Investigating the efficiency of AES and RSA Encryption algorithms in terms of the memory used and time taken to encrypt/decrypt alphanumeric data.

Research question: How is AES symmetric encryption algorithm more efficient in regards to the speed and memory used compared to the RSA asymmetric algorithm when encrypting /decrypting alphanumeric data?

Subject: Computer Science

Word count: 3855

CS EE World
https://cseeworld.wixsite.com/home
May 2020
24/34
B
Submitter Info: Anonymous

Table of Contents

Introduction

Over the past decade the Internet has become a major aspect of human connection. Major tasks done by MNCs, governments and individuals are done on the internet. However, this may leave the people exposed to hackers who may gain access to confidential information that companies/governments may not want to show others. This is where encryption comes in.

Encryption is the process of converting messages, data or information into an unreadable format by anyone except the intended recipient. This is called encrypted data and can be decrypted only using a secret key (decryption key) and the recipient has the key which can decipher the encrypted data[1]. The encrypted data is normally referred to as ciphertext and decrypted data is referred to as plaintext[2].

There are mainly two types of encryption algorithms:

1. Symmetric algorithms: Algorithms where only one key is used to encrypt and decrypt the electronic information. The sender must share the key with the recipient so that the recipient can decrypt the data. Once the key is used by the recipient, the algorithm reverses the action done to encrypt the data and the message becomes readable again.

[1] Jackob, Melis. "History of Encryption." Web. 28 Aug. 2019 <https://www.sans.org/reading-room/whitepapers/vpns/history-encryption-730>

[2] Lord, Nate. "What Is Data Encryption? Definition, Best Practices & More." *Data Insider*, Digital Guardian, 15 July 2019, Web. 27 Aug. 2019 <https://digitalguardian.com/blog/what-data-encryption>

The code used by the sender can either be a string of letters and numbers or numbers generated by a random number generator[3]. Some examples are AES, Blowfish, DES.

2. Asymmetric algorithms: These algorithms use two keys to encrypt a plain text. The secret keys are exchanged on the internet or over a Large Area Network. This is known as the public key. The other key isn't available on the internet and is only with the sender and the receiver in order to boost security[4]. This is known as the private key. Either of the keys can be used to encrypt the message. The other one is then used to decrypt the message. These keys aren't identical, hence the name asymmetric. Many protocols like SSL, OpenPGP, SSL/TLS rely on asymmetric encryption algorithms for encryption of data and digital signature functions[5]. Because of the two keys, these algorithms are generally considered to be more secure than the symmetric encryption algorithms. Some examples are RSA (Rivest–Shamir– Adleman), DSA (Digital Signature Algorithm) and ECC (Elliptic curve cryptography).

[3] Smirnoff, Peter, and Dawn M Turner. "Symmetric Key Encryption - Why, Where and How It's Used in Banking." *Cryptomathic*, Cryptomathic, 18 Jan. 2019, Web. 27 Aug. 2019 <https://www.cryptomathic.com/news-events/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking>

[4] Publishers. "Symmetric vs. Asymmetric Encryption – What Are Differences?" *Global SSL Provider*, SSL2BUY, 7 Feb. 2019, Web. 27 Aug. 2019 <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>

[5] Rouse, Margaret. "What Is Asymmetric Cryptography? - Definition from WhatIs.com." *SearchSecurity*, TechTarget, July 2019, Web 27 Aug. 2019 <https://searchsecurity.techtarget.com/definition/asymmetric-cryptography>

Block ciphers and Stream ciphers

A cipher is the algorithm that performs encryption[6]. The two types of ciphers are:

1. Block ciphers: This method divides the data into blocks which is then encrypted to produce blocks of ciphertext. AES and RSA are examples of block ciphers.
2. Stream ciphers: This method takes in a stream of data and operates on it bit by bit. It consists of two components: a cryptographic key and an algorithm. Examples are RC4, RC2 and RC5.

   This method is not used much nowadays because it's alternative, block cipher acts on blocks of data instead of bits.

Research Question

How is AES symmetric encryption algorithm more efficient in regards to the speed and memory used compared to the RSA asymmetric algorithm when encrypting /decrypting alphanumeric data?

[6] "What Is a Cipher? - Definition from Techopedia." *Techopedia.com*, Techopedia, Web. 28 Aug. 2019 <https://www.techopedia.com/definition/6472/cipher>

AES Encryption Algorithm

AES encryption algorithm is a symmetric block cipher chosen by the US government[7] and is used in software and hardware devices throughout the world to encrypt sensitive data[8]. The National Institute of Standards and Technology(NIST) started the development for the AES algorithm in 1997 when its predecessor, DES started becoming vulnerable to brute force attacks. A brute force attack is a cryptographic hack which relies on guessing the password till the correct password is found[9].

The AES being a block cipher is capable of 128 bit blocks with a key size of either 128, 192 or 256 bits(192 and 256 used only for heavy duty encryption purposes). This is the only publicly available software which is approved by the National Security Agency to protect government information at the highest levels of security clearance and can only be vulnerable to very large brute force attacks.

---

[7] DeMuro, Jonas. "What Is AES?" *TechRadar*, TechRadar Pro, 29 Oct. 2018, Web. 3 Sept. 2019 <https://www.techradar.com/in/news/what-is-aes>

[8] Rouse, Margaret. "What Is Advanced Encryption Standard (AES)? - Definition from WhatIs.com." *SearchSecurity*, Mar. 2017, Web. 1 Dec. 2019 <https://searchsecurity.techtarget.com/definition/asymmetric-cryptography>

[9] "What Is a Brute Force Attack?" *Forcepoint*, Forcepoint, 30 Oct. 2019, Web. 1 Dec. 2019 <https://www.forcepoint.com/cyber-edu/brute-force-attack>

The structure of the AES Encryption algorithm

Parameters for working(for 128 bits plaintext)

-Block size: 128 bits plaintext(4 words/16 bytes)

