Chapter 5

DISKTRUST ARCHITECTURAL FRAMEWORK

The architecture involves the components as shown in the Figure 5.1. DiskTrust is a software based approach [33] to implement Hard Disk Security. Once a new user runs the DiskTrust software, it creates a hidden volume of user specified size. For the existing user to access the hidden volume, system will ask for login-password.

![DiskTrust Architecture Diagram]

**Fig. 5.1 DiskTrust Architecture**
Once the login-password is approved hidden volume would be visible to the user. Now the user can store his sensitive data on this volume. When the user stores data on the volume the data will be automatically encrypted. This is called on-the-fly encryption of data. To read the contents of the volume the user has pass a second level authentication. It is implemented using Pattern matching. The entire architecture is as shown in Figure 5.1. The user interface helps the user to select the specific drive, to save his files in encrypted form and retrieve them after decryption. This very simple and user-friendly user interface is developed in Java to implement DiskTrust architecture.

The proposed research work has developed disk security using Partial disk encryption. PDE is best suited for small scale applications where the user wants to secure partial disk contents and get better performance over full disk encryption. Here partial disk specifies a volume/partition on HDD. Initially a specific size volume is created according to the user’s choice. The created volume is made hidden to avoid the data exposure. The authorized person can access the contents of the hidden volume. As mentioned earlier while implementing PDE/FDE use of efficient encryption algorithm is very important to achieve good performance. So the proposed work implements PDE with AES algorithm. The AES algorithms with different key sizes (AES-128, 192,256) are implemented. A modified AES with variation in S-Box design is implemented. A hybrid AES-DES algorithm is proposed to achieve better results to implement disk encryption. The algorithms are experimentally implemented and tested with various data values.

The terminologies used to implement DiskTrust architecture are discussed below. Section 5.1 covers creation of hidden volume, section 5.2 discusses authentication scheme used to implement DiskTrust and section 5.3 discusses implementation of hybrid encryption/decryption algorithms.

5.1 Hidden Volume

Hidden volumes are a steganographic feature \([101]\). It allows a second, "hidden", volume to be created within the free space of another "container" volume. Once a hidden volume has been created inside another volume, the user can store sensitive
information in the hidden volume. Following, Figure 5.2 and Figure 5.3 show disk space before and after creating hidden volume.

<table>
<thead>
<tr>
<th>Std Volume</th>
<th>Occupied space</th>
<th>Free space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.2 before creating hidden volume

<table>
<thead>
<tr>
<th>Std Volume</th>
<th>Header of hidden volume</th>
<th>Occupied space</th>
<th>Hidden Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.3 after creating hidden volume

There are various ways to create hidden volume. The present research work uses DiskPart command[^102] to create and hide the partition. The DiskTrust technique is Windows based and can run on Windows Vista, Windows XP, and the Windows 7 family. DiskPart is a text-mode command interpreter on these platforms. This tool enables management of objects (disks, partitions, or volumes) by using scripts or direct input at a command prompt. As DiskPart is a low level command, the user should be very careful while using same[^103]. Diskpart differs from many command-line utilities because it does not operate in a single-line mode. Instead, after a user starts the utility, the commands are read from standard input/output (I/O) The user can direct these commands to any disk, partition, or volume.

Most Diskpart commands operate on a specific target disk, partition, or volume. The targeted object has "focus." Focus simplifies the common configuration task in which the user creates multiple partitions on the same disk. An object is put into focus by the select command. All commands except for list, help, rem, exit and help require focus.

The select command is used to explicitly change the focus. To implicitly change the focus, the user can use a command such as create. The user must set the disk focus, to manage a basic disk. On basic disks, the partition focus and volume focus are the same.
Diskpart supports script operations. To initiate a Diskpart script, use the diskpart/s script.txt command. By default, Diskpart can quit command processing and return an error code if there is a problem in the script. To continue to run a script in this scenario, include the noerr parameter on the command. This parameter allows the user to use a single script to delete all partitions on all data drives, regardless of the total number of drives. However, not all commands support the noerr parameter.

The following list describes the error codes for Diskpart:

1) 0 - No error occurred. The entire script ran without failure.
2) 1 - A fatal exception occurred. There may be a serious problem.
3) 2 - The arguments specified on a Diskpart command line were incorrect.
4) 3 - Diskpart was unable to open the specified script or output file.
5) 4 - One of the services Diskpart uses returned a failure.
6) 5 - A command syntax error occurred. The script failed because an object was improperly selected or was invalid for use with that command.

