A Performance Comparison of Advanced Encryption Standard Across Javascript Libraries

M. Rifky I. Bariansyah and 13517081[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13517081@std.stei.itb.ac.id

Abstract— **JavaScript is a very popular multi-paradigm programming language and also one of the core technologies of the world wide web alongside HTML and CSS. When building using JavaScript we need to reinforce secure communication. To achieve this we can implement AES specification as the block cipher standard. Considering JavaScript and AES popularity it would be insightful and valuable for us to compare the performance of AES implementation across JavaScript libraries. In this paper we find that Crypto, the native Node.js module, have the fastest time execution. Although other libraries have a more simple interface which makes them easier to use.**

Keywords—**Javascript, RSA Algorithm, Libraries, Cryptography**

I. Introduction

We live our lives by exchanging information. With information we attach meaning to data. Through it we learn and gain understanding of a concept. Through the advancement of technology, the number of information exchanges is increasing as the time goes by. However, not all information is for everybody. A piece of information is for someone who has the right to it and not anybody else. Rightful access to information is so important, that the way we store and transmit it keeps evolving to make sure only those for whom the information is intended can read it. Cryptography is the study to attempt just that. Since Julius Caesar hid a message for his general in the war front, it has been hundreds of years of attempt put into securing communication. Cryptography protects information for secure communication by transforming their form that unintended recipients or adversaries cannot understand.

One of the remarkable milestones in modern cryptography was the Rijndael algorithm. Rijndael is the brainchild of Vincent Rijmen and Joan Daemen from Belgia, hence the acronym. It is an innovation for encrypting and decrypting a message to provide secure communication. At that time, National Institute of Standards and Technology pick Rijndael algorithm as Advanced Encryption Standard, replacing Data Encryption Standard which is considered not safe from brute-force attacks. People have their believe in this algorithm that even the US National Security Agency authorizes transmission of classified data through AES. Other than that, it is used in a lot of ways including wireless security, SSL/TLS, and file encryption.

In this day and age a lot of information are exchanged through the web. In its development one of the most popular and widely-used programming language is JavaScript. JavaScript is most well-known as the scripting language for web pages, although many of the non-browser environments are also using it, such as Node.js. In its uses, to support development of a program, JavaScript has a set of a collection of functions called a library. Some of those libraries are designed to reinforce secure communication for the program. Considering JavaScript and AES popularity it would be insightful and valuable for us to compare the performance of AES implementation across JavaScript libraries.

II. Literature Review

A. Advanced Encryption Standard

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information [1]. At that time, Data Encryption Standard or DES for short are considered unsafe due to its vulnerability from brute-force attacks. The National Institute of Standards and Technology (NIST) proposed to the Federal Government a new cryptography standard which after will be called Advanced Encryption Standard. To pick the algorithm for this standard NIST holds an international standard competition. Rijndael algorithm comes out as the winner, written by Vincent Rijmen and Joan Daemen from Belgia. Rijndael algorithm supports keys with length from 128 bit to 256 bit with 32 bit step. In the official announcement AES are categorized into "AES-128", "AES-192", and "AES-256" based on key length. Also, with Rijndael we can set the length of block size and key size independently. From Rinaldi Munir presentation about block cipher review (AES) [2] In general the algorithm is as follows:

1. Add round key, perform XOR between initial state (plaintext) with cipher key. This step is also called the initial round.
2. In number of rounds – 1 rounds, do:
    a. Substitute byte using S-box
    b. Shift state array by wrapping
    c. Mix data within each column in state array
    d. Perform XOR between current state and round key
3. In the final round, do:
    a. Substitute byte using S-box

b. Shift state array by wrapping
c. Perform XOR between current state and round key

Or better illustrated in figure 1.

