Eduvest - Journal of Universal Studies
Volume 5 Number 3, March, 2025
p- ISSN 2775-3735 - e-ISSN 2775-3727

STUDENT ID CARD SECURITY SYSTEM BASED ON ADVANCED ENCRYPTION STANDARD AND ROLLING CODE

Reza Syah Pahlevi Natanagara, Vera Suryani
Telkom University, Indonesia
Email: rezassyah@student.telkomuniversity.ac.id, verasuryani@telkomuniversity.ac.id

ABSTRACT

Student identification cards (KTM) are currently vulnerable to fraudulent activities and counterfeiting, including replay attacks. KTMs have several functions, such as providing access to parking areas, lecture buildings, libraries, and student attendance systems. Replay attacks allow attackers to falsify student attendance and gain unauthorized access to campus facilities. Therefore, the main objective of this Final Project is to enhance the security of KTMs by implementing a security system based on Advanced Encryption Standard (AES) and rolling code to prevent replay attacks. The method used involves encrypting data on KTMs using the AES-128 algorithm and applying rolling code in the KTM security system. The test results show that the implementation of a security system based on AES-128 encryption and rolling code on KTMs can enhance the security and authenticity of KTMs and prevent fraudulent activities and counterfeiting, including card duplication. Thus, this Final Project provides a solution to improve KTM security and offers significant benefits for KTM users.

KEYWORDS *Replay Attack, Radio Frequency Identification, Advanced Encryption Standard, Near Field Communication, Rolling code*

INTRODUCTION

Student identity cards are one of the important identities for students in carrying out lecture activities and other campus activities. However, currently student identity cards are still vulnerable to fraud and counterfeiting, such as copying student cards. This is due to the lack of an adequate security system on the student identity card. Therefore, it is necessary to conduct research to improve the security of student identity cards.

This final project focuses on improving the security of student ID cards because RFID-based student cards can be easily copied using the MIFARE Classic Tool application available for Android phones by utilizing the *Near Field Communication* or NFC for short. Student ID card using a card *Radio Frequency Identification* or commonly abbreviated as RFID type Mifare Classic 1K made by NXP Semiconductor, this card uses a frequency of 13.56 MHz with a memory capacity of 1KB with a distance from the reader up to 100 mm. This type of card is commonly used as an access card, employee card, and public transportation card (NXP Semiconductors, 2014). The card media used to store copies using an RFID card with the serial number EL-MF1WA-CNB, this card runs at a frequency of 13.56 MHz and has a memory size of 1 KB. Unlike other RFID cards, this card was created to be rewritten many times in sector 0, sector 0 is the sector where the UID is stored.

In addition, Android phones with NFC features can be utilized to simulate student cards by utilizing the Card Emulator PRO application available for Android phones. This app requires ROOT access. To use this application, users only need to enter the UID of their student card to be able to simulate the card. After success, users will only need to stick their mobile phone to the student card reader and it will be detected as a valid student card.

This research aims to improve the security of student identity cards by using *the Advanced Encryption Standard* (AES)-128 encryption system and *rolling code* technology. With the AES-128 encryption system, the data on the student ID card is encrypted so that it cannot be read by unauthorized people. Whereas *rolling code* technology generates a different random code each time a card is used. This random code is 6 characters long with a combination of numbers from 0 to 9 and all alphabets from a to z, both uppercase and lowercase letters, making it difficult for unauthorized people to access such access.

Through this Final Project, it is hoped that improvements can be made to the student ID card security system by implementing AES-128 encryption and *rolling codes*. By integrating an encryption system with additional codes that are always changing after students tap the card, it can prevent copying of student cards.

The formulation of the problem in this study is: 1. How to implement a security system based on AES encryption and *rolling code* on student ID cards? 2. Does the application of AES encryption and *rolling code* on student ID cards cause a heavy computing burden or not? The objectives to be achieved in this study are: 1. Integrating a student ID card security system based on AES encryption and *rolling code* to protect valid user data and access. 2. Evaluate the impact of implementing AES encryption and *rolling code* on student ID cards related to processing time and computing load.

RESEARCH METHOD

In this study, a specific methodology is needed to complete the research on the security system of student identity cards based on Advanced Encryption Standard (AES) and rolling codes. The stages of this research methodology are explained as follows: In the first stage, problems are identified to assess the vulnerability of KTM, especially against counterfeiting and replay attacks. The focus of the research is to fix these security problems by implementing AES-128 encryption and rolling code to ensure the security of data on student cards.

The second stage is a literature study, where various theories and applications of Radio Frequency Identification (RFID), Near Field Communication (NFC) technology, as well as AES encryption algorithms and rolling code are discussed. This literature includes case studies of attacks on RFID systems as well as existing security solutions, including microcontroller technology that can be used in the implementation of security systems for KTM.

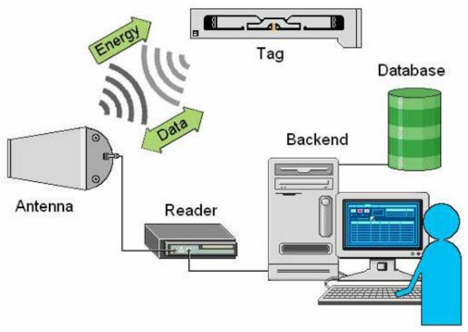
Furthermore, at the system design stage, hardware and software are designed that support the implementation of AES-128 encryption and rolling code. The implementation process involves using a microcontroller to write data on the RFID card, which includes an encrypted UID and random code. Finally, the system is tested through evaluation to ensure the success of encryption and the implementation of rolling code in preventing replay attacks, as well as to evaluate the effectiveness and performance of the security system that has been built.

RESULT AND DISCUSSION

Radio Frequency Identification (RFID)

RFID (*Radio-Frequency Identification*) is a method of identifying objects using electromagnetic waves(Kamaludin et al., 2018). This technology consists of two main components *Tags* and readers. *Tags* Comes with *Microchip* and an antenna to store and transmit data, while a reader has an antenna to receive electromagnetic waves from *Tags*.