To create volume and make it hidden the current work has used Diskpart command. After the user runs Diskpart, the Diskpart version and current computer name are displayed. To create the volume and make it hidden, the following algorithm has been used. The algorithm and the input and output are mentioned below.

Input: Disk number, partition size, partition name (alphabet)
Output: The desired partition is created and hidden from view.

The steps to create hidden volume:

**Procedure 5.1 Hidden Volume Creation**

1) DiskPart Enter
2) Check out the disk specification using one of following command
   DiskPart> list disk/ list partition/ list volume
3) Create new volume
   DiskPart> create partition logical[size=n] [offset=n] [noerr]

This command creates a logical drive in the extended partition. After the partition has been created, the focus automatically shifts to the new logical drive.
size = n: The size of the logical drive is in megabytes (MB). If no size is given, then the partition continues until there is no more free space in the current region.

offset = n: Applies to master boot record (MBR) disks only. Specifies the byte offset at which to create the logical drive. The offset is cylinder snapped; that is, the offset rounds up to completely fill whatever cylinder size is used. If no offset is given, then the partition is placed in the first disk extent that is large enough to hold it. The partition is at least as long in bytes as the number specified by size = n. If you specify a size for the logical drive, it must be smaller than the extended partition.

Noerr: For scripting only. When an error is encountered, specifies that DiskPart continues to process commands as if the error did not occur. Without the noerr parameter, an error causes DiskPart to exit with an error code.

4) Select the created volume.

    DiskPart> select partition = [{n|d}]

Selects the specified partition and gives it focus. If the user does not specify a partition, the select command lists the current partition with focus. The user can view the numbers of all partitions on the current disk by using the list partition command.

N : Specifies the number of the partition to receive the focus.

D : Specifies the drive letter or mount point path of the partition to receive the focus.

5) Make the created volume invisible/ hidden.

    H :> attributes volume [{set | clear}] [{hidden | readonly | nodefaultdriveletter | shadowcopy}] [noerr]

Volume: Displays the attributes of the selected volume.
Set : Sets the specified attribute (hidden, read-only, nodefaultdriveletter, or shadowcopy volume) on the selected volume.
Clear : Clears the specified attribute (hidden, read-only, nodefaultdriveletter, or shadowcopy volume) from the selected volume.
Hidden : Specifies that the volume is hidden.
Readonly : Specifies that the volume is read-only.
Nodefaultdriveletter : Specifies that the volume does not receive a drive letter by default.
Shadowcopy : Specifies that the volume is a shadow copy volume.
Noerr : For scripting only. When an error is encountered, DiskPart continues to process commands as if the error did not occur. Without the Noerr parameter, an error causes DiskPart to exit with an error code.

The java code has been written to create and hide volume using Diskpart. The successful hidden volume is created on disk with suitable drive letter, eg. K: as shown in the results in Figure 5.4. The creation of hidden or secure volume avoids exposure of the secure data.

![Fig. 5.4 The Hidden Volume creation](image)

5.2 Authentication
The above task creates secure partition on the hard drive. To avoid data exposure the created partition is made hidden. To make the partition visible user has to prove authentication. The current password authentication schemes are not strong enough. Anyone can crack the password by using keyloggers or sniffers\(^{104}\). So, there is a need to develop a completely secure password authentication scheme which is not crackable.

DiskTrust implements a new password authentication scheme which is dynamic in nature. Every time when the user wants to log in a new password\(^{105}\) is generated and
mailed to the user. The algorithm used for password generation and the required
Input, Output are as follows:
Input: array of characters
Output: random password string of min size 8 and max size 11

Procedure 5.2 Dynamic Password Authentication
1. Create a dynamic array which will contain all the characters (digits, alphabets, special characters, space) which will be used in the password.
2. Use java util function random to generate the length of password. It has to be between 8 and 11 (max and min length is fixed initially).
3. Instantiate a String buffer, which is used to create the password.
4. Start for loop (i = 0 to length of password (from step 2)).
   Use java util function random to generate an index x, array value at position x (from step 1) will be the i\textsuperscript{th} character in the string buffer. Append the character from above sub-step to the string buffer. End for loop.
5. Convert the string buffer to a String using toString() function.

