Abstract — Information inside the database is shared by multiple parties such as internal users, partners, contractors and others. Sensitive data stored in the database could be a target to attackers. The attacker for data stored in the database not only from external but also from within the organization. Adding the database encryption, valuable information in database becomes more secure since the encrypted data ensure the confidentiality of the data. A new affine block cipher named Enhanced Affine Block Cipher technique is proposed for database encryption. This algorithm improves the weakness of the original affine cipher. The new encoding schema and modification Cipher Block Chaining (CBC) mode of operation for block cipher is designed for the new algorithm and then the prototype of the system is built and implemented into existing system for protecting user password. The result has shown that the algorithm is working properly, where the decryption process produced similar output as the original plaintext and it ran through specified configuration and evaluated thoroughly with respect to database approach and algorithm technique to prove the design.

Keywords — Database, Enhanced Affine Block Cipher, Encryption, Decryption.

I. INTRODUCTION

In today’s enterprise environment, database systems are distributed and used in various applications such as e-Business and e-Commerce. These applications are examples of real-time online resources that need to deliver value-added services through high confidentiality and availability of databases. With a networked database in the complex multi-tiered applications, multiple parties such as partners, contractors, internal users and others will share the information inside the database. The sensitive data could be a target to attackers. The attacker for data stored in database not only from external parties but also from internal. However, their vulnerability to external attack increases and critical business data stored in databases are obviously vulnerable for attackers. Therefore, to properly maintain the integrity and confidentiality of the data, database security becomes one of the most urgent challenges in database research. Database security is a wide research area and includes topics such as statistical database security, intrusion detection and most recently privacy preserving data mining [4].

There are some techniques have been done to protect data stored in database such as firewall, intrusion detection system (IDS) and access control but this is not enough. Databases are still vulnerable to be attacked from internal and external threats. Firewall and IDS only provide network layer protection. Access control is based upon the concept of privilege and it is basic for many security features [2]. One of the requirements for database security is database encryption. With database encryption, the valuable information in database becomes more secure since the encrypted data ensure the confidentiality of the data.

Thus, this paper will focus specifically on some of the details on cryptographic algorithm technique used to implement the database encryption. Throughout this paper, the cryptography algorithm that will be used to provide security and confidentiality of data in the database are discussed and elaborated.

II. DATABASE ENCRYPTION

In general, database sharable resource among many user or applications. A multiuser application in distributed system complicates the data security problem imposed upon a database. Hence, security is becoming one of the most urgent challenges in database research and industry. Past studies reviewed that database security is the most common architectures and methodologies for designing secure database [7, 5]. One of the important aspects of database security is database encryption [1, 2, 10].

The original data that is readable and understood is called plaintext or cleartext. Method that used to code a plaintext that can conceal its meaning is called encryption. Once a message has been transformed with an encryption algorithm, the resulting message is called ciphertext. The encryption is used to ensure that information is hidden from unintended person, even from those who can see the encrypted data. In order to be able to read ciphertext, the other process is needed to decipher the ciphertext. The study of encryption and decryption is called cryptography [9]. According to Menezes et al. [6] and
Russell and Gangemi [9] cryptography provides security in the areas of confidentiality, data integrity, authentication, and non-repudiation.

The goal of encryption is to make data unintelligible to unauthorized users and extremely difficult to decipher when it is attack. Symmetric key cryptography is the most commonly used technique to encrypt data in the storage or database. This ciphers use the same key when to encrypt and decrypt the data. There are two types of symmetric ciphers; block ciphers and stream ciphers. Stream ciphers are generally twice as fast as block ciphers but they require the use of unique keys. Block ciphers on the other hand, allow keys to be reused. There are some encryption features of block cipher technology were included in Database Management System (DBMS). The recommended minimum key length for all symmetric key ciphers is 128 bits.

A block cipher is a type of symmetric key cryptography that transforms a fixed length block of plaintext (unencrypted text) data into a block of ciphertext (encrypted text) data of the same length. This transformation takes place under the action of a user-provided secret key. The fixed length is called the block size, and for many block ciphers, the block size is 64 bits. Meanwhile, Cipher Block Chaining (CBC) mode adds a feedback mechanism to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the previous ciphertext block prior to encryption. In this mode, two identical blocks of plaintext never encrypt to the same ciphertext.

Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key. An early and highly influential block cipher design was the Data Encryption Standard (DES), developed at IBM in 1974, and published as a standard in 1977. A successor to DES, the Advanced Encryption Standard (AES), was adopted in 2001 [12].

The affine block cipher [11] is one of the symmetric key cryptography that was known as classical cryptography and it is easier to break by ciphertext-only cryptanalysis. Some improvements have been done on affine cipher. Instead of using single letter, Koblitz [3] shows digraphs in his works but it is still not enough because of the second letter of each ciphertext digraph depends only on the second letter of the plaintext digraph. Thus, one could obtain a lot of information keys from a frequency analysis of the even numbered letters of the ciphertext. In this paper, enhanced affine block cipher algorithm with its encoding schema was designed to overcome affine cipher and it was implemented in securing data stored in database.

III. DATABASE ENCRYPTION APPROACH

There are two main approaches for database encryption which is whether performing encryption and decryption inside the database or performing encryption and decryption outside the database [4, 8]. After reviewing the database encryption, the best ways to secure the information stored in database is database encryption and apply it at outside the database i.e. at application level encryption. This approach was selected because it provides good end-to-end data protection. By using this approach, encryption will be on the column and row basis. Hence, not all data stored in the database will be encrypted. Only sensitive information such as user identification, credit card number and password will be encrypted. By applying this approach, it will be more efficient in reducing the overhead of reading data. The cryptographic algorithm used for the database encryption is designed and implemented in java programming language and it acts as application server whereby the encryption and decryption processes are done at the application level.

This approach applied end-to-end encryption between client and applications server. For encryption process, the data is encrypted at application server and then inserted into the appropriate fields or columns in the database. For decryption process, the encrypted information is retrieved from the database and then decrypts it at application server so that only authorized user can see the information. The keys used to encrypt and decrypt the data in this approach is stored in file storage at application server not in the database. Hence, this approach will add one security layer in securing the data stored in the database. The keys must be found before the attacker can see and know the contents of data. Figure 1 depict database encryption outside the database.

IV. ENHANCED AFFINE BLOCK CIPHER

The analysis on affine block cipher was done and revealed that some new features can be added into its cipher such as the encoding schema and mode operation of block cipher. Therefore the new affine block cipher was designed and called enhanced affine block cipher to overcome the weaknesses of the original affine cipher. For implementation of these algorithms, the activity diagram was used to model the work flow behind the implemented system. The activity diagram is useful in understanding work flow analysis of synchronous behaviours across the process.

Figure 2 shows the process flow of encryption and decryption using Enhanced Affine Block Cipher. As seen in figure 2, the process started with either plaintext or ciphertext format as an input.

When plaintext is taken as an input, the Encoding activity is performed and followed by the Encryption activity and next the DecodingHex activity.
indicates that both Display Result activity and the Store Result in Database activity occur at the same time.

Fig. 2 activity diagram for database encryption using enhanced affine block cipher

Meanwhile if ciphertext is the input, the EncodingHex activity is performed and would then indicate the Decryption activity and next the Decoding activity. The Decoding activity indicates the Display Result activity. Finally the parallel activities are combined to end the activity.

The inputs that have been used in the encryption process are plaintext, key, block length and initial vector. In the decryption process, the ciphertext, key, block length and initial vector are its input. The plaintext was divided into simple and long plaintext. The main purposes of the testing are to validate the functionality of the algorithm and also to ensure that the database encryption is working properly. From the result, it was found that the algorithm is working properly where the decryption process produced a similar output to the original plaintext.

A. The Design of Enhanced Affine Block Technique

The design of enhanced affine block technique would be described in the next sections.

i) Encoding Schema

The encoding schema designed and developed was based on ASCII format. The plaintext and ciphertext is code and decode into certain number or value before encryption or decryption process. Hence, the encoding schema was used to enhanced affine block cipher.