-Number of rounds: 10 rounds

-Key size:128 bits

-Number of subkeys: 44

-Single subkey size: 32 bits

-Subkeys used in each round: 4

-Subkeys used in pre-round calculation: 4

-Resultant ciphertext: 128 bits

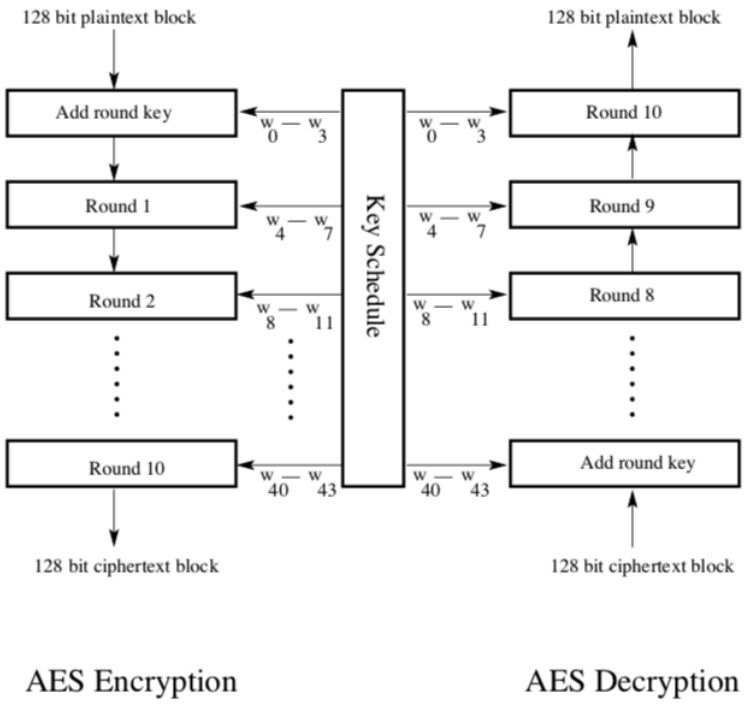


Figure 1: AES Encryption and Decryption process

AES encryption follows the structure shown on the left. When the size of the key is 128 bits, the number of rounds is 10. Similarly, when the key size is 192 bits, the number of rounds is 12 and 14 when the key size is 256. These are found using the following formula:

$$Number\ of\ rounds = \frac{key\ size}{32} + 6$$

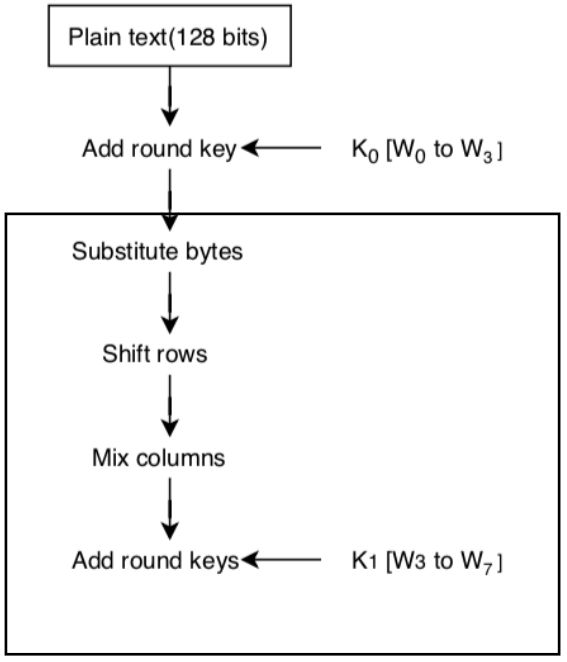
Before the encryption process begins, the input array is XORed with the first four words of the key schedule. The same happens during the decryption process, except the ciphertext state array is XORed with the last four words of the Rjindael's key schedule.

The AES algorithm breaks data into 4 x 4 tables which are referred to as state arrays. In a cipher with a 128-bit key, a two dimensional array with 4 rows and 4 columns is formed where each input in the array is one byte. Therefore, there are 16 bytes in total. This can be represented by the diagram:

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

There are four steps to each round in AES encryption:

1. Substitute bytes
2. Shift rows
3. Mix columns
4. Add round key



For decryption, each round consists of the following four steps:

1. Inverse add round key
2. Inverse mix columns
3. Inverse shift rows
4. Inverse byte substitution

$s_{(0,0)}$	$s_{(0,1)}$	$s_{(0,2)}$	$s_{(0,3)}$
$s_{(1,0)}$	$s_{(1,1)}$	$s_{(1,2)}$	$s_{(1,3)}$
$s_{(2,0)}$	$s_{(2,1)}$	$s_{(2,2)}$	$s_{(2,3)}$
$s_{(3,0)}$	$s_{(3,1)}$	$s_{(3,2)}$	$s_{(3,3)}$

The input value is stored in a two dimensional array, a 4x4 table which looks like the table shown on the left:

Each value has a size of 8 bits, therefore,

$$16 \times 8 = 128\ bits$$

Which is the size of the plaintext input.

The output array is exactly the same.

In both encryption and decryption, during the add round key step, the output of the previous step (three for encryption, two for decryption) is XORed with four words from the key schedule. The last round for both encryption and decryption does not involve the Mix columns step.

The four steps in each round of processing:

1. Substitute bytes[10]: In this step, byte-by-byte substitution occurs using a rule that is the same for all encryption rounds. For the decryption process, the rule will change, but it will remain the same for all the ten rounds (provided that the key size is 128 bits).

There are two ways of performing the byte substitution process. They are:

- The modern way: Java uses this way of finding a substitute byte. In this step, a given byte is substituted by a different byte using a pre-computed 256-element array.
- The traditional way: In this step, a 16x16 lookup table is used in order to find the substitute bytes.

The goal of this step is to reduce the correlation between the input and the output bits at the *byte* level and it is done in such a way such that it cannot be described by a mathematical function.

2. Shift Rows: The following circular transformations in the state array take place in this step during encryption:
   - First row does not shift at all
   - The second row shifts by one byte to the left
   - The third row shifts by two bytes to the left
   - The last row shifts by three bytes to the left

[10] Kak, A. "Lecture 8: AES: The Advanced Encryption Standard Lecture Notes on 'Computer and Network Security.'" *Engineering*, Purdue University, 31 Jan. 2019, Web. 1 Dec. 2019 <https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf>

These steps can be represented by the following diagram:

$$\begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} ===> \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2.2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3.3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

For decryption, the steps take place in opposite order. The first row remains unchanged, the second row is shifted to the right by one byte, the third row by two bytes to the right and the last row by three bytes to the right, all shifts being circular. This can be shown by the following diagram:

$$\begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} ===> \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.3} & s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2.2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3.1} & s_{3,2} & s_{3,3} & s_{3,0} \end{bmatrix}$$

3. Mix Columns: This step replaces each byte of a column by a function of all the bytes in the same column.

Each byte in a column is replaced by two times that byte, plus three times the next byte, plus the byte that comes next, plus the byte that follows.