The Figure 5.5 is screenshot of user interface for login, password authentication and new password generation. This appears when the user is identified to be already existing in the system.

Fig. 5.5 The Primary Authentication with Login and Dynamic password
The second level authentication is implemented while reading the data stored on the secure volume. The second level authentication is implemented using pattern matching. The algorithm used and the required Input, Output is as follows:

Input: panel with a 25 grid image.
Output: 3 grid numbers corresponding to 3 clicks on the panel.

---

**Procedure 5.3 Pattern Matching Authentication**

1. Create an image of 25 grids and note down the axis values of each grid.
2. Create a panel and add the above image to it.
3. Instantiate a mouse event listener on the panel.
4. Handle the mouse event only for 3 times (3 clicks).
   
   Start for loop (i-> 0 to 3)
   
   a) Extract X and Y axis values of the clicked point using getX() and getY() functions.
   b) Compare the axis values of the current click with the axis values previously noted.
   c) Identify the grid number by analyzing the axis values from the current click and previously noted values of the 25 grids.
   d) Store the grid numbers in a file corresponding to the username.
   
   End for loop.
5. Close the panel and free all allocated panel resources.

To view the decrypted files the user must specify the correct three points revealing the pattern as shown in Figure 5.6.

---

Fig. 5.6 The Second Level Authentication using pattern Matching
Once the second level authentication is done user can retrieve data from the secured partition. The data which the user stores on the secured partition is seamlessly encrypted/decrypted while accessing/storing respectively, from the HDD. This seamless encryption/decryption of data is called on-the-fly encryption which is the major beauty of PDE/FDE disk security techniques \cite{36,37} has implemented disk security with FDE while the current work proposed disk security using PDE. The research work mainly focused on the strong and highly secure encryption/decryption algorithm. This strategy is implemented using standard AES algorithm with different key size (AES-128, 192,256) as mentioned in chapter 3, modified AES with S-Box implementation and hybrid AES-DES are discussed follows.

5.3 Advanced Encryption Standard (AES) Substitution-Box (S-Box)

The thesis work proposes AES(128) implementation with variation in S-Box implementation. S-Box (Substitution-box) is a basic element of symmetric key algorithms which performs substitution \cite{57}. In block ciphers, they are typically used to hide the relationship between the key and the cipher text \cite{58}. It makes the resulting cipher text satisfy Shanon’s confusion. The original AES implementation is modified in S-Box design. In many cases, the S-Boxes are carefully chosen to avoid cryptanalysis.

5.3.1 Algebraic Preliminaries and S-Box Construction

For the study of algebraic construction of the S-Box a theorem is stated here. \cite{107}.

5.3.1.1 Theorem 1

Let \( p \) be a non-zero element of a principle ideal domain \( R \) then, \( R/(p) \) will be a field if and only if, \( p \) is irreducible. According to this theorem, for a prime \( p \) Galios field \( GF( p^n ) \) is constructed by using a generating polynomial \( m(x) \) of degree \( n \) taking

\[
GF( p^n ) = \frac{GF(p)(x)^n}{m(x)} \quad (5.1)
\]

In the AES-128 (Rijndael) algorithm, the irreducible polynomial \( x^8 + x^4 + x^3 + x + 1 \) is used to generate the underlying field \( GF(2^8) \). All bytes \( b \) in Rijndael are interpreted as elements of this field represented by a polynomial \( a_1 + a_2 x^1 + a_3 x^2 + a_4 x^3 + a_5 x^4 + a_6 \).
\[ x^5 + a_7 x^6 + a_8 x^7 \] where, each bit \( a_i \in \text{GF}(2) \) and \( b \in \text{GF}(2^8) \). In this field, addition \( \oplus \) and multiplication are defined by the XOR operation and polynomial multiplication respectively.

