Enhancing Protection for Database Management System via Encryption using Rivest-Shamir-Adleman (RSA) and Advanced Encryption Standard (AES) Algorithm

Muhammad Aufar Rizqi Kusuma - 13524061
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ksmaachmad@gmail.com, 13524061@std.stei.itb.ac.id

***Abstract*— Personal information has become an asset in digital world nowadays. Exchange in information and digital transformation has become vast and widely-applied in various sector. This changes, unfortunately, comes with a trade-off. The risk of leakage for personal information increases, leading to misuses of it by unauthorized party. One of the factor that can make it happen is poor database management by government/organization. The storage of sensitive information in plain text was still a common practiced. Therefore, an approach must be created to tackle this. This paper introduces an alternative solution to enhance database management by encrypting sensitive information in a database using hybrid encryption combining RSA and AES. Experimental results demonstrate a database management system created using Python and SQLite, followed by RSA and AES being applied to the database, especially to sensitive field in the database.**

Keyword: Database management system, RSA, AES, Cryptography, Number theory.

I. INTRODUCTION

In our digital world today, safeguarding personal data is one of the most important aspects when creating a digital product. Digitalization makes all data, including important data, more accessible. On one hand, this can help digital transformation in various lines of life such as health, community services, and so on. Services that were unthinkable to exist become possible with digitalization. However, this also comes with the risk of data leakage or theft. These data may be misused by irresponsible individuals or parties in the future. This kind of security breach is not only materially detrimental, but can also lead to legal issues with the aggrieved parties. We have seen the failures of several organizations and government agencies in protecting the personal data of their users. For example, in 2024, the Temporary National Data Center (PDNS), an Indonesian government agency responsible for protecting the personal data of Indonesians, suffered a cyber attack that caused sensitive data to leak. As a result, 210 agencies both central and regional were affected by the attack [1].

Reflecting from the previous failures of organizations in storing essential information, clearly we see there's a huge gap of how to store data in a database in a secure way especially when it come to securing data in a large volume. Traditionally, a Database Management System (DBMS) implementation often rely on access control and firewalls but may lack strong internal encryption, making stored data vulnerable if attackers bypass authentication or gain physical access. So, the million question is, how to secure data both when the data is at rest and in transit as such so that the data can be distributed and stored safely without significantly degrading system performance?

Fortunately, number theory, one of the branch in discrete mathematics, offered us a robust tool that can be utilize to encapsulate information in a systematic way. Rivest-Shamir-Adleman (RSA) algorithm is one of the algorithm frequently used to do so. It is being remarked as one of the robust algorithm in encrypting information, thanks to it's computational complexity that involve huge prime number factorization.

In practical usage, to ensure both efficiency and strong security in database systems, the combination of Advanced Encryption Standard (AES) and Rivest–Shamir–Adleman (RSA) algorithms is widely adopted as a hybrid encryption approach. AES is a symmetric-key algorithm known for its high speed and efficiency in encrypting large volumes of data, such as entire tables or database records. Unfortunately, because AES is a symmetric encryption (where there are only one key that is both used to encrypt and decrypt the data), the problem lies in the secure distribution of keys. To handle that, RSA will be use to create a pair of public and private key to enable secure key exchange and identity verification.

This paper aims to demonstrate a hybrid encryption scheme to enhance DBMS security using AES for data encryption and RSA for secure key management. A simulation will be created using Python and SQLite3 to demonstrate the effectiveness of this approach in a lightweight and practical implementation. Hopefully, it can give a picture of RSA and AES algorithm implementation in term of securing sensitive information.

II. THEORETICAL FRAMEWORK

A. Number Theory

Number theory is one of the branch in discrete mathematics that study about the set of integers and their properties [2]. The topic that usually arose from number theory is divisibility, modular arithmetic, prime number, Greatest Common Divisor (GCD), congruences, cryptography, and etc.

Number theory gives a foundation for various encryption algorithm, including RSA. RSA itself become a powerful tool in encrypting message thanks to the nature of difficulty of solving prime factorization for huge number.

The following concept are usually found when discussing encryption algorithm.

Divisibility. If a and b are integers with $a \neq 0$, we say that a divides b if there is an integer c such that $a = bc$. The notation $a \mid b$ denotes that a divides b. We write $a \nmid b$ when a does not divides b. When an integer is divided by a positive integer, we can represent the integer using its quotient and remainder. According to The Division Algorithm, let a be an integer and d a positive integer. Then, there are unique integer q and r, with $0 \leq r < d$, such that

$$a = dq + r.$$

Similarly, we can use the notation

$$r = a \bmod d.$$

to represent when a is divided by d, the residue/remainder of the division is r.

Greatest Common Divisor (GCD). If a and b is non-zero integers, the greatest common divisor between a and b, denote as GCD(a, b), is define as the largest integer d such that $d \mid a$ and $d \mid b$. We can write this as

$$\text{GCD}(a, b) = d.$$

The process of finding GCD can be perform intuitively by guessing the factor that can both divide a and b. Usually the guess was start from the smallest possible factor (which is 2).

Further more, by combining the definition of GCD and The Division Algorithm, we will obtain an algorithm to find the GCD between two number without have to *brute force* all the possible factor between the two number. This algorithm, that can find the GCD between two numbers a and b, was called Euclidean Algorithm [2]. Given two non-negative integers m and n where $m \geq n$, The Euclidean Algorithm is stated as follow.