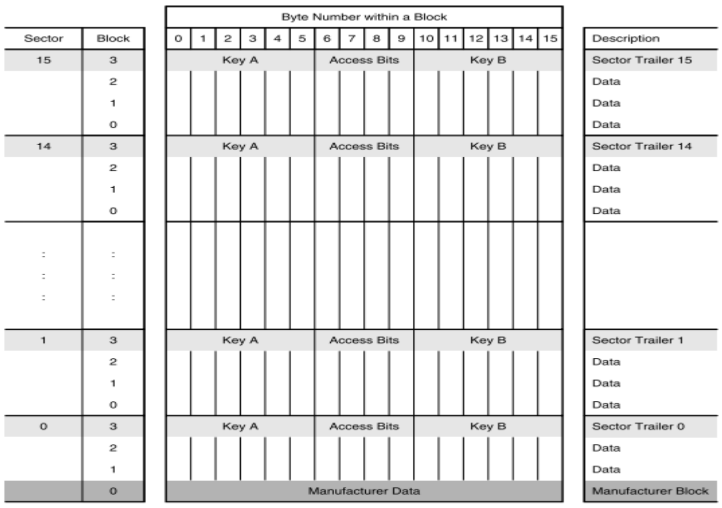
RFID systems generally consist of several *Tags* RFID, a reader, and a system *backend*. *Tags* RFID stores information, while the reader emits radio signals to activate and receive data from *Tags* aforementioned. This data is then processed and stored by the system *backend*(Xiao et al., 2008). RFID systems generally have an image as seen in Picture 1.



Picture 1. RFID working system

RFID is used in a variety of applications such as security, access management, and transportation, making it suitable for collecting large amounts of data. There are different types of RFID that operate at different frequencies, depending on the specific needs. Low frequencies (125/134 KHz) are used for entrance access, high frequencies (13.56 MHz) for tickets and payment systems, and ultra-high frequencies (860/960 MHz) for tracking goods in warehouses. Proper frequency selection is essential to ensure effective communication between *Tags* RFID and RFID readers.(Supriyati et al., 2019).

Memory structure of the card *Mifare Classic 1k* which consists of 16 sectors each of which has 4 blocks of memory. Each block of memory consists of 16 bytes. The first block in sector 0 is called the producer block (*Manufacturer Block*) containing the card's serial number (UID) and immutable manufacturer data. The last block in each sector is *Sector Trailer*, which stores key A, *Access Bits*, and key B which is used to secure access to the sector. Other blocks are used to store data(Rahim et al., 2019). Memory structure *Mifare Classic 1k* can be seen on Picture 2.



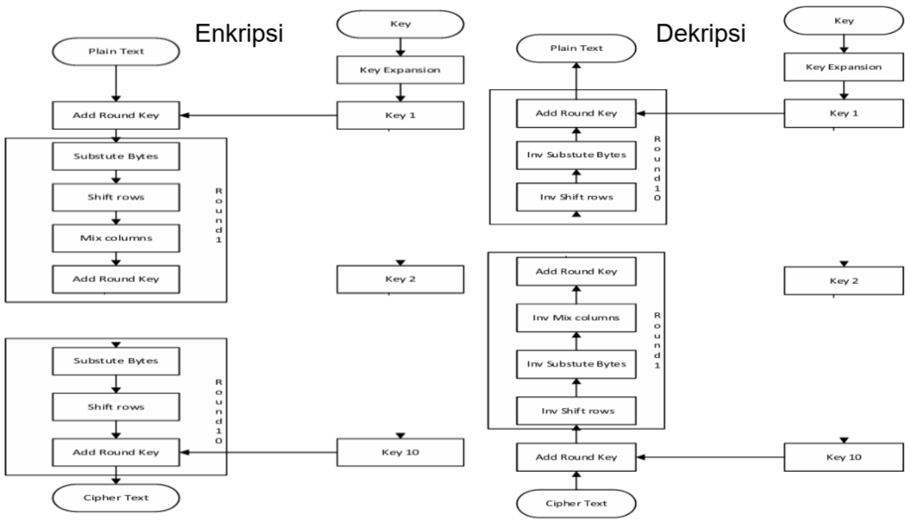
Picture 2. RFID structure on Mifare Classic 1K card

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is one of the methods used in securing data and information. AES is a proposed cryptographic algorithm as a replacement for *Data Encryption Standard* (DES) because of its better security. The AES algorithm was invented by Vincent Rijmen and Joan Daemen from Belgium, and until now there is no known type of attack that can solve this algorithm. AES uses blocks *Cipher* symmetric that is capable of encrypting and decrypting information(Simbolon et al., 2020).

AES uses blocks of data that are 128 bits long and keys that are 128, 192, or 256 bits long. AES uses several types of transformations, such as *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*, to encrypt data repeatedly over multiple rounds. AES is considered one of the most secure encryption algorithms and is widely used in various applications and systems, such as data security on computer networks, banking, and the military. AES also has advantages in speed and efficiency in its use(Tulloh et al., 2016).

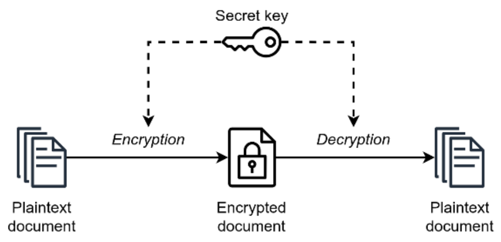
The encryption process begins with *Plaintext* which went through a series of transformations such as *Add Round Key*, *Substitute Bytes*, *Shift Rows*, and *Mix Columns*, which is repeated for 10 rounds to produce *Ciphertext*. Instead, the decryption process reverses those steps, starting with *Ciphertext* and through *Inverse Shift Rows*, *Inverse Substitute Bytes*, and *Inverse Mix Columns*, to return the data to *Plaintext*. Both processes use a series of keys generated from *Key Expansion* for each round(Dang & Vo, 2019). The AES encryption and decryption process can be seen in Picture 3.