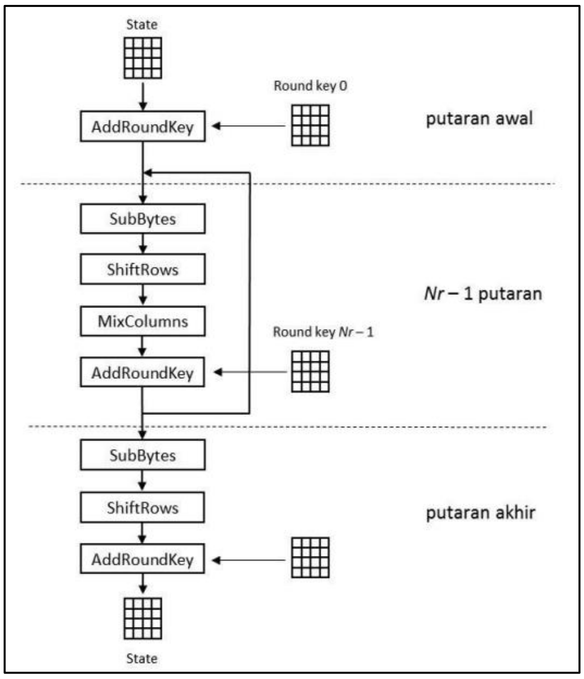

Figure 1 Rijndael Algorithm [2]

B. JavaScript

JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages and JavaScript to specify the behaviour of web pages[3]. JavaScript is a programming language that conforms to the ECMAScript specification, high-level, mostly just-in-time compiled. JavaScript is created by Brendan Eich, an American software technologist who also co-founded the Mozilla Corporation. Every major web browser has a dedicated JavaScript engine to run it. The vast majority of websites use JavaScript for client-side page behavior. This diagram below shows the market position of JavaScript in terms of popularity and traffic compared to the most popular client-side programming languages.

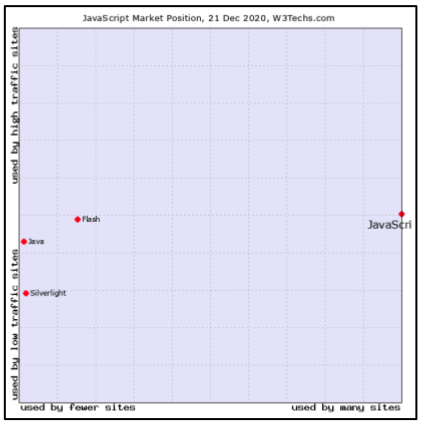

Figure 2 JavaScript Market Position [4]

Examples of scripted behavior that can be implemented using JavaScripts are:

1. Animation of page components
2. Interactive content
3. Transmitting user behavior for analytics
4. Form validation
5. Loading new content without refresh

Nowadays, JavaScript is also a popular language for back-end programming. Node.js for example, is an open-source JavaScript runtime environment that executes JavaScript code outside a web browser. A lot of the time, developers utilize libraries and frameworks when building an application using JavaScript. Some of these libraries are dedicated for secure communication which we will benchmark in the next section. We will compare AES performance from Crypto [5], SJCL[6], AES-JS[7], and CryptoJS[8].

III. BENCHMARK DESIGN

A. Environment

We run this benchmark in a 2017 Macbook Pro with specification below:

Processor	: 2,3 GHz Dual-Core Intel Core i5
Memory	: 8 GB 2133 MHz LPDDR3
OS	: macOS Catalina version 10.15.7

B. Experimentation Design

You can see the full source code by visiting https://github.com/Barbariansyah/js-cryptography-library-benchmark. The experimentation will run in Node.js v15.3.0. We will also need to install CryptoJS (4.0.0), AES-JS (3.1.2), and SJCL (1.0.8). The main components of the experimentation are utilized as follows:

1. Node.js Performance Measurement APIs

   Performance hooks is a Node.js module that provides an implementation of subset of the W3C Web Performance APIs as well as additional APIs for Node.js-specific performance measurements. We will be using perf_hooks.performance which is added in v8.5.0 as an object that collects performance metrics from the current Node.js instance. It is similar to window.performance in browsers. We will also be using perf_hooks.PerformanceObserver, an object that provides notifications when a new performance entry instance has been added to the performance timeline.