Algorithm 1 Euclidean Algorithm

Input: m, n (integers, $m \geq n$, and $m, n \geq 0$)
Output: GCD(m, n)

```
r ← 0
while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
end while
return m    ▷ Because now n = 0, so PBB(m, n) = m
```

Fig. 1: Euclidean Algorithm for Finding $GCD(m, n)$. Adapted from [3].

Modular Arithmetic. If a and b are integers and m is a positive integer, then a *is congruent to* b *modulo* m if m divides $a-b$ (denoted as $m \mid (a-b)$). We use the notation $a \equiv b \pmod{m}$ to indicate that a is congruent to b in modulus m. If a and b are not congruent in modulo m, we write $a \not\equiv b \pmod{m}$.

As the consequences of congruence definition, the following theorem will hold. If $a \equiv b \pmod{m}$ and c is an arbitrary integers, then

1) $(a + c) \equiv (b + c) \pmod{m}$
2) $ac \equiv bc \pmod{m}$
3) $a^p \equiv b^p \pmod{m}$, where p is a non-negative integers.

And, if $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then

1) $(a + c) \equiv (b + d) \pmod{m}$
2) $ac \equiv bd \pmod{m}$.

Prime number. Every integer greater than 1 is at least can be divided by 1 and itself. By definition, a number that can only be divided by 1 and itself is what we called prime number. On the contrary, a number that is not a prime number is called composite number. The prime number become a unique object to study due to its ability to express every integer in its terms. According to The Fundamental Theory of Arithmetic, every integer greater than 1 can be written as a prime or as the product of two or more primes, where the prime factors are written in order of non-decreasing size.

Modulus Inverse. If p and m is relatively prime and $m > 1$, then the inverse modulus of $p \bmod m$ exist. Further more, inverse modulus of $p \bmod m$ is an integer x such that

$$px \equiv 1 \pmod{m}$$

hence,

$$p^{-1} \bmod m = x.$$

B. Cryptography

Etymologically, cryptography comes from the words “crypt” which means "hidden" and "graphy" which means "writing". By definition, cryptography is a method used to protect data and communication by converting information into codes, ensuring privacy, accuracy, and identity verification [4]. The method being used to encrypt a text was usually arose from a mathematical concept and a set of rule-based calculations known as algorithms. Cryptography itself will encapsulate text/information so that the messages is looks like a "gibberish" text that is difficult to decode and read by unauthorize party, where in fact, it can be decoded using the correct key and being read to the right party.

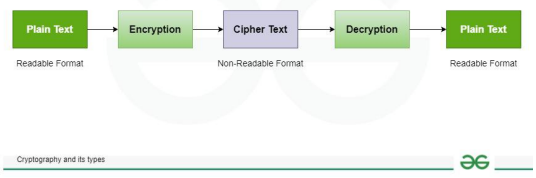

Fig. 2: Simplified Process of Encryption. Source: [4]

Cryptography become a popular choice for various application due to its features that aligned for many purposes. The features of cryptography is as follows.

1) **Confidentiality**. Information can only be be accessed by authorized party. Hence, people who don't have an access to it won't be allowed to see it.
2) **Integrity**. Party that received the message is assured to be given an information that is original and not been altered in any way.
3) **Non-repudiation**. The creator/sender of information cannot deny his intention to send information at a later stage.
4) **Authentication.** The identities of the sender and receiver are confirmed. As well destination/origin of the information is confirmed.
5) **Interoperability**. Cryptography facilitates secure communication across heterogeneous systems and computing environments.
6) **Adaptability**. Cryptography evolves constantly to counteract new threats and accommodate technological developments.

There are a few types of cryptography, namely Symmetric Key Cryptography, Asymmetric Key Cryptography and Hash functions. **Symmetric Key Cryptography** is an encryption system where the sender and the receiver use a single common key to encrypt and decrypt messages. It is, of course, faster and simpler since it only generate one key to decrypt the information. But, there's a serious issue when we want to exchange the key between the sender and receiver. Since it only use one key, if an outsider already got the key, they can interfere with the information, which can be dangerous.

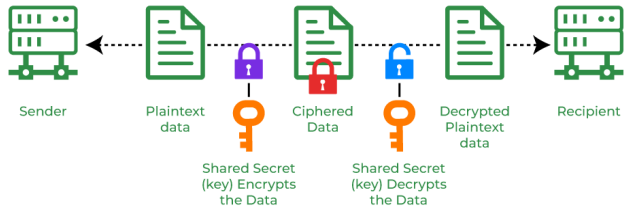

Fig. 3: Symmetric Key Cryptography. Source: [4].

Nevertheless, symmetric key cryptography is still use in many application. The method still deem as an effective alternative to encrypt data under certain condition, especially for small scale. The usage of this type of cryptography for a larger scale will require a combination with other type of cryptography such as Asymmetric Key Cryptography. The most popular symmetric key cryptography systems are Data Encryption Systems (DES) and Advanced Encryption Systems (AES).

Asymmetric Key Cryptography is cryptography method that involve the use a pair of keys to encrypt and decrypt an information. A sender's public key is used for encryption and a receiver's private key is used for decryption. It must be underlined that public key and private key are two different keys. As the consequence, to decrypt the information from a sender, the receiver must have a private key specifically for that information. The most popular asymmetric key cryptography algorithm is RSA algorithm.