The operations in each column can be shown by the following diagram:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0.0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1.0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2.0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3.0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

4. Add Round Key: This is the most important stage in the encryption algorithm as it provides uniqueness to the encryption. Due to this stage, it becomes a complex operation to decrypt. The values of the array after this stage depend on the subkey (same size as the state array) is computed using Rjindael's Key Schedule. Once a subkey is generated, the following steps are applied to the state array which results in the sum of the state and subkey being obtained:
   - Rotate: This step is to rotate the bytes that form the word 1 byte to the left. This step is similar to the second step of the Shift Rows step.
   - Rcon: Name of a sub-operation applied to the state array after the rotate step.
   - Key expansion: This step expands the main key to the required number of keys. However, due to the complexity of this process, it won't be explained in this paper.

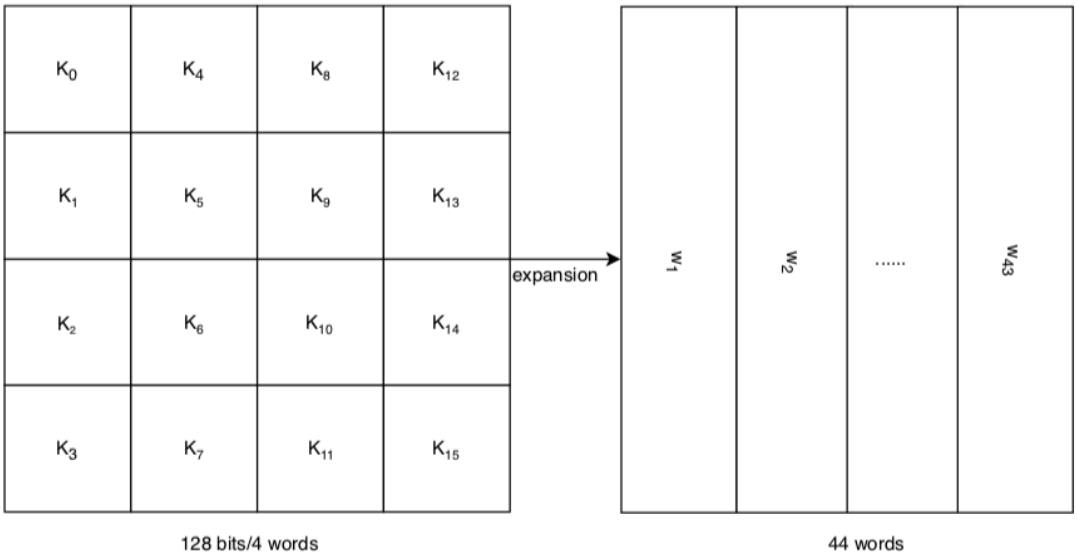


Figure 2: Key Expansion process where 4 words/128 bits are expanded to 44 words

To better understand the process, we can take the help of an example[11]:

- Let's say that the string we want to encrypt is "Thats my Kung Fu".
- To perform the steps of encryption, first, we have to convert the text into ASCII characters:

$$\begin{vmatrix} T & s & & g \\ h & & K & \\ a & m & u & F \\ t & y & n & u \end{vmatrix} \text{->} \begin{vmatrix} 54 & 73 & 20 & 67 \\ 68 & 20 & 4B & 20 \\ 61 & 6D & 75 & 46 \\ 74 & 79 & 6E & 75 \end{vmatrix}$$

- Let's say that our key for the round is 'Two One Nine Two'. This translated to hexadecimal becomes:

$$\begin{vmatrix} T & O & N & \\ w & n & i & T \\ o & e & n & w \\ & & e & o \end{vmatrix} \rightarrow \begin{vmatrix} 54 & 4F & 4E & 20 \\ 77 & 6E & 69 & 54 \\ 6F & 65 & 6E & 77 \\ 20 & 20 & 65 & 6F \end{vmatrix}$$

- Now, the state array is XORed with the round keys, for example 69 ⊕ 4B becomes 22:

$$\begin{array}{r} 0110\ 1001 \\ 0100\ 1011 \\ \hline 0010\ 0010 \end{array}$$

- After the XOR process, the new matrix becomes:

$$\begin{vmatrix} 00 & 3C & 6E & 47 \\ 1F & 4E & 22 & 74 \\ 0E & 08 & 1B & 31 \\ 54 & 59 & 0B & 1A \end{vmatrix}$$

- After performing the substitute bytes step using the traditional method, the matrix becomes:

[11] "AES Example - Input (128 Bit Key and Message)." *AES Example*. Kavaliro, Web. 2 January 2020 < https://kavaliro.com/wp-content/uploads/2014/03/AES.pdf>

$$\begin{vmatrix} 63 & EB & 9F & A0 \\ C0 & 2F & 93 & 92 \\ AB & 30 & AF & C7 \\ 20 & CB & 2B & A2 \end{vmatrix}$$

- Performing the shift rows step:

$$\begin{vmatrix} 63 & EB & 9F & A0 \\ 2F & 93 & 92 & C0 \\ AF & C7 & AB & 30 \\ A2 & 20 & CB & 2B \end{vmatrix}$$

- Performing the mix columns step:

$$\begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \begin{vmatrix} 63 & EB & 9F & A0 \\ 2F & 93 & 92 & C0 \\ AF & C7 & AB & 30 \\ A2 & 20 & CB & 2B \end{vmatrix} \rightarrow \begin{vmatrix} BA & 84 & E8 & 1B \\ 75 & A4 & 8D & 40 \\ F4 & 8D & 06 & 7D \\ 7A & 32 & 0E & 5D \end{vmatrix}$$

- The array after the add round key step and the first round of encryption becomes:

$$\begin{vmatrix} 58 & 15 & 59 & CD \\ 47 & B6 & D4 & 39 \\ 08 & 1C & E2 & DF \\ 8B & BA & E8 & CE \end{vmatrix}$$

The steps shown above happen ten times, after which the final encrypted array is obtained.

RSA Encryption Algorithm

The RSA Algorithm is the most famous asymmetric encryption algorithm[12]. The name RSA comes from its founders: Ron Rivest, Ali Shamir and Leonard Adelman. It is a public key algorithm and is considered to be the standard for encrypting data over the internet. Being an asymmetric algorithm, it has all the advantages and disadvantages that come with asymmetric algorithms.