The total of the ASCII characters set is 128. Therefore, the encoding schema is used based on these numbers where it contains encode and decode schema. In this encoding schema, during encryption process, the number will be converted into hexadecimal code whereas during decryption process, the number will be converted into characters.

Before plaintext and ciphertext is encrypted or decrypted, it was broken up into message units (block size). A message unit might be a single letter, a pair of letters (digraphs), a triple of letters or any number of letters. The encoding schema of message unit is done by an enciphering transformation function where it takes any plaintext message unit and transformed into a ciphertext message unit. In other words, it is a map from the set of P all possible plaintext message units to a set of C all possible ciphertext message units. The encoding schema of message unit is also done by deciphering transformation function where it takes any ciphertext message unit and transformed into an original plaintext message unit. In other words, it is also a map from the set of C all possible ciphertext message units to a set of P all possible plaintext message units.

ii) Encode and Decode Schema of Plaintext Message Unit

First, let start with encode schema and the case of a message unit (block size of plaintext message) is single letter in ASCII character (128 characters) was labeled by integer 0, 1, 2, 3... 128-1.

For block size = 1, the message unit of plaintext is \( p = x_1 \).

The formula of encoding schema is as follows:

\[
\text{plaintext of } p \text{ every for so } \\
= \sum_{1}^{128} 1^{-i} x_i \\
p \in \{0,1,2,3,... 128 - 1\} = Z_{128}
\]

With the same techniques, it can be applied for block size equal to two.

For block size = 2, the message unit of plaintext is \( p = x_1x_2 \).

\[
\text{plaintext of } p \text{ every for so } \\
= \sum_{1}^{128} 2^{-i} x_i \\
p \in \{0,1,2,3,... 128 - 1\} = Z_{16384} = Z_{128^2}
\]

Therefore, with the same techniques it could be used for block size = n, the message unit of plaintext is \( p = x_1x_2...x_n \).

\[
\text{plaintext of } p \text{ every for so } \\
= \sum_{1}^{128} n^{-i} x_i \\
p \in \{0,1,2,3,... 128^n - 1\} = Z_{128^n}
\]

In decode schema of plaintext, the value or number was obtained from encrypting process is converted into
appropriate code. The process of converting a number (decimal numbers) into digits \(y_n, y_{n-1}, \ldots, y_1\) and \(y_0\) such that
\[
y = 128^n y_n + 128^{n-2} y_2 + \ldots + y_n
\]

It can be obtained by successively dividing \(y\) by 128 until quotient is 0. So the values are the remainders \(y_n, y_{n-1}, \ldots, y_1, y_0\).

In case of encryption, the combination of these values is in hexadecimal number and is called ciphertext message.

### iii) Encode and Decode Schema of Ciphertext Message Unit

First, let start with encode schema and the case of a message unit (block size of ciphertext message) is single letter in ASCII character (128 characters) was labeled by integer 0, 1, 2, 3, \ldots, 128.

For block size = 1, the message unit of ciphertext is \(c = y_1\)
The formula of encoding schema is as follows:
\[
c = y_1 = \sum_{i=1}^{n} 128^{1-i} y_i \quad \text{so for every } c \text{ of ciphertext}
\]
\[
c \in \{0,1,2,3, \ldots 128^{1} - 1\} = Z_{128}
\]

With the same techniques, it was also apply for block size equal to two.

For block size = 2, the message unit of ciphertext is \(c = y_1y_2\)
\[
c = 128y_1 + y_2 = \sum_{i=1}^{n} 128^{2-i} y_i \quad \text{so for every } c \text{ of ciphertext}
\]
\[
c \in \{0,1,2,3, \ldots 128^{2} - 1\} = Z_{16384} = Z_{128^2}
\]

Therefore, with the same techniques, it was also concluded that as follows;

For block size = \(n\), the message unit of ciphertext is \(c = \prod_{i=1}^{n} y_i\)
\[
c = 128^{n-1}y_1 + 128^{n-2}y_2 + 128^{n-3}y_3 + \ldots + y_n = \sum_{i=1}^{n} 128^{n-i} y_i \quad \text{so for every } c \text{ of ciphertext}
\]
\[
c \in \{0,1,2,3, \ldots 128^{n} - 1\} = Z_{128^n}
\]

In decode schema of ciphertext, the value or number was obtained from decrypting process is converted into appropriate code. The process of converting a number (decimal numbers) into digits \(x_n, x_{n-1}, \ldots, x_1, x_0\) such that
\[
x = 128^{n-1}x_1 + 128^{n-2}x_2 + \ldots + x_n
\]

It can be obtained by successively dividing \(x\) by 128 until quotient is 0. So the values are the remainders \(x_n, x_{n-1}, \ldots, x_1, x_0\).