Picture 3. AES encryption and decryption process

Symmetric Cryptography

Symmetric cryptography, also known as single-key cryptography, is a cryptographic method that uses a single key to encrypt and decrypt information. In this method, the sender and receiver must agree on the same key, which is known only to both of them. Original information, the so-called *Plaintext*, along with a 16-bit key, are processed in an encryption process to generate random data known as *Ciphertext*. Long *Ciphertext* This is equal to the length *Plaintext*. Decryption is the opposite process of encryption, using the same key to return *Ciphertext* become *Plaintext*(Widura et al., 2015).



Picture 4. Symmetric Cryptography

Application of Rolling Code

Rolling Code is a security protocol where every process is carried out, the system will generate a unique code called *rolling code*, The system will grant access only if the code received matches the code generated by the system(Alkalah, 2016). On KTM's security system, *Rolling Code* This is *Plaintext* which will be made as long as 6 characters randomly arranged from a combination of numbers between 0 to 9 and all letters, whether capital letters or not, *Rolling Code* This will be compiled every time the decryption and encryption process is carried out on KTM.

UID								Rolling Code					

Picture 5. Structure of Rolling Code Implementation with UID

Implementation of AES-128 and Symmetric Cryptography

In a research made by Yuda Purwanto and colleagues entitled "Data Encryption on RFID Cards Using the AES-128 Algorithm for Public Transportation in Bandung Regency" in 2015. In this study they created an RFID security system using AES-128, AES-128 was chosen because of several things such as low memory usage accompanied by the use of Symmetric Cryptography which does not require additional memory and processes, this is suitable for the Mifare Classic 1K card which has a small memory, besides AES-128 has efficient encryption time because the process is carried out less than other types of AES and compatibility is supportive. many tools with limited specifications(Widura et al., 2015).

Table 1. Comparison of the Number of Processes Performed by AES

	Number of Processes
AES-128	10
AES-192	12
AES-256	14

System Planning

Devices used

In a student card security system, there are three main devices used. These devices can be described as follows:

1. **RFID Module**

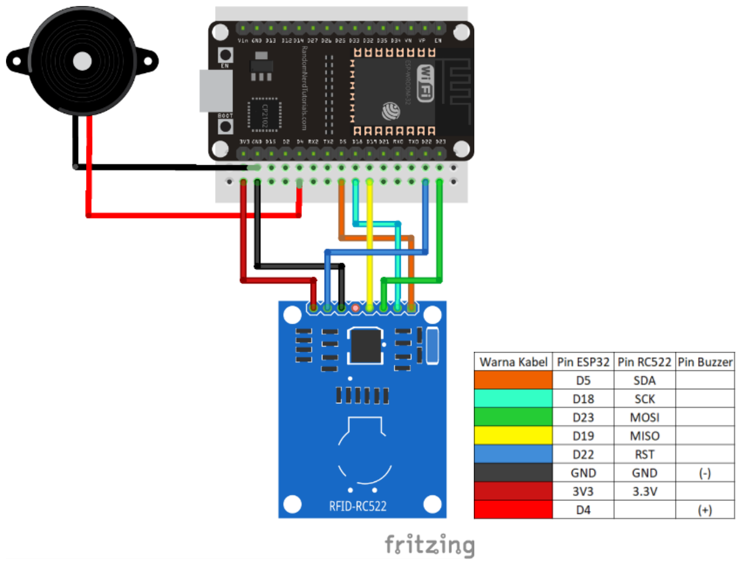
   The RFID module serves as a student card literacy device. This module is composed of several devices with their respective functions, as can be seen in

Table **2**.

Table 2. RFID module device list

No.	**Device/Sensor**	**Information**
1	ESP32 Devkit V1	As a *microcontroller*, it converts data between Base64 and Byte Array formats, breaks down and combines data, and connects devices to servers.
2	RFID-RC522	Reader and author of the Student Identity Card (KTM).
3	Buzzer	As a notification to give an indication of the card validation status.

The devices are assembled by following the device schema as depicted in the Picture 6. The RFID-RC522 module connects with the ESP32 Devkit V1 via an SPI interface, where the ESP32 acts as the primary controller. When KTM is affixed to an RFID reader, the data from KTM (including UID and stored information) is converted by the ESP32 into a format that can be further processed. After validation and encryption is performed, the results are sent back to the RFID module to rewrite the updated data on the card. In addition, the buzzer will emit a sound to indicate whether access is accepted or denied based on data validation.



Warna Kabel	Pin ESP32	Pin RC522	Pin Buzzer
🟧	D5	SDA	
🟦	D18	SCK	
🟩	D23	MOSI	
🟨	D19	MISO	
🟦	D22	RST	
⬛	GND	GND	(-)
🟥	3V3	3.3V	
🟥	D4		(+)

Picture 6. Sensor design schematic

2. Server Laptop

The Laptop server serves as the processing center and *database* for KTM's security system. In this study, the laptop used as a server is a Lenovo Ideapad Gaming 3I with the following specifications: Intel Core i5-10300H processor, 16 GB RAM, and GTX 1650 GPU. The laptop runs a local server that manages the KTM user database as well as stores the results of the encryption and *rolling code* used for card validation. The processes performed by the server include:

- **UID and Card Data Validation:** The server receives data from the ESP32 in the form of UIDs and encrypted data sent over the Wi-Fi network. The UID is then matched with *a database* to identify the cardholder.
- **Decryption and Verification:** Data that is still in the form of *ciphertext* is decrypted by the server using a predefined encryption key. The server verifies that the *received rolling code* matches the one in the *database*.
- **Rolling Code Update:** After successful verification, the server will generate *a new rolling code* and save it to the *database*. *This rolling code* is then re-encrypted along with the UID to be stored in KTM.
- **Database Management:** The server is also responsible for maintaining the integrity of the *database*, ensuring that any UIDs and *rolling codes* are stored securely and can only be accessed by legitimate systems.