The working of the RSA algorithm is as follows:

Being an asymmetric algorithm, the algorithm has a public key and a private key. They are calculated using the following steps:

1. Consider two large prime numbers: *p* and *q.* They have to be large in order to make the ciphertext secure and not be vulnerable to brute force attacks.
2. Calculate $n = p * q$
3. Calculate Euler's totient function for *n*. Euler's totient function, is defined as the number of positive integers less than or equal to *n* that are coprime to (i.e., do not contain any factor in common with) *n*, where 1 is counted as being coprime to all numbers. Since a number less than or equal to and coprime to a given number is called a totative, the totient function $\phi(n)$ can be simply defined as the number of totatives of *n*[13]. For example, $\phi(1)$ =1. Some of the values that the function returns for the first few numbers are:

[12] "The Mathematical Algorithms of Asymmetric Cryptography and an Introduction to Public Key Infrastructure." *Infosec Resources*, 7 Feb. 2017, Web. 9 Sept. 2019 < https://resources.infosecinstitute.com/mathematical-algorithms-asymmetric-cryptography-introduction-public-key-infrastructure/#gref >

[13] Weisstein, Eric W. "Totient Function." *From Wolfram MathWorld*, Wolframalpha, Web. 1 Dec. 2019 <http://mathworld.wolfram.com/TotientFunction.html>

n	$\phi(n)$	numbers coprime to n
1	1	1
2	1	1
3	2	1, 2
4	2	1,3
5	4	1,2,3,4
6	2	1,5
7	6	1,2,3,4,5,6
8	4	1,3,5,7
9	6	1,2,4,5,7,8
10	4	1,3,7,9

4. Assume *e* to be the encryption key. It is calculated by using a number *x* such that the highest common factor of *x* and $\phi(n)$ is 1.
5. Assuming *d* to be the decryption key. It is calculated such that $(d * e)\ mod\ \phi(n) = 1$.
6. Now, the public key and private key can be formed:

   Public key: {*e*, *n*}

   Private key: {*d*, *n*}

The encryption process: The condition before starting the encryption process is that *M*<*n* where *M* is the plaintext. Then, the ciphertext is formed using the formula:

$C = M^e \ mod \ n$ where C is the ciphertext.

Here, it must be understood that *M* can only take numeric values. In order to take worded messages like, “hello world”, each alphabet’s ASCII values are taken. This can be represented like:

The plaintext would be 48 65 6C 6F 20 57 6F 72 6C 64 (hexadecimal). Some junk values are added to the start and the end which makes it harder for hackers to decrypt the data. This process is called padding due to which the plaintext would look something like not at all like the plaintext which is then used as the plaintext and the formula can be applied to form the ciphertext.

The decryption process: The plaintext can be calculated using the formula:

$M = C^d \ mod \ n$ where *M* is the plaintext.

Let’s perform a simple RSA Encryption process using small numbers for understanding purpose. However, in proper encryption processes, the numbers have to be large in order to have good security.

1. Let *p*=3 and *q*=5
2. $n = p * q = 3 * 5 = 15$
3. $\phi(n) = (3 - 1)(5 - 1) = 2 * 4 = 8$
4. *e* is a number such that *e* and $\phi(n)$’s HCF is 1. Therefore, assuming *e* to be 3, the HCF of 4 and 8 is 1, therefore the value of *e* is 3.
5. Calculating *d*:

   $(d * e) \ mod \ \phi(n) = 1.$

   $(d * 3) \ mod \ 8 = 1$

If *d* is assumed to be 3, then the equation is true. Hence, the value of *d* is 3.

6. Now, the public key is {3,15} and the private key is also {3,15}.
7. *Encryption*: The condition that *M*<*n* has to be satisfied. Therefore, *M* has to be less than 15. Let's assume the plaintext to be 10.

   $C = M^e \ mod \ n = 10^3 \ mod \ 15 = 1000 \ mod \ 15 = 10$. Hence, the value of *e* is 10. Since, this the prime numbers taken are small, the value is the same but if we take large prime numbers and perform the padding process, the value will change.
8. *Decryption*:

$M = C^d \ mod \ n = 10^3 mod \ 10 = 10$. Therefore, the same plaintext is obtained.

Experiment

Comparing the efficiency of AES and RSA encryption algorithms with respect to speed and memory used.

Methodology

Primary experimentation will provide a majority of the data used in this extended essay.

To compare the **speed** of the two algorithms, Java programs have been written[14] where there is a time function (found in .util package) which will allow me to get the time taken to encrypt a given string in nanoseconds. For executing the Java programs, NetBeans IDE, version 8.2 will be used to run the values and get the time taken to encrypt and decrypt the values. To get the time taken by each program, I will use the nanoTime() function provided by Java.

[14] Refer to appendix for code

To compare the **memory** used by the two algorithms, the program used to compare the speed will be used. To get the memory used by the encryption and decryption process, the totalMemory() and freeMemory() function will be used (found in .util package) and subtract the two values before and after the functions to encrypt and decrypt the values are called. This will provide an estimated value as the value will count in the function calling and other processes as well, but will provide an estimate for the comparison of the two algorithms.

Due to the presence of various background processes on the computer, a fixed value for the time taken or the memory used won't be possible and hence, the average of five trials will be taken.

As the research question states, this extended essay will compare the two parameters using alphanumeric data. Therefore, the experiment will be conducted 3 times: first being only using alphabets, second only using numbers and third using both in a mixed string. The three strings are as follows:

1. Alphabets: "buy me a cake"
2. Numbers: "1902319181"
3. Alphanumeric: "430 Wood Street, 47906, Jakarta"

Hypothesis

My hypothesis is that AES will be faster that RSA to encrypt and decrypt the values and wouldn't use as much memory and thus is good enough to protect it from hackers, but wouldn't be good enough against large scale brute-force attacks.

Results

1. **String with only alphabets ("buy me a cake"):**

Time taken:

	AES (in nanoseconds)	**RSA (in nanoseconds)**
Trial 1	336	235
Trial 2	201	295
Trial 3	320	296
Trial 4	115	322
Trial 5	256	547

Data calculation:

Calculating the mean value of AES timings: $\frac{336+201+320+115+256}{5} = 245.6\ nanoseconds$

Calculating the mean value of RSA timings: $\frac{235+295+296+322+547}{5} = 339\ nanoseconds$

Memory used:

	AES (in KB)	RSA (in KB)
Trial 1	13.355	27.49188
Trial 2	13.357792	31.90023
Trial 3	13.412088	28.93351
Trial 4	13.397936	29.760448
Trial 5	13.364272	30.438008

Data Calculation:

Calculating the mean value of the memory used by AES:

$$\frac{13.357792+13.412088+13.397936+13.355+13.364272}{5} = 13.3774176\ KB$$

Calculating the mean value of the memory used by RSA:

$$\frac{27.49188+31.90023+28.93351+29.760448+30.438008}{5} = 29.7048152\ KB$$

2. String with only numerical values ("1902319181"):

Time taken:

	AES (in nanoseconds)	RSA (in nanoseconds)
Trial 1	163	292
Trial 2	189	265
Trial 3	122	339

Trial 4	217	316
Trial 5	195	277

Data calculation:

Calculating the mean values of the AES timings: $\frac{163+189+122+217+195}{5} = 177.2\ nanoseconds$

Calculating the mean values of the RSA timings: $\frac{292+265+339+316+316+277}{5}\ 297.8\ nanoseconds$

Memory used:

	AES (in KB)	**RSA (in KB)**
Trial 1	13.411856	29.095424
Trial 2	13.389632	30.102696
Trial 3	13.328792	31.096128
Trial 4	13.35568	32.75972
Trial 5	13.355784	28.431168

Data Calculation:

Calculating the mean value of the memory used by AES:

$$\frac{13.411856+13.389632+13.328792+13.35568+13.355784}{5} = 13.3683488\ KB$$

Calculating the mean value of the memory used by RSA:

$$\frac{29.095424+30.102696+31.096128+32.75972+28.431168}{5} = 30.3170819\ KB$$

3. **String with alphanumeric values ("430 Wood Street, 47906, Jakarta")**

Time taken:

	AES (in nanoseconds)	**RSA (in nanoseconds)**
Trial 1	238	480
Trial 2	166	233
Trial 3	155	360
Trial 4	132	280
Trial 5	145	345

Data calculation:

Calculating the mean values of the AES timings: $\frac{238+166+155+132+145}{5} = 167.2\ nanoseconds$

Calculating the mean values of the RSA timings: $\frac{480+233+360+280+345}{5} = 339.6\ nanoseconds$

Memory used:

	AES (in KB)	**RSA (in KB)**
Trial 1	13.355584	32.102376
Trial 2	13.355616	28.423992
Trial 3	13.36424	29.430352
Trial 4	13.345008	30.432024
Trial 5	13.3555752	31.5653344

Data calculation:

Calculating the mean value of the memory used by AES:

$$\frac{13.355584+13.355616+13.36424+13.345008+13.3555752}{5} = 13.3552046\ KB$$

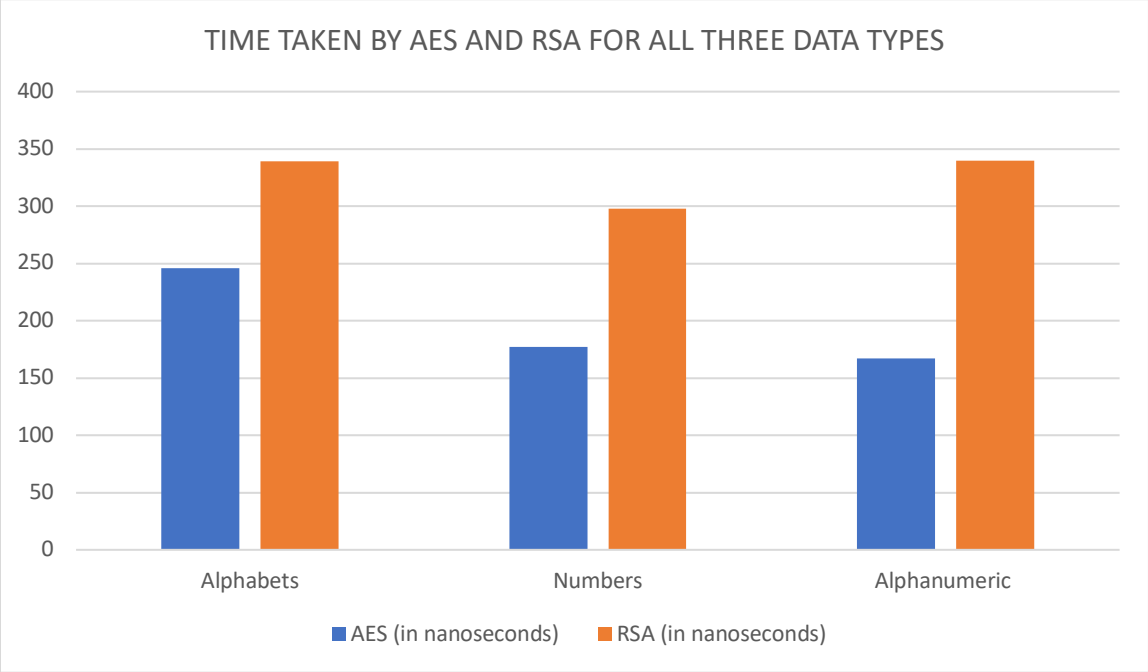
Calculating the mean value of the memory used by AES:

$$\frac{32.102376+28.423992+29.430352+30.432024+31.5653344}{5} = 30.3908168\ KB$$

Analysis

Time taken:

	AES (in nanoseconds)	RSA (in nanoseconds)
Alphabets	245.6	339
Numbers	177.2	297.8
Alphanumeric	167.2	339.8



