than AES. It is still in reasonable time on MSP432 and HiFive1 rev B. For MSP432, the most suitable implementation is also Gladman AES. For HiFive1 rev B, The most suitable implementation is MbedTLS. For The MSP430 family, CMAC is not suitable for both devices due to the CBC mode. The reason that causes them out of memory is total memory usage for CMAC is 5,294 bytes. For MSP430G2553, it only has 512 Byte RAM. For MSP430FR5994, it has 8k RAM, but the GCC runtime library uses more global and static variables. In the implementation, the plaintext size is 507 bytes which are too large for MSP430. Even if we reduce the length of plaintext, the implementation also needs to initialize a global two dimension char array that is too big to fit in the RAM region.

# Chapter 5

# Conclusions & Future Work

## 5.1 Conclusions

This thesis is trying to find out the suitable implementation of MAC for embedded systems. The implementation we chose was open source and relatively popular in the community. Our four devices have different ranges of resource limitations that can be used as a reference for other devices that have similar features. MAC schema needs to cooperate with SHA or AES. Therefore, we initially chose different AES and SHA implementations, and then we chose HMAC and CMAC. The performance result also includes the AES and SHA algorithms because it is essential for HMAC and CMAC.

In order to get reliable performance results, we have to make sure the time measurement is accurate. So we tried different ways to test the code. The final version of the time measurement is relatively accurate. We should note that the results of performance time do not apply to all embedded systems. However, the result of this thesis is valuable.

Referring to the original research questions, we can conclude that AES and SHA implementations are not all suitable for those devices that have more constrains resource like MSP430FR5994 and MSP430G255. The reason that causes the performance difference of AES or SHA on the same board depends on the implementation process. The device's computational feature makes it fit for some implementation. Some devices good at addition operation and some devices good at multiply operation. Although the overall algorithm

is the same, some implementation has their initial step or validation step that causes the performance difference. For example, AES uses SBOX to get the value. The SBOX is a constant array on the Tiny AES, but the SBOX is a computational value on MbedTLS. So we can find that the performance of Tiny AES is always better than MbedTLS. This is just one factor that causes the performance difference on the same board. The data structure and pointer would also cause performance differences. The HMAC is suitable for all our target devices except the MSP430G2553, which only have 16 KB Flash memory, 512 B SRAM. The CMAC is not suitable for MSP430FR5994 and MSP430G255 because CBC mode need more resource. Focusing on energy consumption. For single devices, execution time tells us the computational duration, which means it has lower energy consumption. The result of execution time could tell us which implementation consumes less energy on certain boards.

## 5.2 Future Work

Currently, we only chose three AES and SHA implementations, and we do not optimize them. The implementation is directly from the open-source. We extract the function we need and do not modify any data structure or process. The main goal of this thesis is to test the original implementation. The next task is going. Since the HMAC and CMAC are relatively simple, it is possible that all our chosen implementation could be worked on MSP430FR5994 and MSP430G255 if we do further optimization. To optimize the implementation, First, we should know about the computational ability of the devices and which operation would be fast on them. We can test it by timing the program section. Second, we should remove unnecessary processes and check whether the initial step could be removed or not. The last step is to use proper data structure and initial fit size array to reduce stack and heap size. The second task is going to include more devices. There are many microcontrollers out there

from different companies. We could understand MACs on microprocessors more after we contain more range of resource-constrained devices.

# Bibliography


[1] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 1–15, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[2] Christof paar, Jan Pelzl. *Understanding Cryptography.*

[3] C. Craven. What is the advanced encryption standard (aes). https://www.sdxcentral.com/security/definitions/what-is-advanced-encryption-standard-aes-definition/.

[4] B. Gladman. Brian gladman aes. https://github.com/BrianGladman/aes.

[5] B. Gladman. Brian gladman sha256. https://github.com/BrianGladman/sha.

[6] B. Jovanovic. A not-so-common cold: Malware statistics in 2022. http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/, 2022.

[7] V. Keränen. Cryptographic algorithm benchmarking in mobile devices. Master's thesis, University of Oulu, 2013.

[8] G. C. Kessler. An overview of cryptography. Master's thesis, 1998.

[9] MbedTLS. Mbedtls aes. https://github.com/Mbed-TLS/mbedtls.

[10] MbedTLS. Mbedtls sha256. https://github.com/wolfeidau/mbedtls/blob/master/source/sha256.c.

[11] A. Z. Mustafeez. What is the aes algorithm? https://www.educative.io/edpresso/what-is-the-aes-algorithm.

[12] C. Point. Check point research: Cyber attacks increased 50% year over year. https://blog.checkpoint.com/2022/01/10/check-point-research-cyber-attacks-increased-50-year-over-year/.

[13] A. Saddi. Allan saddi sha256. https://opensource.apple.com/source/clamav/clamav-158/clamav.Bin/clamav-0.98/libclamav/sha256.c.auto.html.

[14] Tiny-AES. https://github.com/kokke/tiny-AES-c.

[15] A. G. Wadday. Study of wimax based communication channel effects on the ciphered image using maes algorithm. https://www.researchgate.net/publication/324796235_Study_of_WiMAX_Based_Communication_Channel_Effects_on_the_Ciphered_Image_Using_MAES_Algorithm.

[16] Wikipedia. Block cipher mode of operation. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation.

[17] Wikipedia. Confusion and diffusion. https://en.wikipedia.org/wiki/Confusion_and_diffusion.