3. Android Smartphones

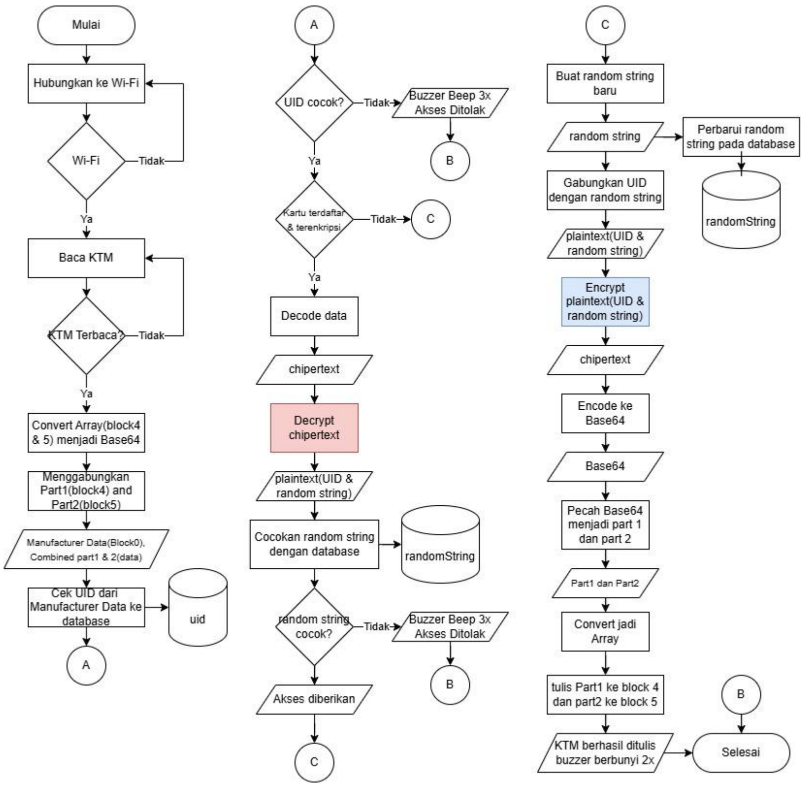
Android smartphones are used as a tool to duplicate cards and simulate Student Identity Cards (KTM). In this study, the *smartphone* used is POCO F4, which has the following specifications:

- **Processor:** Qualcomm Snapdragon 870
- **RAM:** 6 GB
- **Internal Storage:** 256 GB
- **Operating System:** Android 14 based on Pixelstar OS
- **Features:** *Near Field Communication* (NFC), which allows for short-range communication with RFID devices

The smartphone is equipped with an NFC feature, which allows the device to communicate with the RFID module. In this study, POCO F4 was used for two main functions:

- **Card Duplication:** By using the **Mifare Classic Tool app**, *smartphones* can read and copy data from the original KTM card to a blank card that supports repeated reads and writes. This duplication process includes copying the UID and other data stored in certain sectors of the card.
- **KTM Simulation:** The Card Emulator Pro **app** installed on the POCO F4 is used to simulate KTM. The app leverages *root access* to allow the device to mimic the UID of the original card. After the UID is inserted into the application, the POCO F4 can be used like a physical KTM, by attaching *a smartphone* to the RFID-RC522 reader.

System flowchart


Mulai
Hubungkan ke Wi-Fi
Wi-Fi
Tidak
Ya
Baca KTM
KTM Terbaca?
Tidak
Ya
Convert Array(block4 & 5) menjadi Base64
Menggabungkan Part1(block4) and Part2(block5)
Manufacturer Data(Block0), Combined part1 & 2(data)
Cek UID dari Manufacturer Data ke database
uid
A
A
UID cocok?
Tidak
Buzzer Beep 3x Akses Ditolak
B
Ya
Kartu terdaftar & terenkripsi
Tidak
C
Ya
Decode data
chipertext
Decrypt chipertext
plaintext(UID & random string)
Cocokan random string dengan database
randomString
random string cocok?
Tidak
Buzzer Beep 3x Akses Ditolak
B
Akses diberikan
C
C
Buat random string baru
random string
Perbarui random string pada database
randomString
Gabungkan UID dengan random string
plaintext(UID & random string)
Encrypt plaintext(UID & random string)
chipertext
Encode ke Base64
Base64
Pecah Base64 menjadi part 1 dan part 2
Part1 dan Part2
Convert jadi Array
tulis Part1 ke block 4 dan part2 ke block 5
B
KTM berhasil ditulis buzzer berbunyi 2x
Selesai

Picture 7. KTM safety system flowchart

Figure 7 illustrates the workflow of the Student Identity Card (KTM) security system that has been implemented using encryption *Advanced Encryption Standard* (AES) and *Rolling Code*. The process starts with the system trying to connect to a Wi-Fi network. If the Wi-Fi connection is successful, the system proceeds to read the KTM using the RFID module. If the card is successfully read, the data from block 4 and block 5 of the card containing important information is converted to Base64 format.

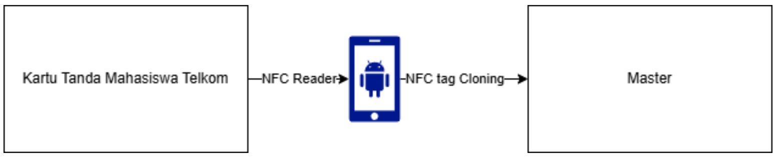
Next, the system will retrieve the UID from KTM and check it for compatibility with the data in the *database*. If the UID does not match the *database*, direct access is denied, and the buzzer will sound three times as a sign of failure. If the UID matches, the system checks if the card is registered and encrypted before. Otherwise, the process will continue to create a new *random string* and update the card data.

The verification process involves decrypting *the ciphertext* inside the card to check if the *stored random string* matches the one in the *database*. If *the random string* doesn't match, re-access is denied, and the buzzer will sound three times. However, if *the random string* matches, access will be granted, and the system will generate *a new* random string.*This random string* is then combined with the UID to form a new *plaintext*, which is then encrypted into *ciphertext*.