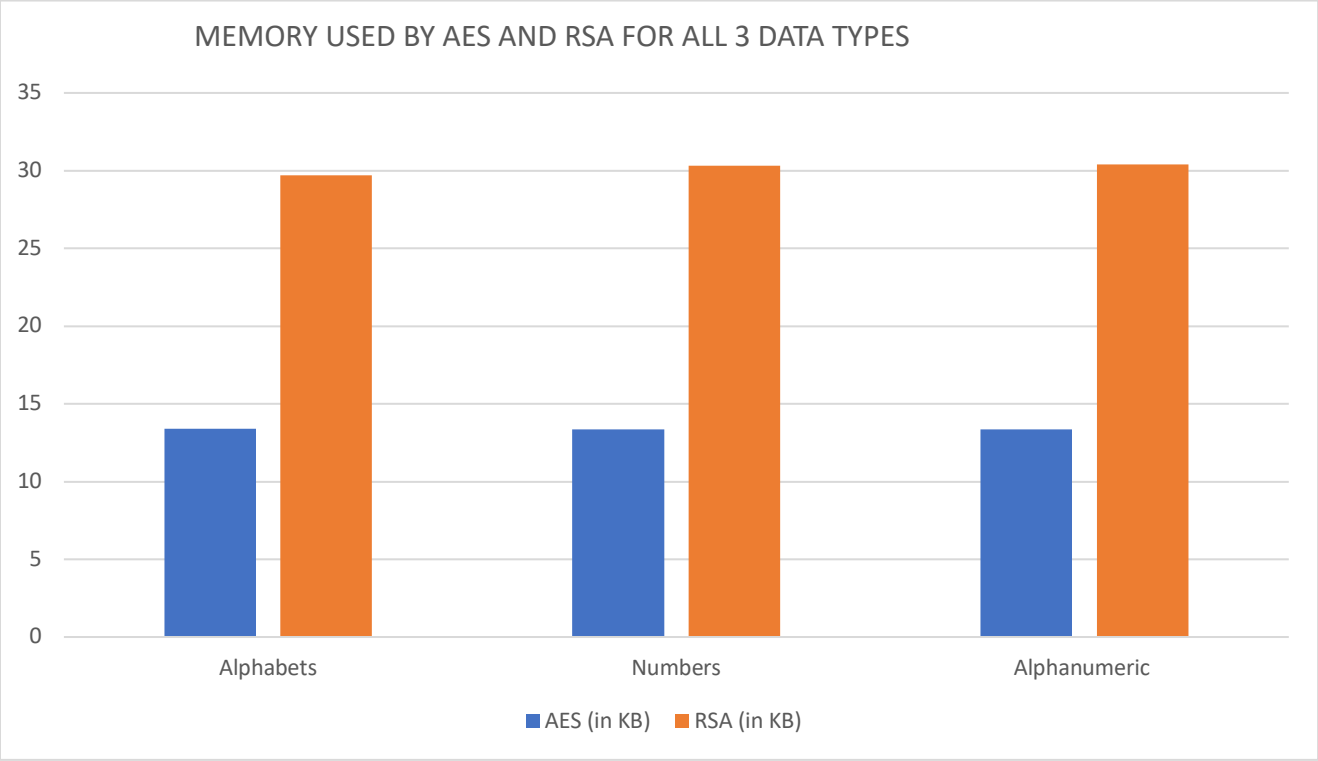
Graph 1: Timings for AES and RSA

As it can be seen from the graph, the AES algorithm is consistently faster than the RSA algorithm for all three types of data, being around 172 nanoseconds faster for alphanumeric data, around 94 nanoseconds faster for alphabetical data and approx. 121 milliseconds faster for numerical data. Keeping in mind that only a small string for all the types of data was used while comparing the two algorithms, the difference between the two algorithms would be more pronounced

when comparing the two algorithms using large datasets which would be the situation in real life scenarios. The hypothesis therefore holds true, with AES being faster than RSA by quite a significant amount.

Memory used:

	AES (in KB)	**RSA (in KB)**
Alphabets	13.3774176	29.7048152
Numbers	13.3683488	30.3170819
Alphanumeric	13.3552046	30.3908168



Graph 2: Memory used by AES and RSA

As it can be seen from graph 2, AES takes up less than half the memory used by RSA with it using 13 KB compared to the 30 KB used by RSA. Again, keeping in mind that only a small string was

used for all the different types of data while comparing the algorithms, the difference would be more pronounced in real life scenarios. Therefore, the hypothesis holds true here as well, with AES being more efficient than RSA by quite a margin.

Evaluation

The programs[15] were appropriate methods of evaluating:

1. The memory used by the two algorithms to encrypt and decrypt the three different types of data. The higher the memory usage, the more the memory required to perform the task, hence the requirements of a system will be higher, increasing the costs. Therefore, AES is more advisable in real life scenarios due to its lesser memory usage.
2. The time taken by each algorithm to encrypt and decrypt the different types of data. A lesser encryption/decryption time results in the system being faster and more responsive. data. The programs provided the time taken in nanoseconds quite accurately and as hypothesized, AES was quicker than RSA because of various factors such as the number of operations required to be done by each algorithm, the size of the key used and the type of operations. Therefore, RSA requires a more powerful system in order to encrypt/decrypt large files with sizes in gigabytes or larger than AES. Hence, AES is more advisable in real life scenarios due to its lower time and hence, lesser system requirements.

Overall, the AES algorithm is more efficient in most real-life scenarios. However, data which absolutely cannot be risked should not be encrypted using the AES algorithm because it is more

[15] Refer to appendix for code

vulnerable to large brute force attacks. In this scenario, asymmetric algorithms like RSA are more useful and do a much better job at encrypting the data.

Limitations

There were some limitations in the experiment that need to be taken a note of:

1. As the number of processes the CPU has cannot be fixed, the values for each result varies. However, to keep this error to a minimum, whenever a new reading had to be taken, the computer was restarted and when booted up, only NetBeans was opened so that the number of processes were kept to a minimum. Moreover, an average of 5 readings were taken so as to provide the most accurate readings possible.
2. Small strings of all types of data were used which wouldn't represent real life scenarios where huge files with sizes over a gigabyte are encrypted. This couldn't be done due to the unavailability of a computer that could perform the task in a reasonable amount of time.

Conclusion

In this paper, the difference between AES Symmetric Encryption Algorithm and RSA Asymmetric Encryption Algorithm's efficiency in terms of **time taken** to encrypt and decrypt and **memory used** during the encryption/decryption process was analysed. Both the algorithms' working was also provided.

After code experimenting and collecting the results, it was found that the hypothesis was proved to be correct as AES was faster than RSA and was more efficient in terms of the memory used. This trend was followed for all three data types, namely numeric, alphabetical and alphanumeric data. There were some limitations in the experimental procedure, the most important one being the randomness in the experiment due to the different number of processes the CPU had. However, precautions were taken in order to keep them to a minimum.

It was found out in the evaluation that AES is better than RSA for most use scenarios except during the transfer of highly sensitive data, where RSA was preferred. Hence, the research question, **"How is AES symmetric encryption algorithm more efficient in regards to the speed and memory used compared to the RSA asymmetric algorithm when encrypting /decrypting alphanumeric data?"** was answered both qualitatively, through the analysis and evaluation and quantitatively, through the results of the experiment.

Bibliography

Benvenuto, Christoforus Juan. “Galois Field in Cryptography.” *Galois Field in Cryptography*, University of Washington, 31 May 2012, Web. 1 Dec. 2019 <https://sites.math.washington.edu/~morrow/336_12/papers/juan.pdf>

Boneh, Dan. “The RSA Cryptosystem.” Stanford University, n.d. Web. 1 Dec. 2019 < http://crypto.stanford.edu/~dabo/courses/cs255_winter03/rsa-lecture.pdf>

“Euler's Totient Function and Euler's Theorem.” *Chapter 5: Elementary Number Theory*. Imperial College London, n.d. Web. 1 Dec. 2019. <https://www.doc.ic.ac.uk/~mrh/330tutor/ch05s02.html>

“RSA Cryptosystem.” *Princeton University*, The Trustees of Princeton University, Web. 1 Dec. 2019 <https://introcs.cs.princeton.edu/java/99crypto/RSA.java.html >

Dhiraj. “RSA Encryption and Decryption in Java.” *Devglan*, Devglan, 10 Mar. 2018, Web. 12 Jun. 2019 <https://www.devglan.com/java8/rsa-encryption-decryption-java>