An S-Box is a transformation \( s : \text{GF}(p^n) \rightarrow \text{GF}(p^n) \). In AES the S-Box \( \sigma : \text{GF}(2^8) \rightarrow \text{GF}(2^8) \) is constructed by substituting each element with its inverse and applying a suitable affine transformation \( \sigma : X \rightarrow AX^{-1} + b \) where, \( A \in \text{GL}_8(2) \), the general linear group of degree 8 over \( \text{GF}(2) \) and \( b \in \text{GF}(2^8) \). Both of these \( A \) and \( b \) are fixed in AES:

\[
A = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
\quad \text{and} \quad
b = \begin{bmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
1 \\
1 \\
0 \\
\end{bmatrix}
\]

The catalyst for this S-Box design is to be resistant to differential, linear cryptanalysis and interpolation attacks. The gist design is a simple transformation \( x \rightarrow x^{-1} \) in \( \text{GF}(2^8) \), this mapping has a simple algebraic expression. Nevertheless, the simplicity itself makes it vulnerable to attacks like the interpolation attack. Hence, it is combined with a suitable affine transformation: \( x \rightarrow Ax^{-1} + b \). The affine mapping is so selected that, it has a very simple description, but a muddle algebraic expression. If combined with the ‘inverse’ mapping, it can be seen as modular polynomial multiplication followed by addition,

\[
b(x) = (x^7 + x^6 + x^2 + x) + a(x) (x^7 + x^6 + x^5 + x^4 + 1) \mod (x^8 + 1) \ldots (5.2)
\]

The purpose of the constant translation vector \( b \) is to ensure that \( \exists \) no fixed and conjugate fixed points (i.e. \( \exists \) no \( x \in \text{GF}(2^8) \) such that \( \sigma(x) = x \) or \( \sigma(x) = x \)) in the S-Box.
5.3.2 Characteristics of a Good S-Box and Bias Parameters

To analyze the characteristics of an efficient S-Box $\sigma: GF(p^n) \rightarrow GF(p^n)$, it is realized as vectorial Boolean function $\sigma(x) = (\sigma_1(x), \sigma_2(x), ..., \sigma_n(x))$ where, each $\sigma_i$ is a Boolean function of the Boolean variables $x_1, x_2, ..., x_n$. Characteristics of the good S-Box are:

1) It has to be balanced number of zeroes and ones.
2) It has to satisfy propagation criterion, is evaluated using Boolean function (PCB)
3) It has to satisfy correlation immunity criterion, is evaluated using Boolean function (CIB)
4) It has to have input/output bit-to-bit entropy ($H$) = 1.
5) Its nonlinearity ($N$) has to be = 120.

Each of these criteria's are explained in the following subsection. However, most of these are satisfied only for an ideal case and not in practical level. So, biases from each of these criterions are derived.

5.3.2.1 Balancedness Property

This property states that each of the Boolean functions of the S-Box should be balanced, i.e., the number of ones and zeros in the truth table of the Boolean function must be equal.

5.3.2.2 Propagation Criterion

A Boolean function is said to satisfy propagation criterion of degree $k$ and order $m$, if any function obtained by keeping $m$ input bits fixed $f(x)$ changes with probability half, whenever $i (1 \leq i \leq k)$ bits of $x$ are complemented.

Mathematically, by fixing $m$ number of bits $\binom{n}{m} 2^m$ set of functions $g$ are obtained from $f$, let it be denoted by $F$. Let $\alpha \in GF(2^n) : W(\alpha) \in [1, k]$ then a function is said to satisfy the propagation criterion of degree $k$ and order $m$, if for each $g \in F$, $g(x) \oplus g(x \oplus \alpha)$ is balanced.
The propagation criterion is the measure of randomness of the differences in output pairs to the input pairs. This is a very important criterion as the bias of the distribution of the differences of the output pairs and the input pairs is utilized in the differential cryptanalysis of the conventional ciphers.

5.3.2.2.1 Definition 1
Propagation criterion bias of a Boolean function of degree $k$ and order $m$ is defined by [108]:

$$PCB\sigma_i(k, m) = \max_{\alpha \in A} \left| \sum_{x \in F_2^n} (g(x) \oplus g(x \oplus \alpha)) - 2^{n-m-k} \right|$$

(5.3)

where $A = \{\alpha \in GF(2^n) : w(\alpha) \in [1,k]\}$.

For the S-Box: $PCB\sigma(k, m) = \max PCB\sigma_i(k, m)$

The propagation criterion of degree one and order zero is very well known criterion, termed as the strict avalanche criterion (SAC). The criterion is satisfied, if whenever a single input bit is complemented, each of the output bits changes with a probability $\frac{1}{2}$.