After the encryption process, the encrypted data is converted back to Base64 format, broken into two parts (part 1 and part 2), and written to block 4 and block 5 on the KTM. If the writing is successful, the buzzer will sound twice as a sign that the process has been successful. Thus, every time KTM is used, the data in it is always updated with a new *random string*, which ensures additional security against duplication attempts and *replay attacks*.

Test Scenarios

To test the effectiveness of the system created, several test scenarios were made, but due to the limitations of the original KTM, before testing the original KTM system will be duplicated to the Mifare Classic 1K card that supports repeated read/write which will be called Master.



Picture 8. Making KTM Masters

KTM duplicate testing with Card Emulator app

Card Emulator is an Android application created by yuanwofei, this application will be installed and run on POCO F4 devices with the Pixelstar operating system based on Android 14, to run the Card Emulator application the device used must have the NFC feature and the android device used must have *root* access. This test aims to find out whether the security system can prevent unauthorized access from the simulated KTM Master from an android device with the Card Emulator application installed.

Testing of duplicate KTM to other cards

The Mifare Classic 1K card that can be read and written is repeatedly traded freely, therefore a test was carried out to duplicate the Master card to another card to see if the system could deny the unauthorized access. Duplication is done using the Mifare Classic Tool application installed on POCO F4 android devices with the Pixelstar operating system based on Android 14, to run this application, the device is not required to have Root access, but having the NFC feature is an obligation. Because in the original KTM the data used for student identification is *sector* 0 of KTM, so in this trial there will also be a duplicate for *sector* 0 of KTM master.

Runtime and memory usage testing

This test aims to measure the computational load during the reading and writing process of a KTM card. The test was carried out by bringing the Master card closer to the RFID-RC522 10 times on a system with Intel Core i5-10300H specifications and 16GB of RAM.

Testing Method:

1. **Measuring the Runtime:**
   - **Logging:** The Runtime is recorded via *the output serial logs* of the Arduino IDE application. This time is measured from the time the card is read by RFID until the data re-checking process is completed.
2. **Measuring CPU Time:**

- **Code:** *cpuTime* is logged on the server side using microtime(true); to calculate how long the CPU is used during the process, from the time the data is read to the time the data is stored in the *database*.
- **Calculation:** The CPU time difference is calculated by the formula:

$$CPU\ Time\ (ms) = (\$endTime - \$startTime) \times 1000 \qquad (1)$$

3. **Measuring the Runtime:**
   - **Code:** Memory usage is logged with memory_get_usage(); to find out how much memory is used during the process.
   - **Calculation:** The difference in memory usage is calculated by the formula:

$$Memory\ Usage\ (bytes) = \$endMemory - \$startMemory \qquad (2)$$

   - **Memory Usage Percentage Calculation:** The percentage of memory used is calculated by the formula:

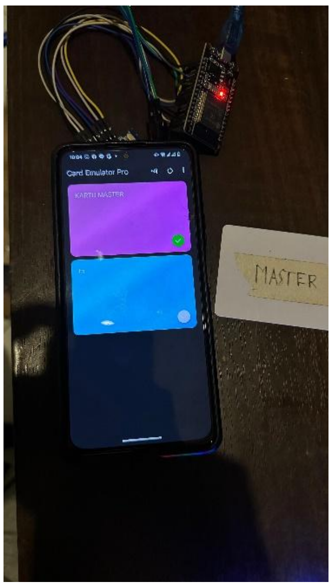
$$Memory\ Usage\ \% = \left(\frac{Memory\ Usage\ (bytes)}{Total\ RAM\ (bytes)}\right) \times 100 \qquad (3)$$

4. **Evaluation**
   This chapter describes the results of testing and analysis of KTM's security system based on the permitting scenario described in the previous chapter.

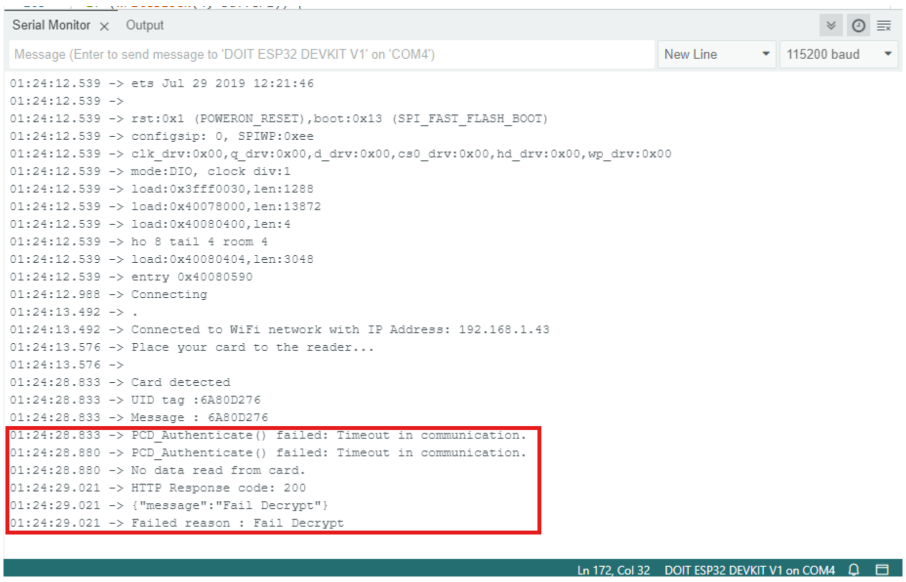
Analysis of Test Results

KTM duplicate testing with Card Emulator app

The test was carried out by copying the UID and simulated with NFC on an android *smartphone* running the Card Emulator application. The test begins by saving the UID into the Card Emulator application, then save it with the name MASTER CARD, after that activate the option to simulate the card, the next stage is to bring the android *smartphone* closer to the RFID-RC522.