“The Mathematical Algorithms of Asymmetric Cryptography and an Introduction to Public Key Infrastructure.” *Infosec Resources*, 7 Feb. 2017, Web. 9th September 2019 <https://resources.infosecinstitute.com/mathematical-algorithms-asymmetric-cryptography-introduction-public-key-infrastructure/#gref>

“What Is a Brute Force Attack?” *Forcepoint*, Forcepoint, 30 Oct. 2019, Web. 1 Dec. 2019 <https://pdfs.semanticscholar.org/1bbe/ac3a18d5d1877868f67abfe4dc240d819389.pdf?_ga=2.226994590.2004659540.1582963925-808173691.1582875584>

DeMuro, Jonas. “What Is AES?” *TechRadar*, TechRadar Pro, 29 Oct. 2018, Web. 3 Sept. 2019

Jackob, Melis. “History of Encryption.” Web. 28 Aug. 2019
<https://www.techradar.com/in/news/what-is-aes>

Jeeva, AL, V. Palanisamy, and K. Kanagaram. “[PDF] COMPARATIVE ANALYSIS OF PERFORMANCE EFFICIENCY AND SECURITY MEASURES OF SOME ENCRYPTION ALGORITHMS: Semantic Scholar.” *Semantic Scholar*. International Journal of Engineering Research and Applications (IJERA), May 2012. Web. 1 Dec. 2019.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.416.1532&rep=rep1&type=pdf>

Kumar, Yogesh, Rajiv Mundal, and Harsh Sharma. “Comparison of Symmetric and Asymmetric Cryptography with Existing Vulnerabilities and Countermeasures.” *Semantic Scholar*. IJCSMS International Journal of Computer Science and Management Studies, Oct. 2011. Web. 1 Dec. 2019
<https://pdfs.semanticscholar.org/d139/89613117ab2cfcf25bb0bfbb94dc71360bad.pdf?_ga=2.261030158.2004659540.1582963925-808173691.1582875584>

Lord, Nate. “What Is Data Encryption? Definition, Best Practices & More.” *Data Insider*, Digital Guardian, 15 July 2019, Web. 27 Aug. 2019
<https://digitalguardian.com/blog/what-data-encryption>

Mehrotra Seth, Shashi. “Comparative Analysis Of Encryption Algorithms For Data Communication.” *International Journal for Computer Science and Technology* , IJCST, June 2011, Web. 1 Dec. 2019
<http://www.ijcst.com/vol22/2/shashi.pdf>

Patil, Priyadarshini D, et al. “A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish.” *International Conference on Information Security and Privacy, 11-12th December 2015, Nagpur, India*, ScienceDirect.com, 2016. Web. 1 Dec 2019 <https://www.sciencedirect.com/science/article/pii/S1877050916001101#!>

Publishers. “Symmetric vs. Asymmetric Encryption – What Are Differences?” *Global SSL Provider*, SSL2BUY, 7 Feb. 2019, Web. 27 Aug. 2019 <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>

Ray, Dhiraj. “AES Encryption and Decryption in Java(CBC Mode): Java Code Geeks - 2019.” *Java Code Geeks*, Java Code Geeks, 12 Mar. 2018, Web. 12 Jun. 2019 <https://www.javacodegeeks.com/2018/03/aes-encryption-and-decryption-in-javacbc-mode.html>

Rouse, Margaret, et al. “What Is Advanced Encryption Standard (AES)? - Definition from WhatIs.com.” *SearchSecurity*, Mar. 2017, Web. 1 Dec. 2019 <https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard>

Smirnoff, Peter, and Dawn M Turner. “Symmetric Key Encryption - Why, Where and How It's Used in Banking.” *Cryptomathic*, Cryptomathic, 18 Jan. 2019, Web. 27 Aug. 2019
What Is a Cipher? - Definition from Techopedia.” *Techopedia.com*, Techopedia, Web. 28 Aug. 2019
<https://www.cryptomathic.com/news-events/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking>

Kanthety, Sundeep Saradhi. “NETWORK SECURITY - RSA ALGORITHM.” *YouTube*. YouTube, 18 Jan. 2018. Web. 3 Sept. 2019.
<https://www.youtube.com/watch?v=2Z3toEiY5lI&t=1243s>

Kanthety, Sundeep Saradhi. “NETWORK SECURITY- AES (ADVANCED ENCRYPTION STANDARD) Algorithm.” *YouTube*. YouTube, 11 Jan. 2018. Web. 3 Sept. 2019. <https://www.youtube.com/watch?v=vZ7YQ67Cbtc&t=1508s>

Appendix

The following code was used for my experiments (heavily adapted from the sources cited below).

I used it to run the programs for AES and RSA and get the time taken and memory used.

Program for AES Encryption Algorithm[16]:

1. Alphabetical data

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;


import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;


public class AES {

    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String strToEncrypt, String secret)
```

[16]Ray, Dhiraj. "AES Encryption and Decryption in Java(CBC Mode): Java Code Geeks - 2019." *Java Code Geeks*, Java Code Geeks, 12 Mar. 2018, Web. 12 Jun. 2019 <https://www.javacodegeeks.com/2018/03/aes-encryption-and-decryption-in-javacbc-mode.html>

```
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new
String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }

    public static void main(String[] args)
    {
        final String secretKey = "this might be the longest key in the world";

        Scanner obj=new Scanner(System.in);

        String encryptedString = AES.encrypt("Buy me a cake", secretKey) ;
        String decryptedString = AES.decrypt(encryptedString, secretKey) ;

        System.out.println(encryptedString);
        System.out.println(decryptedString);
        long startTime=System.nanoTime();
        long endTime=System.nanoTime();

        System.out.println("The time taken:"+(endTime-startTime));
    }
}
```

2. Numeric data

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;


import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;


public class AES {

    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(int intToEncrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
```

```
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new
String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }

    public static void main(String[] args)
    {
        final String secretKey = "this might be the longest key in the world";

        Scanner obj=new Scanner(System.in);

        String encryptedString = AES.encrypt(1234567890, secretKey) ;
        String decryptedString = AES.decrypt(encryptedString, secretKey) ;

        System.out.println(encryptedString);
        System.out.println(decryptedString);
        long startTime=System.nanoTime();
        long endTime=System.nanoTime();

        System.out.println("The time taken:"+(endTime-startTime));
    }
}
```

3. Alphanumeric data

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;


import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;


public class AES {

    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
```

```
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String strToEncrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new
String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }

    public static void main(String[] args)
    {
        final String secretKey = "this might be the longest key in the world";
```

```
        Scanner obj=new Scanner(System.in);