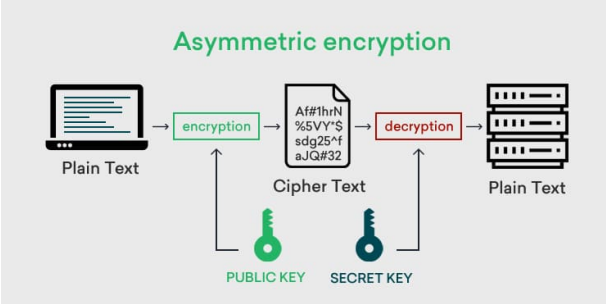

Fig. 4: Symmetric Key Cryptography. Source: [5].

Hash Functions is cryptography method that doesn't required any keys to encrypt or decrypt the information. It uses mathematical equations to generate a hash message for arbitrary length of message and the output will be of fixed length. Hash function has a different purposes compare to the previous cryptography method. Hash function is a one-way encryption that maps your input of all sizes to a unique output of a fixed length in bits. Usually, it is use for comparison purposes.

C. Rivest-Shamir-Adlmean (RSA) Encryption

Rivest-Shamir-Adlmean (RSA) encryption is a public-key cryptography algorithm which means it works on two distinct keys, consist of public and private keys [7]. The public key is used for encryption and is known to everyone. On the contrary, private key is use to decrypt the information and only known to the receiver of the information. It is essential to kept secret the private key since the leakage of it will lead to message hijacking.

RSA algorithm relies on computational complexity of factorization of large numbers and modular arithmetic to convert plain text to ciphered text, and *vice versa*. It consist of three important steps as follows.

1) Key generation: Creating public and private key,
2) Encryption: Sender encrypt information using public key. As result, the plain text will be converted to cipher text,
3) Decryption: Decrypting the cipher text using private key to get the original data.

Key Generation. In this step, a pair of keys will be generated. Public key will be available to everyone, particularly to the receiver and the sender. For private key, it will only available to the receiver. As the consequence, the sender won't have an access to decrpyt the message, ensuring there are no alteration to the original message. The steps of the key generation is as follow.

1) Choose two prime numbers, p and q. These two prime numbers must be kept secret.

2) Calculate the value of $n = p \cdot q$. n is a publicly known value.
3) Calculate the Euler-Totient function of n.

$$\begin{aligned}\phi(n) &= \phi(p \cdot q) \\ &= \phi(p) \cdot \phi(q) \\ &= (p-1) \cdot (q-1).\end{aligned}$$

4) Choose an integer e, for public key, that satisfies

$$\text{GCD}(e, \phi(n)) = 1 \quad \text{and} \quad 1 < e < \phi(n)$$

5) Calculate private key d that satisfies

$$ed \equiv 1 \pmod{\phi(n)}.$$

This algorithm yield a tuple of public key (n, e) and private key (n, d).

Encryption. To encrypt a message M, first convert it to numerical representation using ASCII and other encoding schemes. Now, use the public key (n, e) to encrypt the message and get the cipher text using the formula:

$$C = M^e \bmod n$$

where *C* is cipher text and e, n is a component of a public key.

Decryption. To decrypt the cipher text C, use the private key (n, d) and get the original data using the formula:

$$M = C^d \bmod n$$

where *M* is the message and d,n are parts of private key [7]. Notice that calculating $M^e \bmod n$ directly will be inefficient and slow because e is huge (range from hundreds to thousand of bits) and computing M^e directly will take a lot of resources.

Fortunately, from [8], the writer gives an alternative to compute it. The writer claim that "computing $M^e \bmod n$ only require $2 \cdot \log_2(e)$ multiplications and $2 \cdot \log_2(e)$ divisions" through this procedure. This procedure was called "exponentiation by repeated squaring and multiplication." The procedure is shown in Fig. 5.

This procedure can be apply both for public and private keys, since they shared similar mathematical operations to find them. In terms of time complexity, a high-speed computer can encrypt a 200-digit message M in a few seconds; special-purpose hardware would be much faster. The encryption time per block increases no faster than the cube of the number of digits in n.

D. Advance Encryption Standard (AES)

Advanced Encryption Standard (AES) is a widely respected encryption method designed to protect data by transforming it into an unintelligible form that requires a specific key to decode. It was established by the National Institute of Standards and Technology (NIST) in 2001 [9]. It is widely use today, making it a replacement for the previous encryption standard, DES (Data Encryption Standard) despite being harder to implement. It is categorized as symmetric key cryptography because it only use one key to encrypt

Algorithm 2 Modular Exponentiation by Repeated Squaring

```
1:  procedure ModExp(M, e, n)
2:      C ← 1
3:      Let e_k e_{k-1} ... e_0 be the binary representation of e
4:      for i ← k down to 0 do
5:          C ← (C × C) mod n                ▷ Square step
6:          if e_i = 1 then
7:              C ← (C × M) mod n            ▷ Multiply step
8:          end if
9:      end for
10:     return C
11: end procedure
```

Fig. 5: Modular exponentiation by repeated squaring algorithm to calculate $C^d \bmod n$. Adapted from:[8].

and decrypt the information. AES support various length of key, start from 128 bit, 192 bit, and 256 bit. This length of key ultimately lead to the difficulty of decrypting a message, even for super computer itself. With 128 bit, there will be at minimum $2^{128} = 3.4 \times 10^{38}$ possibilities. As comparison, a super computer that can iterate up to 1 million key per milisecond will require 5.4×10^{18} years to *brute force* all possible combination.

Algorithm that being used in AES was called *Rijndael* algorithm. Similar to DES, *Rijndael* use techniques such as permutation, shifting, and substitution that being applied for several rounds and utilize different key for different round (called *round key*) [10]. To be precise, the framework of *Rijndael* algorithm is as follows.