   Firstly, we will instantiate a new perf_hooks.PerformanceObserver object. Then we subscribe the object to notifications of the new PerformanceEntry instance, in our case we will pass entryTypes measure as the options. To measure elapsed time for an operation we will be using perf_hooks.performance mark method to mark the start and end of measurement. To present the result as a performance entry, we will use perf_hooks.performance measure method. For clarity, below is an example of how we will implement this design.

```
performance.mark('start');
foo();
performance.mark('end');

performance.measure('entryOne', 'start', 'end');
```

2. CryptoJS

   CryptoJS is an open source collection of standard and secure cryptographic algorithms implemented in JavaScript written by Evan Vosberg. They have a consistent and simple interface. It also claims to be fast and implemented using best practices and patterns. It provides hashing, HMAC, PBKDF2, ciphers, and encoders. For AES cipher, CryptoJS supports AES-128, AES-192, and AES-256 depending on the size of the key we pass in. To use AES, we simply need to pass in the message and the key for encryption. The same goes for decryption. Below is an example of how we can use the library.

```
var encrypted = CryptoJS.AES.encrypt(message, key);
var decrypted = CryptoJS.AES.decrypt(encrypted, key);
```

3. AES-JS

   AES-JS is a pure JavaScript implementation of AES block cipher algorithm and common modes of operation written by Richard Moore. Pure JavaScript means this library has no dependencies. AES-JS also supports AES-128, AES-192, and AES-256. It works on both Node.js and web browsers. There are several modes of operations, each with its pros and cons, to use this library the writer recommended to utilize CBC or CTR (counter). To encrypt in CTR mode, firstly we need to convert the message into bytes. Then we need to create a new ctr modes operation object passing key and a Counter object as parameters. Then we can use the new ctr object to encrypt the message bytes. Lastly, we might want to convert the encrypted bytes to a hex representation. Below is an example of how we can encrypt using CTR mode.

```
var key = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16 ];
var textBytes = aesjs.utils.utf8.toBytes(message);
var aesCtr = new aesjs.ModeOfOperation.ctr(key, new
aesjs.Counter(5));
var encryptedBytes = aesCtr.encrypt(textBytes);
var encryptedHex =
aesjs.utils.hex.fromBytes(encryptedBytes);
```

4. SJCL (Stanford Javascript Crypto Library)

   SJCL is a project by the Stanford Computer Security Lab to build a cross-browser library for cryptography in JavaScript. It claims to be a fast, small, and easy-to-use library. SJCL supports AES-128, AES-192, and AES-256 as well. SJCL is written by Emily Stark, Mike Hamburg, and Dan Boneh. You can see in the example below, the library lives up to its claim of simplicity. It has a similar interface as CryptoJS interface.

```
var encrypted = sjcl.encrypt(key, message)
var decrypted = sjcl.decrypt(key, encrypted)
```

5. Crypto, Node.js Native Module

   Crypto is a Node.js narive module that provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign, and verify functions. It has a stability 2 index which means it prioritizes compatibility with npm ecosystem. To encrypt a message, firstly we need to call createCipheriv to create a Cipher object with the given algorithm, key, and initialization vector. The algorithm parameter is dependent on OpenSSL, in this case we can pass “aes-256-ctr”. To see all available cipher algorithms we can use “openssl list -cipher-algorithms”. Then we can use the new Cipher object to encrypt a message by calling update, passing the message as parameter, and finish by calling the final method. Below is an example of how we can use the library.

```
const encrypt = (text) => {
const cipher = crypto.createCipheriv('aes-256-ctr',
key, iv);
const encrypted = Buffer.concat([cipher.update(text),
cipher.final()]);
  return {
      iv: iv.toString('hex'),
      content: encrypted.toString('hex')
  };
};
```

IV. Experimentation

In this chapter we will present and evaluate the result of experimentation. We will divide the experimentation for each library.