        String encryptedString = AES.encrypt("430 Wood Street, 47906, Jakarta2",
secretKey) ;
        String decryptedString = AES.decrypt(encryptedString, secretKey) ;

        System.out.println(encryptedString);
        System.out.println(decryptedString);
        long startTime=System.nanoTime();
        long endTime=System.nanoTime();

        System.out.println("The time taken:"+(endTime-startTime));
    }
}
```

Program for RSA Encryption Algorithm[17]:

1. Alphabetical data

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class RSA {
    private BigInteger p;
    private BigInteger q;
    private BigInteger N;
    private BigInteger phi;
    private BigInteger e;
    private BigInteger d;
    private int bitlength = 1024;
    private Random r;

    public RSA() {
        r = new Random();
        p = BigInteger.probablePrime(bitlength, r);
        q = BigInteger.probablePrime(bitlength, r);
        N = p.multiply(q);
        phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0) {
```

[17] "RSA Encryption and Decryption in Java." *Devglan*, Devglan, 10 Mar. 2018, Web. 12 Jun. 2019 <https://www.devglan.com/java8/rsa-encryption-decryption-java>

```
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }

    public RSA(BigInteger e, BigInteger d, BigInteger N) {
        this.e = e;
        this.d = d;
        this.N = N;
    }

    @SuppressWarnings("deprecation")
    public static void main(String[] args) throws IOException {
        RSA rsa = new RSA();
        DataInputStream in = new DataInputStream(System.in);
        String teststring;

        teststring = “buy me a cake”;
        System.out.println("Encrypting String: " + teststring);
        System.out.println("String in Bytes: "
                + bytesToString(teststring.getBytes()));
        // encrypt
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        // decrypt
        byte[] decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
        System.out.println("Decrypted String: " + new String(decrypted));

        long startTime = System.nanoTime();
        long endTime = System.nanoTime();

        System.out.println("The time taken:" + (endTime - startTime));
    }

    private static String bytesToString(byte[] encrypted) {
        String test = "";
        for (byte b : encrypted) {
            test += Byte.toString(b);
        }
        return test;
    }

    // Encrypt message
    public byte[] encrypt(byte[] message) {
        return (new BigInteger(message)).modPow(e, N).toByteArray();
    }

    // Decrypt message
    public byte[] decrypt(byte[] message) {
        return (new BigInteger(message)).modPow(d, N).toByteArray();
    }
}
```

```
BigInteger(message)).modPow(d, N).toByteArray();
    }
}
```

2. Numeric data

```
3.  import java.io.DataInputStream;
    import java.io.IOException;
    import java.math.BigInteger;
    import java.util.Random;

    public class RSA {
        private BigInteger p;
        private BigInteger q;
        private BigInteger N;
        private BigInteger phi;
        private BigInteger e;
        private BigInteger d;
        private int bitlength = 1024;
        private Random r;

        public RSA() {
            r = new Random();
            p = BigInteger.probablePrime(bitlength, r);
            q = BigInteger.probablePrime(bitlength, r);
            N = p.multiply(q);
            phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
            e = BigInteger.probablePrime(bitlength / 2, r);
            while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) <
    0) {
                e.add(BigInteger.ONE);
            }
            d = e.modInverse(phi);
        }

        public RSA(BigInteger e, BigInteger d, BigInteger N) {
            this.e = e;
            this.d = d;
            this.N = N;
        }

        @SuppressWarnings("deprecation")
        public static void main(String[] args) throws IOException {
            RSA rsa = new RSA();
            DataInputStream in = new DataInputStream(System.in);
            int testInt;

            testInt = 123;
            System.out.println("Encrypting String: " + teststring);
            System.out.println("String in Bytes: "
                    + bytesToString(teststring.getBytes()));
            // encrypt
            byte[] encrypted = rsa.encrypt(teststring.getBytes());
            // decrypt
```

```
            byte[] decrypted = rsa.decrypt(encrypted);
            System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
            System.out.println("Decrypted String: " + new String(decrypted));

            long startTime = System.nanoTime();
            long endTime = System.nanoTime();

            System.out.println("The time taken:" + (endTime - startTime));
        }

        private static String bytesToString(byte[] encrypted) {
            String test = "";
            for (byte b : encrypted) {
                test += Byte.toString(b);
            }
            return test;
        }

        // Encrypt message
        public byte[] encrypt(byte[] message) {
            return (new BigInteger(message)).modPow(e, N).toByteArray();
        }

        // Decrypt message
        public byte[] decrypt(byte[] message) {
            return (new BigInteger(message)).modPow(d, N).toByteArray();
        }
    }
4.  BigInteger(message)).modPow(d, N).toByteArray();
        }
    }
```

3. Alphanumeric data

```
4.  import java.io.DataInputStream;
    import java.io.IOException;
    import java.math.BigInteger;
    import java.util.Random;

    public class RSA {
        private BigInteger p;
        private BigInteger q;
        private BigInteger N;
        private BigInteger phi;
        private BigInteger e;
        private BigInteger d;
        private int bitlength = 1024;
        private Random r;

        public RSA() {
            r = new Random();
            p = BigInteger.probablePrime(bitlength, r);
            q = BigInteger.probablePrime(bitlength, r);
            N = p.multiply(q);
            phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
```

```
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) <
0) {
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }

    public RSA(BigInteger e, BigInteger d, BigInteger N) {
        this.e = e;
        this.d = d;
        this.N = N;
    }

    @SuppressWarnings("deprecation")
    public static void main(String[] args) throws IOException {
        RSA rsa = new RSA();
        DataInputStream in = new DataInputStream(System.in);
        String teststring;

        teststring = “430 Wood Street, 47906, Jakarta”;
        System.out.println("Encrypting String: " + teststring);
        System.out.println("String in Bytes: "
                + bytesToString(teststring.getBytes()));
        // encrypt
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        // decrypt
        byte[] decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
        System.out.println("Decrypted String: " + new String(decrypted));

        long startTime = System.nanoTime();
        long endTime = System.nanoTime();

        System.out.println("The time taken:" + (endTime - startTime));
    }

    private static String bytesToString(byte[] encrypted) {
        String test = "";
        for (byte b : encrypted) {
            test += Byte.toString(b);
        }
        return test;
    }

    // Encrypt message
    public byte[] encrypt(byte[] message) {
        return (new BigInteger(message)).modPow(e, N).toByteArray();
    }

    // Decrypt message
    public byte[] decrypt(byte[] message) {
        return (new BigInteger(message)).modPow(d, N).toByteArray();
```

```
        }
    }
5.  BigInteger(message)).modPow(d, N).toByteArray();
        }
    }
```