1) AddRoundKey. *XOR* between initial state (plain text) with *cipher key*. This process was called *initial round.*
2) The number of round (*N*) will be depend on the key size. 128 bits will do 10 rounds, 192 bits will do 12 rounds, and 256 bits will do 14 rounds. These number of round included the final round. For $N - 1$ round, the steps that will be done is as follows.
    a) Subbytes. Substituting byte retrospectively to the substitution table (S-box).
    b) ShiftRows. Shift rows *array state* in *wrapping*.
    c) MixColumns. Randomize data in each column of *array state* by multiplying it with a matrix.
    d) AddRoundKey. *XOR* between *array state* with current *round key*.
3) FinalRound. The process performed on the final round was quite similar to the rest of the round, but without MixColumns.
    a) SubBytes
    b) ShiftRows
    c) AddRoundKey

SubBytes. The `SubBytes()` operation is a reversible, non-linear transformation applied to the AES state, where each byte is individually substituted using a lookup table

known as the S-box. This substitution table is referred to as S-Box. The following figure depict the S-box.

TABLE I: `SBox()` substitution values for the byte xy (in hexadecimal format). Source: [11]

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

The step-by-step of the substitution is as follows. For every byte in *array state*, take an example $S[r, c] = xy$, which in this instance, xy is hexadecimal value representation of $S[r, c]$. what `SubBytes()` does to this value is the value will be substitute with $S'[r, c]$, where $S'[r, c]$ is the intercept of row x with column y in the S-Box. For example, $S[r, c] = 0f$. Thus, by finding the intercept of row 0 and column f in S-Box, $S'[r, c] = 76$. This process is performed for every element in the *array state* matrix.

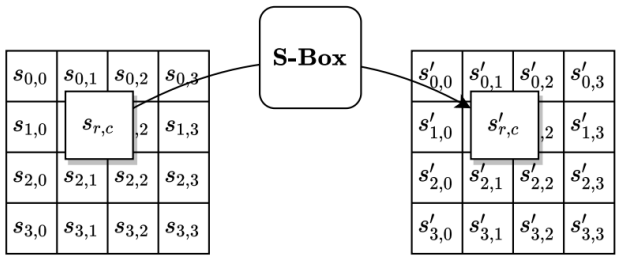

Fig. 6: Illustration of `SubBytes()`. Source: [11].

ShiftRows. `ShiftRows()` is a state transformation in which the bytes in the last three rows are cyclically rotated. The amount each row is shifted depends on its row index r, with each row being shifted by a specific number of positions based on that index. Mathematically, the shifting can be expressed as

$$S'[r, c] = S[r, (c + r) \bmod 4],$$

for $0 \leq r < 4$ and $0 \leq c < 4$. Base on this equation, notice that `ShiftRows()` will shift each rows *r* position to the left. As the consequence, for the first row ($r = 0$), the row will remain unchanged. The illustration of this operation is shown in Fig. 7.

MixColumns. `MixColumns()` is a transformation that multiplies each of the four columns in *state array* with a polynomial $a(x) \bmod (x^4 + 1)$. Each column treated as a 4-term polynomial in GF(2^8). $a(x)$ itself can be expressed as

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

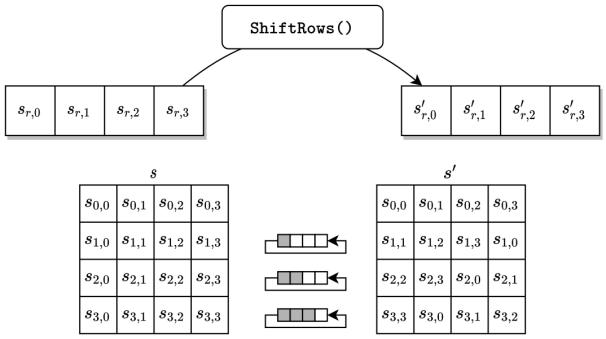

Fig. 7: Illustration of `ShiftRows()`. Source: [11].

and the transformation can be expressed as matrix multiplication over Galois Field as

$$S'(x) = a(x) \oplus S(x).$$

In matrix multiplication, this equation can be written as

$$\begin{bmatrix} S'[0,c] \\ S'[1,c] \\ S'[2,c] \\ S'[3,c] \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[0,c] \\ S[1,c] \\ S[2,c] \\ S[3,c] \end{bmatrix}$$

for $0 \leq c < 4$. Therefore, the individual output bytes are defined as follows:

$$S'[0,c] = (\{02\} \bullet S[0,c]) \oplus (\{03\} \bullet S[1,c]) \oplus S[2,c] \oplus S[3,c]$$
$$S'[1,c] = S[0,c] \oplus (\{02\} \bullet S[1,c]) \oplus (\{03\} \bullet S[2,c]) \oplus S[3,c]$$
$$S'[2,c] = S[0,c] \oplus S[1,c] \oplus (\{02\} \bullet S[2,c]) \oplus (\{03\} \bullet S[3,c])$$
$$S'[3,c] = (\{03\} \bullet S[0,c]) \oplus S[1,c] \oplus S[2,c] \oplus (\{02\} \bullet S[3,c]).$$

The illustration of this operation is shown in the following figure.