A. CryptoJS Experimentation

Using CryptoJS, the experimentation yields these results for each message size.

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	17.841353	13.852418
2	16.396963	14.069593
3	16.515645	13.952843
4	17.146084	14.114056
5	17.977745	14.042595
Average	17.175558	14.006301

Table 1 CryptoJS Experimentation with 10,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	35.440785	25.186723
2	35.795584	28.45994
3	35.151028	26.831793

4	34.939213	25.552472
5	35.717076	25.185647
Average	35.4087372	26.243315

Table 2 CryptoJS Experimentation with 100,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	95.000625	75.852406
2	89.070994	89.135333
3	99.424576	82.708819
4	98.698973	88.54111
5	89.021984	78.228777
Average	94.2434304	82.893289

Table 3 CryptoJS Experimentation with 500,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	141.628682	112.372665
2	148.263237	111.716496
3	155.338804	113.907543
4	141.73455	103.623569
5	140.762151	112.672829
Average	145.5454848	110.8586204

Table 4 CryptoJS Experimentation with 1,000,000 byte-sized message

Here we can observe that encryption takes longer than decryption in every size variation. Although, the time difference between is not significant, ranging from 12 – 25% faster in decryption, with the average of 20%.

B. SJCL (Stanford JavaScript Crypto Library) Experimentation

Using SJCL, the experimentation yields these results for each message size.

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	75.641918	9.497107
2	84.286674	9.679138
3	73.819804	9.089023
4	78.747621	9.456019
5	79.120135	9.252784
Average	78.3232304	9.3948142

Table 5 SJCL Experimentation with 10,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	118.25868	59.13565
2	109.439826	56.286141
3	115.700833	54.824758
4	117.45994	56.030169
5	123.567633	50.173876
Average	116.8853824	55.2901188

Table 6 SJCL Experimentation with 100,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	304.161079	199.06949
2	364.307798	229.534752
3	351.409116	225.021283
4	285.920648	190.658386
5	350.458661	220.828816
Average	331.2514604	213.0225454

Table 7 SJCL Experimentation with 500,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	652.073099	430.906642
2	583.915301	410.622848
3	623.635585	412.520444
4	589.865841	418.501603
5	648.900639	464.372405
Average	619.678093	427.3847884

Table 8 SJCL Experimentation with 1,000,000 byte-sized message

Here we can observe that encryption also takes longer than decryption in every size variation. With this library the time difference between is quite large in smaller message, in this case with the input of 50,000 byte-sized message, it is 88% faster in decryption. In larger messages, it is ranging from 31 – 52% faster in decryption, with the average of 39%.

C. AES-JS Experimentation

Using AES-JS, the experimentation yields these results for each message size.

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	30.052366	7.326593
2	26.488317	7.174984
3	25.551154	7.882844
4	28.011255	8.478245
5	27.937679	8.241946
Average	27.6081542	7.8209224

Table 9 AES-JS Experimentation with 10,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	101.8479	44.801735
2	101.462085	45.430548
3	101.970406	41.799856
4	91.421935	38.576278
5	99.858574	47.004053
Average	99.31218	43.522494

Table 10 AES-JS Experimentation with 100,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	355.048412	147.901876
2	322.731794	144.807294
3	311.07868	137.132927
4	323.040695	154.003107
5	346.979898	165.656987
Average	331.7758958	149.9004382

Table 11 AES-JS Experimentation with 500,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	650.179047	329.7478
2	633.404643	288.69983
3	600.809919	289.285038
4	605.90802	284.774002
5	671.573227	332.880071
Average	632.3749712	305.0773482

Table 12 AES-JS Experimentation with 1,000,000 byte-sized message

Here we can observe that encryption also takes longer than decryption in every size variation. With this library the time difference between is quite large in smaller messages, in this case with the input of 50,000 byte-sized messages, it is 71% faster in decryption. In larger messages, it is ranging from 51 – 56% faster in decryption, with the average of 54%.