Picture 9. Bringing an android smartphone closer to the RFID-RC522


Serial Monitor
Output
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM4')
New Line
115200 baud

```
01:24:12.539 -> ets Jul 29 2019 12:21:46
01:24:12.539 ->
01:24:12.539 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
01:24:12.539 -> configsip: 0, SPIWP:0xee
01:24:12.539 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
01:24:12.539 -> mode:DIO, clock div:1
01:24:12.539 -> load:0x3fff0030,len:1288
01:24:12.539 -> load:0x40078000,len:13872
01:24:12.539 -> load:0x40080400,len:4
01:24:12.539 -> ho 8 tail 4 room 4
01:24:12.539 -> load:0x40080404,len:3048
01:24:12.539 -> entry 0x40080590
01:24:12.988 -> Connecting
01:24:13.492 -> .
01:24:13.492 -> Connected to WiFi network with IP Address: 192.168.1.43
01:24:13.576 -> Place your card to the reader...
01:24:13.576 ->
01:24:28.833 -> Card detected
01:24:28.833 -> UID tag :6A80D276
01:24:28.833 -> Message : 6A80D276
01:24:28.833 -> PCD_Authenticate() failed: Timeout in communication.
01:24:28.880 -> PCD_Authenticate() failed: Timeout in communication.
01:24:28.880 -> No data read from card.
01:24:29.021 -> HTTP Response code: 200
01:24:29.021 -> {"message":"Fail Decrypt"}
01:24:29.021 -> Failed reason : Fail Decrypt
```

Picture 10. The system does not detect the presence of Data

Based on the results of the test, the card simulated by an android smartphone with the Card Emulator application can send a UID that matches the KTM Master, but there is a failure because the application can only simulate the UID not with other data contained on the card. Therefore, it can be concluded that the security system is running well with no access to the simulated KTM with *an android* smartphone.

KTM Duplicate to Other Card Testing

Testing was carried out by duplicating the Master card to a Mifare Classic 1K card that supports repeated reads and writes. This card is henceforth called Clone

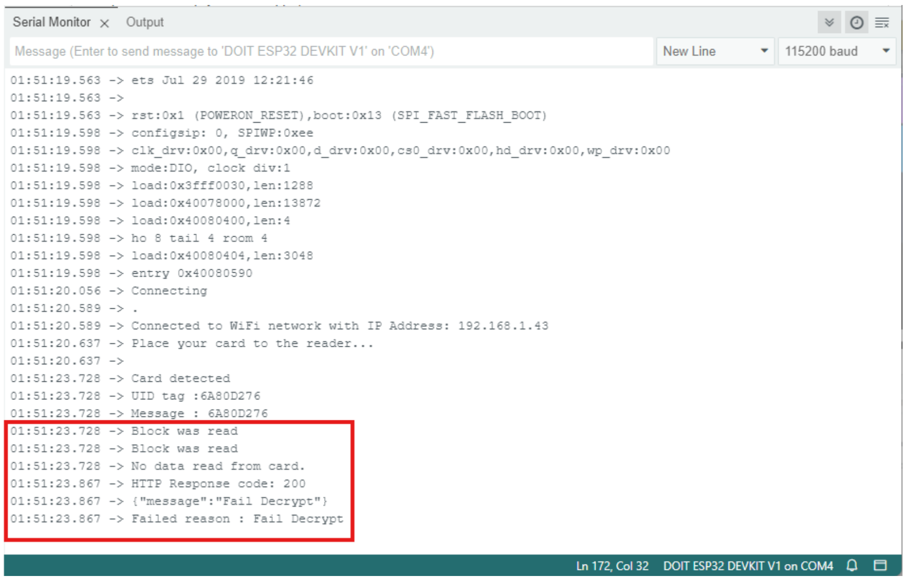
Master. The duplication process is carried out using the Mifare Classic Tool application on *Android smartphones*.

To create a Clone Master, duplication is done by selecting the TOOLS menu, then selecting Clone UID. The following is a comparison table of the contents of the KTM Master and KTM Clone Master that have been successfully duplicated:

Table 3. Comparison of KTM Master and Clone Master contents

	Block	Master	Clone Master
Sector 0	0	6A80D2764E080400475955D141103607	6A80D2764E080400475955D141103607
	1	00000000000000000000000000000000	00000000000000000000000000000000
	2	00000000000000000000000000000000	00000000000000000000000000000000
	3	FFFFFFFFFFFFFF078069FFFFFFFFFFFF	FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector 1	4	4D564A304D316F7964575A446432524C	00000000000000000000000000000000
	5	4F58685455576477546E5530647A3039	00000000000000000000000000000000
	6	00000000000000000000000000000000	00000000000000000000000000000000
	7	FFFFFFFFFFFFFF078069FFFFFFFFFFFF	FFFFFFFFFFFFFF078069FFFFFFFFFFFF

Based on the comparison table, *sector* 0 has the same content, *sector* 0 is the *sector* used to identify each student on the original KTM, but for *sectors* 4 and 5 there is a difference, it can be seen in Clone Master, both *sectors* are empty. Next is to test the Clone Master card is brought closer to the RFID-RC522.


Serial Monitor
Output
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM4')
New Line
115200 baud

```
01:51:19.563 -> ets Jul 29 2019 12:21:46
01:51:19.563 ->
01:51:19.563 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
01:51:19.598 -> configsip: 0, SPIWP:0xee
01:51:19.598 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
01:51:19.598 -> mode:DIO, clock div:1
01:51:19.598 -> load:0x3fff0030,len:1288
01:51:19.598 -> load:0x40078000,len:13872
01:51:19.598 -> load:0x40080400,len:4
01:51:19.598 -> ho 8 tail 4 room 4
01:51:19.598 -> load:0x40080404,len:3048
01:51:19.598 -> entry 0x40080590
01:51:20.056 -> Connecting
01:51:20.589 -> .
01:51:20.589 -> Connected to WiFi network with IP Address: 192.168.1.43
01:51:20.637 -> Place your card to the reader...
01:51:20.637 ->
01:51:23.728 -> Card detected
01:51:23.728 -> UID tag :6A80D276
01:51:23.728 -> Message : 6A80D276
01:51:23.728 -> Block was read
01:51:23.728 -> Block was read
01:51:23.728 -> No data read from card.
01:51:23.867 -> HTTP Response code: 200
01:51:23.867 -> {"message":"Fail Decrypt"}
01:51:23.867 -> Failed reason : Fail Decrypt
```