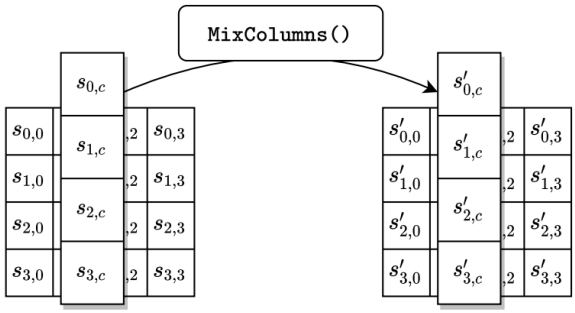

Fig. 8: Illustration of `MixColumns()`. Source: [11].

AddRoundKeys. `AddRoundKeys()` is technique that transform *array state*, using bitwise XOR operation. Furthermore, the procedure will combine each column in *array state* with corresponding round key from the key schedule using XOR operation. The mathematical expression of the procedure is stated as follow.

$$\Big[S'[0,c], S'[1,c], S'[2,c], S'[3,c]\Big] = [S[0,c], S[1,c], S[2,c], S[3,c]] \oplus [w_{(4\times\text{round}+c)}]$$

where *round* is a value in range $0 \leq \text{round} < N$, and $w[i]$ is the array of key schedule words. The transformation in this procedure can be seen in the following figure.

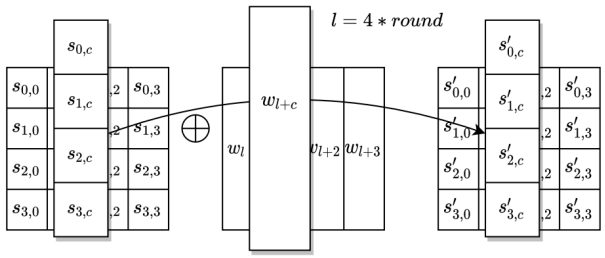

Fig. 9: Illustration of `AddRoundKeys()`. Source: [11].

III. PROPOSED SCHEME

A. Initialization

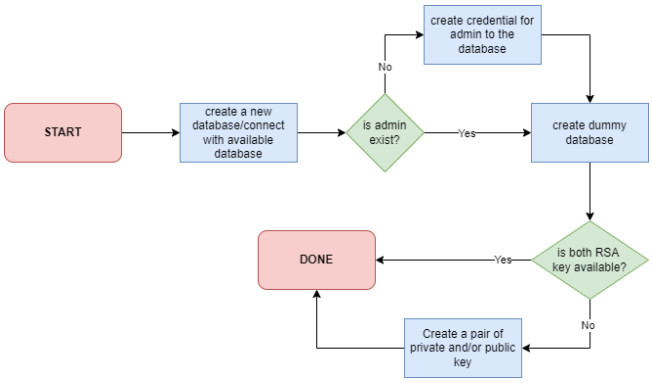

Fig. 10: Scheme for Initialization. Source: Writer's documentation.

The proposed database management system begin by initializing important component of the database. A database will be first created or if there's already a database available in the folder, the program will use the available database. This database will be connected with SQLite, making it possible to save the data even after the program is already done. Through SQLite, any changes to the data will be updated and can be seen *real-time*.

Next, the program will check if there is already an administrator registered in the database. If not, the program will create a credential for the admin. This credential gives the admin access to all features. If it already exists, the program will continue by creating a dummy database. This process is done with the assumption that the database is initially empty, so a dummy database needs to be created in order to run it. In the implementation, the data that will be created only ranges from 1-10 new records.

Finally, the program will check whether an RSA key is available or not. If not, it will generate an RSA-2048 key which, as the name suggests, has a length of 2048 bits. This type of RSA is considered one of the most commonly used RSAs and offers strong security. Later, the key will be used to encapsulate sensitive data such as passwords that have been wrapped by the AES algorithm. Thus, double encryption will be performed on the sensitive data.

B. Main Feature

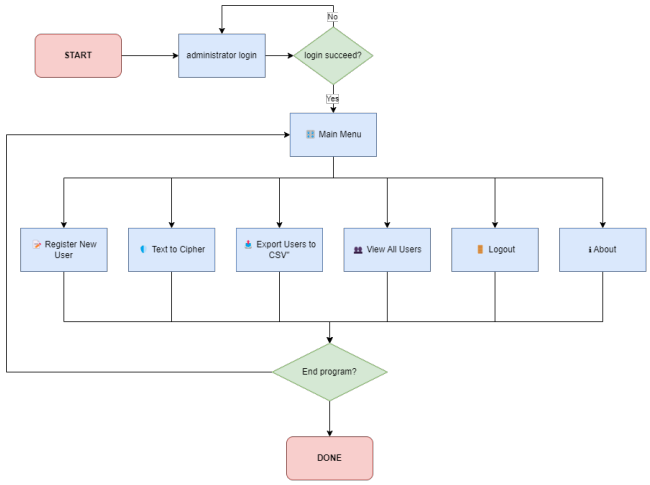

Fig. 11: Scheme for Main Feature. Source: Writer's documentation.

In the main feature, the program will start by requesting credentials from the administrator. Note that the only user who can use this program is the administrator of the database. If the user enters the correct credentials, the user will be taken to the main menu. In the main menu, the admin will be presented with several features that can be accessed. Features that can be accessed by the user are `registerNewUser`, `textToCipher`, `exportCSV`, `viewUsers`, `aboutAdmin`, and `logout`.

For `registerNewUser`, this feature will allow administrator to register a new user and add them to the database. In response, the database in SQLite will append the user's credential to the end of the database. Also, the password of the users will be encrypted using AES encryption and then encrypted again using RSA, making sure the administrator can't gain access to the user's profile.