### B. Design Enhanced Affine Block Cipher

The Affine cipher works by transforming the letters of the alphabet to their corresponding numerical value (which is from 0 to 25), then utilize the encryption formula as follows;
\[
e_{a,b}(x) = (ax + b) \bmod 26
\]

This encryption function must be bijective, and a must have a multiplicative inverse \(\bmod 26\) (gcd\(a,26\)) is equal 1). For decryption function
\[
d_{a,b}(y) = a^{-1}(y-1) \bmod 26
\]

The invertible integers \(26\) mod 26 are set of \(\{1,3,5,7,9,11,15,17,19,21,23,25\}\).

New affine cipher namely enhanced affine block cipher was designed based on the encoding schema as mentioned earlier. For the first step, recall the affine cipher as follows

Let \(P = C = Z_{128^n}\) and \(n\) is block size.
\[
K = \{(a,b) \in Z_{128^n} \times Z_{128^n} : \gcd(a,128^n) = 1\}
\]
for \(K = (a,b) \in \kappa\),
the encryption function is defined as
\[
e_{K}(x) = ax + b \bmod 128^n
\]
and the decryption function is defined as
\[
d_{K}(y) = a^{-1}(y-1) \bmod 26^n
\]
where \((x,y \in 128^n)\).

The second step for enhancement of affine cipher is done by adding modes of operation into the block cipher algorithm. This technique is similar to cipher block chaining (CBC) mode. Figure 3 and figure 4 show that the processes of encryption and decryption of enhanced affine block cipher with its modes of operation.

It was discovered that, by applying CBC mode; during the operation XOR, certain values are more than the value of the modulo 128. Due to the basic properties that is congruence between the value and the modulo; it cannot give exact value during the decryption process. Based on analysis and initial testing, instead of using XOR as mode of operation, this algorithm was used as an additional operation for encryption and subtraction operation for decryption. The mathematical formula for this mode of operation is as follows:

\[
C_i = e_{K}(P_i + C_{i-1}), \quad C_0 = IV \text{ for encryption}
\]
and
\[
P_i = d_{K}(C_i) - C_{i-1}, \quad C_0 = IV \text{ for decryption}
\]
In encryption process, each plaintext block is added with previous ciphertext block, and then encrypted. An initialization vector (IV) is used as a seed for the process. In decryption process, each decrypted ciphertext block is subtracted with the previous ciphertext.

This proposed enhanced affine block cipher could be used for any application systems which needs the sensitive data to be protected.

![Diagram](image.jpg)

**Fig. 3 modes of operation during encryption process**

**Fig. 4 modes of operation during decryption process**

### VI. THE IMPLEMENTATION

In this phase, the development of the system was implemented using java and java server pages code. The system was developed in Netbeans IDE version 4.1 and the database used is PostgreSQL8.1. The Aqua Data Studio 4.5 was used to create table and to query the data in database PostgreSQL8.1. Apache Tomcat5.5 was used as web server and java server pages as its web component language. The enhanced affine block cipher was coded in java beans class where java.math.BigInteger, a class that represents arbitrary precision integers was used to handle the large integer and modular arithmetic involves in this algorithm. Java servlet was used to interact with user input pages and the algorithm in java beans and database. An additional driver such as postgresql-8.0-314.jdbc3.jar is also needed in order to ensure smooth and successful connection between the database and the application of the system.