D. Crypto, Node.js Native Module Experimentation

Using Crypto, the experimentation yields these results for each message size.

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	0.736905	0.195224
2	0.801789	0.199657
3	0.604672	0.20647
4	0.608925	0.207295
5	0.607982	0.21767
Average	0.6720546	0.2052632

Table 13 Crypto Experimentation with 10,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	0.989016	1.258056
2	0.898075	1.436051
3	1.318075	1.330079
4	1.328486	1.31401
5	1.056794	1.279425
Average	1.1180892	1.3235242

Table 14 Crypto Experimentation with 100,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	2.517895	4.221895
2	2.880784	4.835602
3	3.06908	3.820867
4	2.723011	3.746837
5	2.669575	4.114572
Average	2.772069	4.1479546

Table 15 Crypto Experimentation with 500,000 byte-sized message

Experiment No.	Elapsed Time in Millisecond	
	Encryption	Decryption
1	4.871497	7.979456
2	4.770808	7.559927
3	4.60618	8.267171
4	4.297014	8.303212
5	5.276317	7.78932
Average	4.7643632	7.9798172

Table 16 Crypto Experimentation with 1,000,000 byte-sized message

Here we can observe that encryption takes longer than decryption in small message while less in larger messages. We can also see that this library is vastly faster than the others.

D. Elapsed Time Comparison

In graph representation we can see the data from the native Node.js module, Crypto, is in a different range.

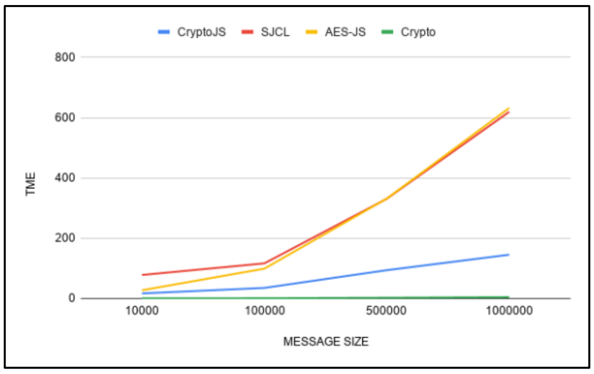

Figure 3 Message Size vs Time (Encryption)

For clarity we will zoom in on the data from Crypto.

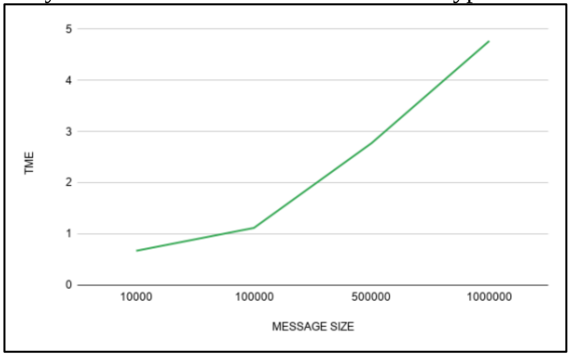

Figure 4 Message Size vs Time (Encryption with Crypto)

For encryption, we can see that Crypto has the fastest execution time, followed by CryptoJS, AES-JS, and SJCL up to 500,000 byte-sized messages. In 1,000,000 byte-sized messages we can see that AES-JS is slightly faster than CryptoJS. In Figure 3, we find that the similar range difference also occurs on decryption.