For `textToCipher`, this feature is actually a bonus feature that is not available in most databases. However, this feature is very useful when you want to encrypt or decrypt a message using the AES algorithm. In encryption, the administrator can type any message into the program and the program will generate a cipher text along with the key to decrypt it. Conversely, in decryption, the administrator can decrypt a cipher provided the administrator has the decryption key for the message.

Broadly speaking, the other features will perform commands that are quite appropriate to their names. Further explanation for the other features will be explained in the implementation section.

IV. IMPLEMENTATION

This program is develop using Python language and SQLite3 as relational database management system

(RDBMS). Python was chosen because its strong support for cryptography algorithm proven by various library such as `PyCryptodome` and `cryptography` being offered to its user. Python also support third-party ecosystem which are needed to develop this program. It also has great GUI framework such as `PyQt5`, `PySide`, and `tkinter`, hence, making it possible to create interactive program within its environment. In addition, this program also use SQLITE as RDBMS due to its ability to be integrated in various programming language, including Python. It doesn't require the user to setup a server, implying a lightweight RDBMS that can be implemented in many computer architecture.

As for Python *built-in* and third-party *library* used in the program, this project utilize several of it to enhance efficiency of the program, avoiding hassle to setup everything from scratch. The *library* used is as follows.

1) `base64` : encode/decode binary data to text (used for encryption output),
2) `sys`: access to system-specific functions like exiting the app,
3) `re`: regular expressions, useful for password strength checks,
4) `sqlite3`: built-in lightweight SQL database for local data storage,
5) `os`: check for file existence or create directories (e.g., for RSA keys),
6) `PyQt5`: building GUI (buttons, labels, input fields, windows),
7) `faker`: generate dummy users (names, dates, usernames, etc.),
8) `pycryptodome`: encryption (AES for password, RSA for key encryption),
9) `csv`: export data to CSV files.

A. *Initialization*

In initialization, the program will setup a database followed by defining the administrator (if administrator doesn't exist in the database) and create dummy record in it using library. The database itself consist of several field, which is id, name, place_of_birth, date_of_birth, gpa, username, encrypted_password, and encrypted_key. The database will be stored in folder `data` in filename `users.db`.

After establishing a database and its property, a pair of RSA key will be generated. The generation will be using RSA-2048 that uses 2048 bits of modulus value (n) and a static public exponent $e = 65537$. Hence, the public key is $(n, e) = (2048, 65537)$. For private key, the generation of it will follow from the RSA algorithm. This key will be store in a folder name `data` with filename `private.pem` and `public.pem`. The implementation of the previous explanation will be depicted in the following block of code.

```
import sqlite3
import os

def init_db():
  conn = connect_db()
  cursor = conn.cursor()
  cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT,
      place_of_birth TEXT,
      date_of_birth TEXT,
      gpa REAL,
      username TEXT UNIQUE,
      encrypted_password BLOB,
      encrypted_key BLOB
    )
  ''')
  conn.commit()
  conn.close()
```

Listing 1: Database initialization. Source: Writers documentation.

```
def insert_admin_user():
  if get_user_by_username("aufarksma") is
      None:
        name = "Muhammad Aufar Rizqi Kusuma"
        pob = "Bekasi"
        dob = "2006-07-02"
        username = "aufarksma"
        password = "aufar12345"
        gpa = 4.0  # Admin GPA

        encrypted_password, encrypted_key =
            encrypt_sensitive_fields(password
            )
        insert_user((name, pob, dob, username
            , gpa, encrypted_password,
            encrypted_key))
        print("[+] Admin user 'aufarksma'
            added.")
   else:
        print("[i] Admin user 'aufarksma'
            already exists.")
def insert_dummy_data(n=10):
   for _ in range(n):
      name = fake.name()
      pob = fake.city()
      dob = str(fake.date_of_birth(
          minimum_age=18, maximum_age=60))
      username = fake.user_name() + str(
          random.randint(100, 999))
      gpa = round(random.uniform(0.0, 4.0),
          2)# Generate GPA (0.00 to 4.00)
      password = fake.password()

      encrypted_data, encrypted_key =
          encrypt_sensitive_fields(password)
      try:
         insert_user((name, pob, dob,
             username, gpa, encrypted_data,
             encrypted_key))
      except ValueError:
         continue  # skip if username already
             exists

    print("Dummy users added.")
```

Listing 2: Administrator and dummy database initialization. Source: Writers documentation.

B. Main Feature

For main feature, after an administrator login, they will be granted access to features available in the program. This program will have a main menu to navigate users to move from one feature to another.

`registerNewUser()`. This feature will allow administrator to register a new user to the database. The program will ask for the credentials of the new user. For the password, there will be a string that will show the evaluation of the password strength being chosen for the new user. If the administrator click submit, then the password will be encrypted using AES + RSA then sent to the database.

`About()`. This feature will allow administrator to see his/her own credentials. The credentials that will be display to the screen is name, place-of-birth, date-of-birth, and username.

`textToCipher()`. This feature is actually a bonus feature that is not available in most databases. However, this feature is very useful when you want to encrypt or decrypt a message using the AES algorithm. In encryption, the administrator can type any message into the program and the program will generate a cipher text along with the key to decrypt it. Conversely, in decryption, the administrator can decrypt a cipher provided the administrator has the decryption key for the message.

`exportUserToCSV()`. This feature will export the users database into a CSV file. The file will be stored in the project's folder.

`viewAllUsers()`. This feature enable administrator to see the database. The database was sorted base on some parameter, which is name and GPA. Both of the parameter can be chosen to be sorted on ascending or descending.