5.3.2.3 Correlation Immunity Criterion
A Boolean function is to satisfy a correlation immune of order $m$, if it is statistically independent of combination of any $m$ input bits [109]. Mathematically, if $m$ inputs bits are fixed then the functions $g$ obtained from Boolean function $f$ must satisfy:

$$W( g) = W(f) / 2^m$$

(5.4)

where $W(f)$ denotes the Hamming weight of a Boolean function given by the number of $x$ for which the function attains a nonzero value.

5.3.2.3.1 Definition 2
The correlation immunity bias of order $m$ for a Boolean function is defined by [107]:

$$CIBf( m) = \max_{F \in A} | 2^m \times W(g) - W(f)|$$

(5.5)

The correlation immunity bias of S-Box is given by

$$CIB\sigma_i( m) = \max_{I \in [1, s]} CIB\sigma_i( m)$$

(5.6)
5.3.2.4 Input/output Bit-to-Bit Entropy
This parameter represents the amount of information about the value of input bit, if the value of the output bit is known. The entropy of a single output function is given by[110]:

\[ H(P_i) = P_i \log_2(1/P_i) + (1 - P_i) \log_2(1-P_i) \]  \hspace{1cm} \text{(5.7)}

Where \( P_i \) is function of 1’s in the output column of truth table.

5.3.2.4.1 Definition 3
The (i, j)th Input/output bit-to-bit entropy \( H(x_i / \sigma_i(x)) \) is computed and the parameter is defined by

\[ H = \min_{i,j} H( x_i / \sigma_i(x) ) \]

5.3.2.5 Nonlinearity
An affine Boolean function does not provide an effective confusion. To overcome this, functions which are as far as possible from being an affine function are needed. The effectiveness of these functions is measured by a parameter called nonlinearity [111].

5.3.2.5.1 Definition 4
Nonlinearity of a Boolean function is measured by the Hamming distance to the set of affine functions [107],

\[ N(f) = 2^n - \frac{1}{2} \times \max F(w) \]  \hspace{1cm} \text{(5.8)}

where \( F \) is the Walsh transformation of \( f \).

For S-Box of \( N = \min_i (N(f_i)) \) \hspace{1cm} i \in [l, n] \]

For good cryptographic properties of the S-Box, these parameters should have the values: \( H = 1 \), PCB(1,0) = 0, PCB(1,1) = 0, CIB(1) = 0 and nonlinearity, \( N = 120 \)[113]. However, values of these parameters for the AES S-Box are: \( H = 0.9887 \), PCB(1,0) = 16, PCB(1,1) = 20, CIB(1) = 16 and nonlinearity \( N = 112 \). The values of these bias parameters are used to analyze the effect of changes in the design components of the AES S-Box on its cryptographic properties. Different possible variations on the S-Box components and their affects have been discussed in the next section.

To meet these criteria an invertible S-Box is constructed by composing two transformations: Inverse and Affine transformation

\[ x = A x^{-1} + b, \hspace{0.5cm} \text{where } x \in GF(2^8) \]  \hspace{1cm} \text{(5.9)}

\[ A \in GL_8(2) \hspace{0.5cm} \text{and } b \in GF(2^8) \]
The variation in the S-Box design component can be brought about by changing the affine matrix $A$. The affine matrix $A \in GL_8(2)$, the general linear group of degree 8 over GF(2) and the order of this group is:

$$\sum_{k=0}^{7} (2^8 - 2^k) \approx 5.3481 \times 10^{18} \ldots \ldots \ldots \ldots \ldots (5.10)$$

### 5.3.3 Results

There are the numbers of matrix affine $A$ can be generated, and used for changing the S-Box.

The S-Boxes are carefully chosen to avoid cryptanalysis. The encryption algorithm are experimentally implemented as well as simulated on MATLAB.

The Table 5.1 shows the observed bias parameter values obtained after changing in the affine matrix.