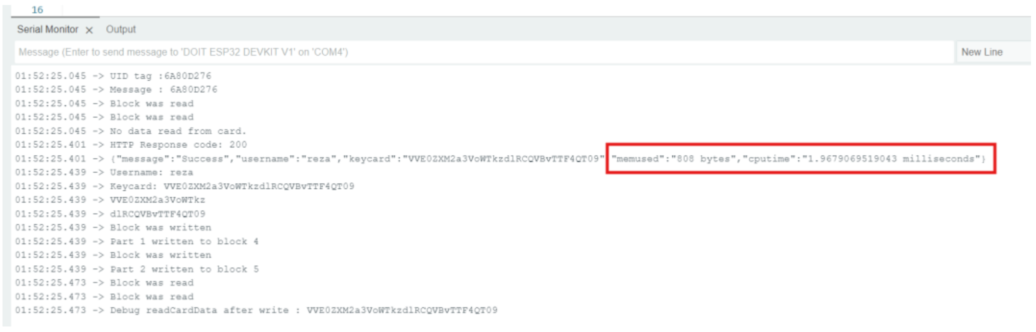
Picture 11. The system cannot decrypt

Based on the test, it can be seen that the difference in the error message given compared to the previous test scenario, in this scenario, *blocks* 4 and 5 are read by the system, but because it is empty, the system cannot decrypt to check whether the data is valid.

Runtime and memory usage test results

The test was carried out by bringing the Master card closer to the RFID-RC522 10 times. The goal is to compare the runtime and the amount of memory used. Testing was conducted on a system with Intel Core i5-10300H specifications and 16GB of RAM.

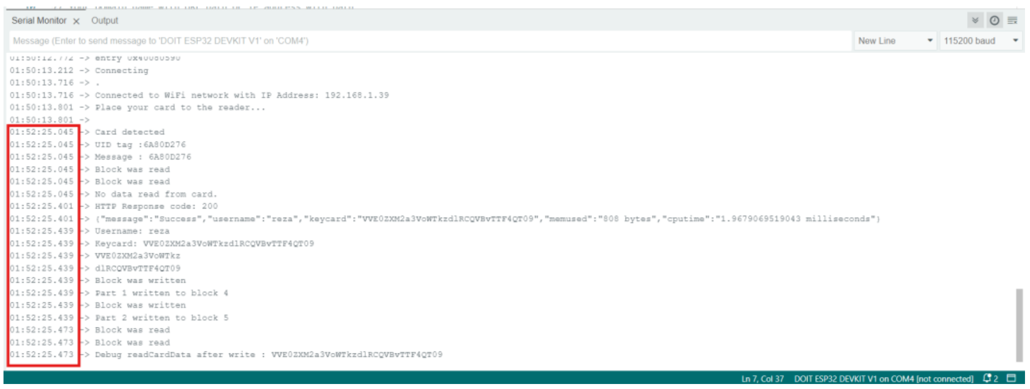
To calculate memory usage and *cpuTime*, using simple code inserted into the *deep* Server. The code starts when the system retrieves the uid and data from the card, then stops when *Update* at *Database* done. The result of the code calculation will be displayed on the *Serial Monitor* Arduino IDE app, example *cpuTime* and the amount of memory displayed on the Arduino IDE application can be seen in the Picture 12.


16
Serial Monitor
Output
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM4')
New Line

```
01:52:25.045 -> UID tag :6A80D276
01:52:25.045 -> Message : 6A80D276
01:52:25.045 -> Block was read
01:52:25.045 -> Block was read
01:52:25.045 -> No data read from card.
01:52:25.401 -> HTTP Response code: 200
01:52:25.401 -> {"message":"Success","username":"reza","keycard":"VVE0ZXM2a3VoWTkzdlRCQVBvTTF4QT09","memused":"808 bytes","cputime":"1.9679069519043 milliseconds"}
01:52:25.439 -> Username: reza
01:52:25.439 -> Keycard: VVE0ZXM2a3VoWTkzdlRCQVBvTTF4QT09
01:52:25.439 -> VVE0ZXM2a3VoWTkz
01:52:25.439 -> dlRCQVBvTTF4QT09
01:52:25.439 -> Block was written
01:52:25.439 -> Part 1 written to block 4
01:52:25.439 -> Block was written
01:52:25.439 -> Part 2 written to block 5
01:52:25.473 -> Block was read
01:52:25.473 -> Block was read
01:52:25.473 -> Debug readCardData after write : VVE0ZXM2a3VoWTkzdlRCQVBvTTF4QT09
```

Picture 12. cpuTime logging from the Arduino IDE

As for the processing time, calculated from the readable Master card, it is marked by *output "Card detected"* until the process of rechecking the data that has been successfully written on the Master card, marked by *output* "Debug readCardData after write : <Base64>". The time is recorded based on the logs that appear on the Arduino IDE application monitor series. An example of the logged logs can be seen in the Picture 13.


Serial Monitor
Output
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM4')
New Line
115200 baud
Ln 7, Col 37
DOIT ESP32 DEVKIT V1 on COM4 [not connected]

```
01:50:13.212 -> Connecting
01:50:13.716 -> .
01:50:13.716 -> Connected to WiFi network with IP Address: 192.168.1.39
01:50:13.801 -> Place your card to the reader...
01:50:13.801 ->
01:52:25.045 -> Card detected
01:52:25.045 -> UID tag :6A80D276
01:52:25.045 -> Message : 6A80D276
01:52:25.045 -> Block was read
01:52:25.045 -> Block was read
01:52:25.045 -> No data read from card.
01:52:25.401 -> HTTP Response code: 200
01:52:25.401 -> {"message":"Success","username":"reza","keycard":"VVE0ZXM2a3VoWTkzdlRCQVBvTTF4QT09","memused":"808 bytes","cputime":"1.9679069519043 milliseconds"}
01:52:25.439 -> Username: reza
01:52:25.439 -> Keycard: VVE0ZXM2a3VoWTkzdlRCQVBvTTF4QT09
01:52:25.439 -> VVE0ZXM2a3VoWTkz
01:52:25.439 -> dlRCQVBvTTF4QT09
01:52:25.439 -> Block was written
01:52:25.439 -> Part 1 written to block 4
01:52:25.439 -> Block was written
01:52:25.439 -> Part 2 written to block 5
01:52:25.473 -> Block was read
01:52:25.473 -> Block was read
01:52:25.473 -> Debug readCardData after write : VVE0ZXM2a3VoWTkzdlRCQVBvTTF4QT09
```