The Enhanced affine block cipher was implemented in the existing Fraud Management System (FMS). Currently, users can logon to the system by entering user ID and their password. The password stored in database is in cleartext (not encrypted). The password can be obtained by other users if they have privileged to access the database and execute the query. The password also can be intercept by someone else in the network using certain tools.

Performing encryption for securing password within FMS will add more security and prevent unauthorized users to logon to the system. The implementation was done by performing encrypted password in table user account. The architecture of the encrypted password for FMS is showed in Figure 5. The modification was made based on the changes of password data in table user_account. The FMS modules were added by the enhanced affine block cipher are login module, insert new password module and change password module. The modification was also made to handle the reading and store the keys stored in file storage.

![Diagram](image.jpg)

**Fig. 5 applying encrypted password for fms**

The implementation of encrypted password for FMS runs through the specified configuration and evaluated thoroughly with respect to database approach and cryptographic algorithm technique to prove the design.

**A. Enhanced Affine Block Cipher Class Diagram**

The enhanced affine block cipher was converted into class diagram as shown in Figure 6. It shows set of classes, attributes, operations and their relationships between them. Class diagram is used to model the static design view of the system. There are three classes for the enhanced affine block cipher; EncodingSchema, EnhancedAffineCipher and User_Info.
B. Input and Output of Encryption and Decryption Process

This section discusses the input and output of the encryption and decryption process by using enhanced affine block cipher algorithm. Several results are shown based on the different of the inputs.

i) Input and Output of Encryption and Decryption Process

There are two samples of same plaintext with different block size and keys. The two type of block size used are 4 and 64 characters with difference key A and key B and also initial vector.

Table I presents the input of the process using this algorithm for a cipher input block size of 4 characters (32 bits length) and key A = 777771; key B = 123456789; initial vector (IV) = 654321.

<table>
<thead>
<tr>
<th>Cipher key</th>
<th>A = 777771 , B = 123456789 , IV = 654321</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>4 Characters (32 bits length)</td>
</tr>
<tr>
<td>Input</td>
<td>Plaintext : Enhanced affine block cipher</td>
</tr>
<tr>
<td>Output</td>
<td>Ciphertext: 17760d5b212b442a603818457101670c344 3353a311d0c7668d430d3a</td>
</tr>
</tbody>
</table>

The following tables (Table III and IV) are 2 samples of half paragraph plaintext with different block size and keys. The two types of block size used are 4 and 64 characters with difference key A and key B and also initial vector.

Table III and IV present the input of the process using the algorithm for a cipher input block size of 4 characters (32 bits length) and key A = 7; key B = 19; initial vector (IV) = 123456.

<table>
<thead>
<tr>
<th>Cipher key</th>
<th>A = 7 , B = 19 , IV = 123456</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>4 Characters (32 bits length)</td>
</tr>
<tr>
<td>Input</td>
<td>Plaintext : PostgreSQL has a rich set of native data types available to users. Users may add new types to PostgreSQL using the CREATE TYPE command. Table 8-1 shows all the built-in general-purpose data types. Most of the alternative names listed in the Aliases column are the names used internally by PostgreSQL for historical reasons. In addition, some internally used or deprecated types are available, but they are not listed here.</td>
</tr>
<tr>
<td>Output</td>
<td>Ciphertext: 36440a7f55014501e167622104721285a16 4d6051044a7665061b1729326b600723371 37425333f14383d5c5832153e2a0b7b39530</td>
</tr>
</tbody>
</table>
Table IV  The Result of Decryption Process of Half Paragraph of Plaintext with Block Size=4 and Key A= 7, B=19

<table>
<thead>
<tr>
<th>Cipher key</th>
<th>A = 7 , B=19, IV=123456</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>4 Characters (32 bits length)</td>
</tr>
<tr>
<td>Input</td>
<td>Ciphertext : 36440a7f55014510e167622104721285a16 4d6051044e76656181792326b600723371 37423333f1434560e123352153e2a0b739530 579652b677f366c02452b1b554a136c202b3 d5b25e0d675f542a3c25177a06367c79202b3 d6d2b5e0d675f542a3c25177a06367c79202b3 d6d2b5e0d675f542a3c25177a06367c79202b3</td>
</tr>
<tr>
<td>Output</td>
<td>Original message: PostgreSQL has a rich set of native data types available to users. Users may add new types to PostgreSQL using the CREATE TYPE command.</td>
</tr>
</tbody>
</table>