The following code is the implementation of main menu that contain all of the procedure/feature in the program. The detailed source code of each feature is available in the Github repository.

```
from PyQt5.QtWidgets import QWidget,
    QVBoxLayout, QPushButton, QLabel
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt
from gui import register_gui
from gui import admin_profile_gui
from gui import text_to_cipher_gui
from gui import export_gui
from gui import view_users_gui

# Add a global variable to keep the window
    from disappearing immediately
main_menu_window = None

def show_main_menu(user_data):
  global main_menu_window  # important: save
      to global variable

  main_menu_window = QWidget()
  main_menu_window.setWindowTitle("Dashboard"
      )
  main_menu_window.resize(500, 400)

  layout = QVBoxLayout()
  layout.setSpacing(20)
  layout.setContentsMargins(40, 40, 40, 40)

  label = QLabel("Main Menu")
  label.setFont(QFont("Segoe UI", 18, QFont.
      Bold))
  label.setAlignment(Qt.AlignCenter)
  layout.addWidget(label)

  btn_register = QPushButton("Register New
      User")
  btn_register.clicked.connect(lambda:
  register_gui.run_register_gui())
  layout.addWidget(btn_register)

  btn_about = QPushButton("About")
  btn_about.clicked.connect(lambda:
  admin_profile_gui.show_user_profile(
      user_data))
  layout.addWidget(btn_about)

  btn_textTocipher = QPushButton("Text to
      Cipher")
  btn_textTocipher.clicked.connect(lambda:
      text_to_cipher_gui.run_text_cipher_gui
      ())
  layout.addWidget(btn_textTocipher)

  btn_export = QPushButton("Export Users to
      CSV")
  btn_export.clicked.connect(lambda:
      export_gui.run_export_gui())
  layout.addWidget(btn_export)

  btn_view_users = QPushButton("View All
      Users")
  btn_view_users.clicked.connect(lambda:
      view_users_gui.run_view_users_gui())
  layout.addWidget(btn_view_users)

  btn_logout = QPushButton("Logout")
  btn_logout.clicked.connect(main_menu_window
      .close)
  layout.addWidget(btn_logout)

  for btn in [btn_register, btn_about,
      btn_logout, btn_textTocipher,
      btn_export, btn_view_users]:
  btn.setStyleSheet("""
  QPushButton {
        font-size: 14px;
        padding: 12px;
        border-radius: 8px;
        background-color: #4c8bf5;
        color: white;
  }
  QPushButton:hover {
        background-color: #3367d6;
  }
  """)

  main_menu_window.setLayout(layout)
  main_menu_window.show()
```

Listing 3: Main menu implementation. Source: Writers documentation.

V. TESTING & RESULTS

Results was obtained by running the program in Windows PowerShell terminal and test all the possible scenarios. The objective of this testing is to make an assessment toward the program and to find any bug related to it both minor and mayor. The program begin by initializing all of the requirement of the program, including the RSA key. The key generated from this program is as follows.

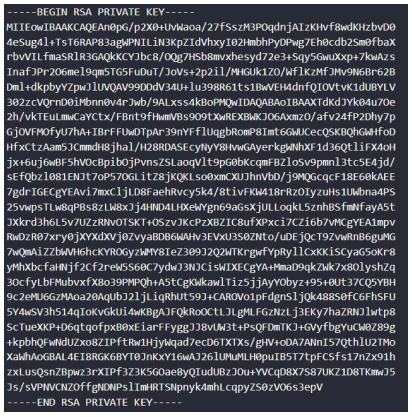

Fig. 12: Private RSA key. Source: Writers documentation.

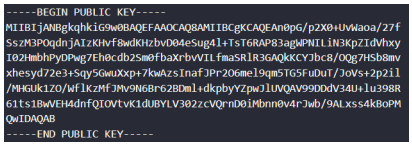

Fig. 13: Public RSA key. Source: Writers documentation.

After the generation of RSA key, a login GUI will appear to the screen. The program will ask for the administrator credentials. If the username and password input is match with the one in the record, then the program will continue.

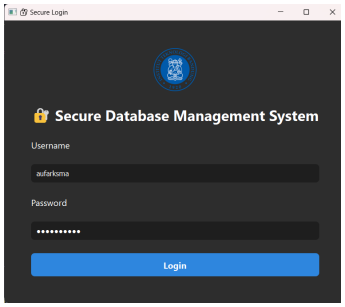

Fig. 14: Login GUI. Source: Writers documentation.

After login, the administrator will be brought to the main menu. Administrator can choose what feature that he/she want.

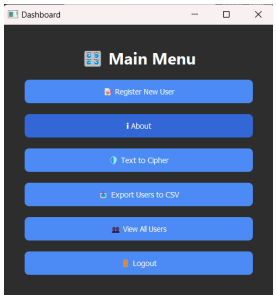

Fig. 15: Main menu GUI. Source: Writers documentation.

The following is testing for `registerNewUser()`.

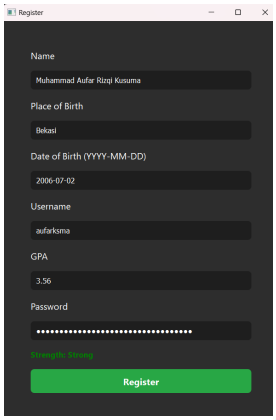

Fig. 16: Register new user GUI. Source: Writers documentation.

The following is testing for `About()`.

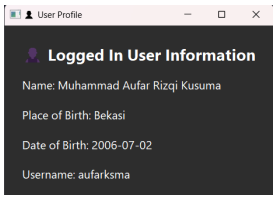

Fig. 17: Administrator's info GUI. Source: Writers documentation.