Picture 13. Runtime logging

Table 4. Compute Load Test Results Table

Testing to	cpuTime (ms)	Runtime (ms)	Memory Usage (bytes)	Memory Usage %
1	14.74	296	960	0,0000056%
2	6.32	98	960	0,0000056%
3	14.51	555	960	0,0000056%
4	6.22	167	960	0,0000056%
5	14.11	154	960	0,0000056%
6	13.86	166	960	0,0000056%
7	13.86	199	960	0,0000056%
8	5.48	111	960	0,0000056%
9	13.60	164	960	0,0000056%
10	5.68	144	960	0,0000056%

Based on the results displayed on Table 4, from 10 experiments carried out, several results were obtained such as, *cpuTime* which varies from 5.68 ms to 14.74 ms. This result is obtained by using equation (1) which is entered into the program inside the server, This variable result can be affected by several factors such as complexity *random string* and other processes that are happening on the server, with an average of *cpuTime* The short 10.84 ms, the addition of this new security system is considered to put no significant additional load on the server.

The processing time obtained also varies, ranging from 98 ms to 555 ms, this result can be affected by *cpuTime* and the number of devices connected to the same Wi-Fi network can cause congestion on the Wi-Fi network. This can increase latency, leading to delays in communication between devices. High latency can affect how quickly the server responds to requests from ESP32, which affects the overall runtime. The process time of not 1 second is considered fast enough and will not cause a queue of students who want to tap KTM.

From the results of the test, it was also obtained that the memory usage was stable at 960 bytes, based on the calculation using equation (3), in a system that has a *Random Access Memory* of 16 GB, or if converted to bytes of 17,179,869,184 bytes, the memory used from the system is obtained of 0.0000056% where this percentage is very small and has no effect on the computational load.

CONCLUSION

Based on the results of the research that has been conducted, it can be concluded that the implementation of the Student Identity Card (KTM) security system based on the 128-bit Advanced Encryption Standard (AES) and rolling code is able to increase protection against duplication attempts, both through simulations using applications such as Card Emulator and through making physical copies of the cards. This implementation demonstrates effectiveness in preventing unauthorized access, especially in test scenarios involving the use of duplicate applications and cards. In addition, the test results show that the implementation of AES encryption and rolling code on the KTM does not incur a significant computational load, with the maximum recorded processing time of 555 ms and very low memory usage, which is 0.000056% of the total 16 GB RAM on the server.

For further research, researchers are further advised to explore the use of more complex encryption algorithms or combine multiple algorithms to improve security. In addition, the study of the impact of system implementation on lower-specification hardware could be the next focus. Researchers may also consider integrating additional technologies, such as multi-factor authentication, to add an extra layer of security. Thus, the security system of the Student Identity Card can continue to be improved in the future.

REFERENCES

Alkalah, C. (2016). *Security Analysis of Rolling Code-based Remote Keyless Entry Systems*. *19*(5), 1–23.

Dang, T. N., & Vo, H. M. (2019). Advanced AES algorithm using dynamic key in the internet of things system. *2019 IEEE 4th International Conference on Computer and Communication Systems, ICCCS 2019*, 682–686. https://doi.org/10.1109/CCOMS.2019.8821647

Kamaludin, H., Mahdin, H., & Abawajy, J. H. (2018). Clone tag detection in distributed RFID systems. *PLoS ONE*, *13*(3), 1–22. https://doi.org/10.1371/journal.pone.0193951

NXP Semiconductors. (2014). *MF1S50yyX/V1 MIFARE Classic EV1 1K - Mainstream contactless smart card IC for fast and easy solution development* (Issue September, p. 40).

Rahim, R., Lubis, S., Nurmalini, N., & Dafitri, H. (2019). Data Security on RFID Information Using Word Auto Key Encryption Algorithm. *Journal of Physics: Conference Series*, *1381*(1). https://doi.org/10.1088/1742-6596/1381/1/012042

Simbolon, I. A. R., Gunawan, I., Kirana, I. O., Dewi, R., & Solikhun, S. (2020). Penerapan Algoritma AES 128-Bit dalam Pengamanan Data Kependudukan pada Dinas Dukcapil Kota Pematangsiantar. *Journal of Computer System and Informatics (JoSYC)*, *1*(2), 54–60.

Supriyati, dan, Pengajar Jurusan Teknik Elektro, S., Negeri Semarang Jl Soedarto, P. H., & Tembalang Semarang, S. (2019). Rancang Bangun Pengontrol Panel Listrik Menggunakan Radio Frekuensi Identifikasi (Rfid). *Orbith*, *14*(1), 28–

39.

Tulloh, A. R., Permanasari, Y., Harahap, E., Matematika, P., Matematika, F., Ilmu, D., & Alam, P. (2016). Kriptografi Advanced Encryption Standard (AES) Untuk Penyandian File Dokumen. *Jurnal Matematika UNISBA*, *Vol 2*(1), 1–8.

Widura, M. S., Purwanto, Y., & Nasution, S. M. (2015). Enkripsi Data Pada Kartu Rfid Menggunakan Algoritma Aes-128 Untuk Angkutan Umum Di Kabupaten Bandung Data Encryption on Rfid Using Aes-128 Algorithm for Public. *E-Proceeding of Engineering*, *2*(2), 3857–3863.

Xiao, Q., Gibbons, T., & Lebrun, H. (2008). *RFID Technology, Security Vulnerabilities, and Countermeasures*. *December*.