Table V  The Result of Encryption Process of Half Paragraph of Plaintext with Block Size=64 and Key A= 5, B=177771

<table>
<thead>
<tr>
<th>Cipher key</th>
<th>A = 5 , B=177771, IV=123456</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>64 Characters (512 bits length)</td>
</tr>
<tr>
<td>Input</td>
<td>Plaintext : PostgreSQL has a rich set of native data types available to users. Users may add new types to PostgreSQL using the CREATE TYPE command.</td>
</tr>
<tr>
<td>Output</td>
<td>004e1a2b36050b350e286f46440d6e377105 59683c71811020704c167038440d6e377105 7a442a37712e0b5f100d6e4450425121456 1303712c4171427253d1b3c61191a75f7f 002e0c3b7c492d1654253ba761456140069 185f1a1e130c7a2e7f412437344e5b776a7a b31254304772d6500491032d3a1d350e 3667672925c2237510b656136e4957154254</td>
</tr>
</tbody>
</table>
Table VI  The Result of Decryption Process of Half Paragraph of Plaintext with Block Size=64 and Key A= 5, B=177771

<table>
<thead>
<tr>
<th>Cipher key</th>
<th>A= 5, B=177771, IV=123456</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>64 Characters (512 bits length)</td>
</tr>
</tbody>
</table>

Input  ciphertext :  
004e1a2b36050b350e286f46440d6e377105 
59683c71181020704c167038422b5c51036e 
7a442377120e5f100de6450425b124156 
1303712c41714127253d1b3c61191a705f7f 
00263b37c492d1654253b1a761456140069 
185f1a1e130c7a2e7f41234734ae5b2b776a 
b31254340772d65004d9100d3a2d13530e 
366757295c2237510b6551362e49517542 
d1f6f56180d574b097e116a196368317a035 
435333864e327632187d31384d455c43473 
f8623ba03435439554a0248067791e0f7 
0275925965432b2af1f26132825f2f10187 
6207e3e3a6c2f615b540441e24672572 
a4a295a246736163d142001304e50742e1 
4af3d580972285440f103752435524e25473 
04604533b77035a6f0456791b657220d116 
a7d28562c4367380865e1241225e6312326 
609002f75356f068f575754b2143b4f93f553 
338166223d1c252577760087762b4600b6 
35c635a4bb0a7b321606403450a2e203b1a0 
94d777a532369606b2e5751647612675826 
204f405a0343488390a192d16201026323 
2826627252ce073a102366656736803403c 
153d0567a1ff5786975242133b1303de14 
5c5366497229a16723b504b321327683f

Output Original message: PostgreSQL has a rich set of native data types available to users. Users may add new types to PostgreSQL using the CREATE TYPE command. Table 8-1 shows all the built-in general-purpose data types. Most of the alternative names listed in the Aliases column are the names used internally by PostgreSQL for historical reasons. In addition, some internally used or deprecated types are available, but they are not listed here.

VII. CONCLUSION

This paper focused on the design of database encryption at application level using enhanced affine block cipher. This improvement has been made because of the weakness found in the original affine cipher. In this paper, the improvement is made by using a new encoding schema and mode of the operation for the encryption and decryption process. The enhanced affine block cipher is developed and implemented where the selected sensitive data is encrypted outside the database (application level) and then it is inserted into database.

There are two types of samples used namely the plaintext and ciphertext for validation purpose. Two types of block size have been used, i.e. 4 and 64 characters. Each of block size has been testified by using different keys and two types of samples; plaintext and ciphertext. This algorithm is then applied in existing system involves in database.

The enhanced affine block cipher can be used to explore other existing symmetric cryptography algorithms or combine it to other techniques. The mode of operations used in enhanced affine block cipher also can be extended into others approaches. The database encryption can also be applied in hybrid cryptography techniques. This technique can be applied by combining the symmetric key cryptography and asymmetric key cryptography.

References