The following is testing for `textToCipher()`.

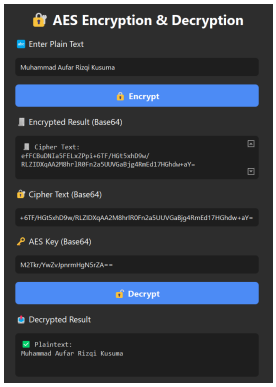

Fig. 18: AES encryption & decryption GUI. Source: Writers documentation.

The following is testing for `exportUsertToCSV()`.

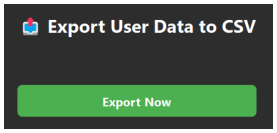

Fig. 19: Export data to CSV GUI. Source: Writers documentation.

The following is testing for `viewAllUsers()`.

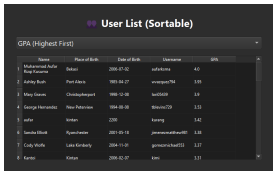

Fig. 20: Show users GUI. Source: Writers documentation.

Due to limited space, full testing will be done via video which will be published on YouTube. The youtube link will be embedded at the end of this paper.

VI. CONCLUSIONS

Through this paper, it successfully demonstrate the implementation of secure database management system using hybrid encryption integrating AES (Advanced Encryption Standard) and RSA (Rivest–Shamir–Adleman) encryption algorithms. Sensitive user data such as passwords are encrypted using AES for speed and efficiency, while AES keys themselves are secured using RSA to ensure strong asymmetric encryption.

The use of Python as the programming language provides flexibility, readability, and access to powerful libraries like PyQt5 for GUI development and PyCryptodome for cryptography. The integration of SQLite3 as the database ensures a lightweight and portable storage solution suitable for small to medium-scale applications. In the end, this paper showcases how modern cryptographic techniques can be applied effectively in real-world software systems to enhance security, protect user data, and ensure privacy.

APPENDIX

The full implementation of this program can be seen in the following Github repository `https://github.com/parkuskus/Makalah-Matdis-RSA_AES_Encryption`

VIDEO LINK AT YOUTUBE

The explain video regarding the paper can be seen in the following link `https://youtu.be/h_9yMdqMrT4`

ACKNOWLEDGMENT

All praise and gratitude are due to Allah Subhanahu wa Ta'ala, whose blessings and mercy have enabled the author to complete this paper. The author extends sincere thanks to Dr. Ir. Rinaldi Munir, M.T., the lecturer of IF1220 – Discrete Mathematics, for his guidance and support throughout the writing process. The author is also grateful to family and friends for their continuous encouragement, and especially to Emilio Justin, Jonathan Kris Wicaksono, Philipp Hamara, and other "AYCE" group member who provided valuable assistance and support during the completion of this work. The author would also like to thank Naufal Agam Ardiansyah, who although separated by distance, still provides emotional support to the author.

REFERENCES

[1] D. Luthfiani, "PDNS diretas, 47 layanan Kemendikbudristek terganggu," Tempo.co, Jun. 24, 2024. [Online]. Available: https://www.tempo.co/hukum/pdns-diretas-47-layanan-kemendikbudristek-terganggu-46152

[2] K. H. Rosen, *Discrete Mathematics and Its Applications*, 7th ed. New York, NY, USA: Mcgraw-Hill, 2011, 1 075 pp, [Online]. Available: https://faculty.ksu.edu.sa/sites/default/files/rosen_discrete_mathematics_and_its_applications_7th_edition.pdf

[3] R. Munir, "Teori Bilangan (Bagian 1)", 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/15-Teori-Bilangan-Bagian1-2024.pdf. [Accessed: Jun. 17, 2025].

[4] GeeksforGeeks, "Cryptography and its types," GeeksforGeeks, https://www.geeksforgeeks.org/computer-networks/cryptography-and-its-types/, [Accessed: Jun. 17, 2025].

[5] Encryption Consulting, "Symmetric vs Asymmetric Encryption," Encryption Consulting, https://www.encryptionconsulting.com/education-center/symmetric-vs-asymmetric-encryption/, [Accessed: Jun. 17, 2025].

[6] Positiwise, "What is a Hash Function within Cryptography? [Quick Guide]," Positiwise, https://positiwise.com/blog/what-is-a-hash-function-within-cryptography-quick-guide, [Accessed: Jun. 17, 2025].

[7] GeeksforGeeks, "RSA Algorithm in Cryptography," [Online]. Available: https://www.geeksforgeeks.org/computer-networks/rsa-algorithm-cryptography/. [Accessed: 17-Jun-2025].

[8] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[9] GeeksforGeeks, "Advanced Encryption Standard (AES)," https://www.geeksforgeeks.org/computer-networks/advanced-encryption-standard-aes/ (accessed Jun. 18, 2025).

[10] R. Munir, Advanced Encryption Standard (AES). [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/Advanced%20Encryption%20Standard%20(AES).pdf (accessed Jun. 18, 2025).

[11] National Institute of Standards and Technology (NIST), Announcing the Advanced Encryption Standard (AES), FIPS PUB 197, Nov. 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf (accessed Jun. 18, 2025).

STATEMENT

I hereby declare that this paper is an original work, written entirely on my own, and does not involve adaptation, translation, or plagiarism of any other individual's work.

Bandung, 19 June 2025

Muhammad Aufar Rizqi Kusuma, 13524061