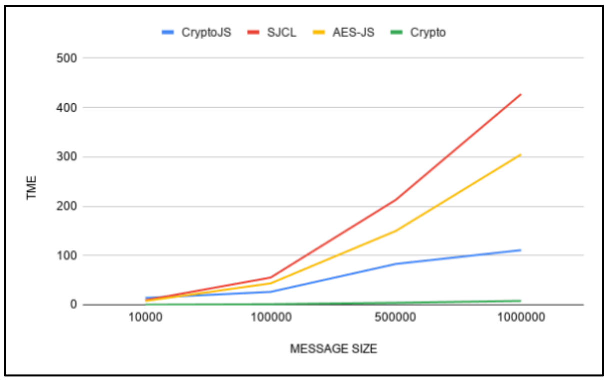

Figure 5 Message Size vs Time (Decryption)

For clarity we will also zoom in on the data from Crypto.

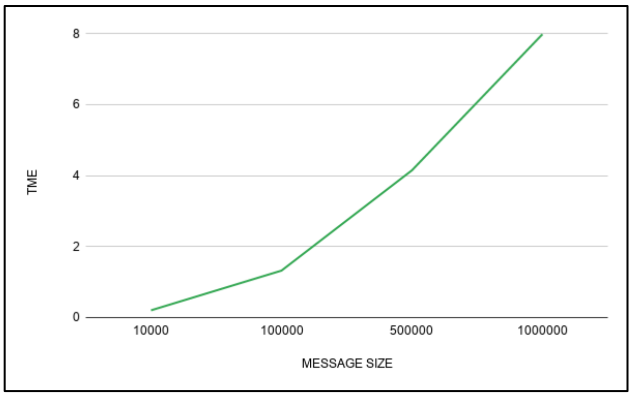

Figure 6 Message Size vs Time (Decryption with Crypto)

For decryption we can see that Crypto also has the fastest execution time, followed by CryptoJS, AES-JS, and SJCL in large messages. In 10,000 byte-sized messages, decryption is faster in AES-JS than in Crypto.

V. CONCLUSION

From the experimentation result across libraries, we can observe that Crypto, the native Node.js module, yields the fastest execution time both in encryption and decryption followed by CryptoJS, AES-JS, and SJCL. Although, from the experimentation design we can see that CryptoJS, AES-JS, and SJCL have a more straightforward interface than Crypto. Another important finding is that between the second and third fastest library in this experimentation, CryptoJS and AES-JS, CryptoJS is faster in encryption while in a certain size range AES-JS is faster in decryption. These three key discoveries can be valuable in the decision making process of choosing a library for implementing AES in JavaScript.

VI. ACKNOWLEDGMENT

Thank God for his blessings I am able to finish this paper with great studiousness. I would like to thank our family and friends for supporting us in the making of this paper. I would also like to thank Dr. Ir. Rinaldi Munir for his guidance and the inspiration all his students receive through his teaching. May this paper be beneficial and insightful for the reader.

REFERENCES

[1] Standard, N. F. (2001). Announcing the advanced encryption standard (aes). Federal Information Processing Standards Publication, 197(1-51), 3-3.
[2] Munir, R. (2020). Review Beberapa Block Cipher dan Stream Cipher (Bagian 4: Advanced Encryption Standard (AES))
[3] Flanagan, D., & Like, W. S. (2006). JavaScript: The Definitive Guide, 5th.
[4] "Usage statistics of JavaScript as client-side programming language on websites". *w3techs.com*.
[5] OpenJS Foundation. Crypto. Retrieved from https://nodejs.org/api/crypto.html
[6] Stark, E., Hamburg, M., & Boneh, D. (2009, December). Symmetric cryptography in javascript. In 2009 Annual Computer Security Applications Conference (pp. 373-381). IEEE.
[7] Moore, R. (2018). AES-JS. Retrieved from https://github.com/ricmoo/aes-js
[8] Vosberg, E. (2020). CryptoJS. Retrieved from https://github.com/ricmoo/aes-js

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 21 Desember 2020

[illegible]

M. Rifky I. Bariansyah
13517081