#### Table 5.1 Proposed S box vs Std. S Box

<table>
<thead>
<tr>
<th>SL No</th>
<th>Generating Polynomial</th>
<th>Proposed S-box</th>
<th>Standard S-box</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$H$</td>
<td>PCB</td>
</tr>
<tr>
<td>1</td>
<td>$x^8 + x^4 + x^3 + x + 1$</td>
<td>0.989</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>$x^8 + x^7 + x^5 + x^4 + 1$</td>
<td>0.989</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>$x^8 + x^6 + x^5 + x^4 + 1$</td>
<td>0.991</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>$x^8 + x^5 + x^3 + x^2 + 1$</td>
<td>0.991</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>$x^8 + x^4 + x^3 + x^2 + 1$</td>
<td>0.991</td>
<td>16</td>
</tr>
</tbody>
</table>

The observed values are graphically plotted as shown in Figure 5.7 Input/output bit-to-bit entropy ($H$), Propagation Criterion is evaluated using Boolean function(PCB), Correlation Immunity criterion is evaluated using Boolean function(CIB) these three parameters for different polynomial are plotted.
It is observed that, the change in affine matrix improves the biased properties to 0.02%.

5.4 Hybrid AES-DES

As the speed of processors and knowledge about cryptanalysis goes on growing day by day, the original encryption methods prove to be insufficient for better security\footnote{58}. The AES and DES algorithms discussed in (Chapter 3) are powerful symmetric ciphers. However both AES as well as DES have been compromised in the past. Attackers have succeeded in estimating the keys and encryption steps in these algorithms. By exploiting the key steps and vulnerabilities in these algorithms it is possible to break through AES and DES\footnote{59}.

The problem with AES, most extensively used encryption, is that it uses many multivariate equations which are linear in nature \footnote{60}. Thus it can be broken using algebraic cryptanalysis. This provides a serious threat as AES was considered to be unbreakable and thus it was used in many encryption systems. The DES predecessor
of AES has problem of multivariate linear equation and speed. This prompted the development of a hybrid AES-DES algorithm. This new algorithm combines the AES algorithm with the DES Feistel structure. This improved algorithm contains the features of AES along with DES algorithm. The use of AES and DES together highly increases the number of permutation, combinations used in the structure. A number of keys required for both the algorithms further increase the encryption strength of the algorithm.

According to literature [60] has implemented hybrid AES-DES image encryption. The other few designs are proposed [58, 59] which combines both the algorithm together. It carries the basic flaws of both the algorithms together. The literature [59] had proposed hybrid AES-DES. In [59] the authors had implemented AES inside the DES Feistel structure. This makes the design more complicated and time consuming.

This section presents a new hybrid design 128 bit key AES-DES algorithm, as security enrichment. Standard AES-128 is implemented using 10-rounds, while standard DES is implemented using 18-rounds. A proposed hybrid algorithm is implemented in 10-rounds. Instead of implementing AES transformation in intermediate rounds [58, 59, 60]. The first and last round is from AES cipher and the intermediate 8-rounds are implemented using DES’s Feistel structure. The implementation details are as follows,

5.4.1 Algorithm Implementation

As described in [48], each AES data block consists of 128 bits (16 bytes) of data. During the encryption and decryption processes these 16 bytes will create a changeable (4*4) array called the state array. This state array is processed by first round which is substitution step in AES. The complete architectural overview of proposed algorithm is as shown in Figure 5.8
5.4.2 First Step AES SubBytes Transformation

The first step in proposed algorithm is AES SubBytes transformation. The SubBytes transformation uses AES substitution table (S-Box) to provide non-linear substitution. To meet the non-linearity requirement, the S-Box is constructed by combining two transformations which include an inverse function and an invertible affine transformation. In the first transformation, the elements of GF(2^8) are represented as polynomials which have degrees less than eight, with coefficients in GF(2). Multiplications are carried out modulo the irreducible polynomial \( x^8 + x^4 + x^3 + x + 1 \), and the multiplicative inverses are defined accordingly. The element \( \{00\} \) is mapped to itself. As the first transformation has a simple algebraic expression, which may lead to attacks such as algebraic attacks, it is essential to apply second transformation is an affine transformation (over GF (2)), which can be expressed in following matrix:
The substitution table is as per discussed in chapter 3, with all elements in hexadecimal form. For example, if the input to SubBytes transformation is \{6f\}, the output would be \{a8\}.

### 5.4.3 Intermediate 8-rounds: DES Feistel Cipher

In cryptography, a Feistel cipher is a symmetric structure used in the construction of block ciphers, named after the German-born physicist and cryptographer Horst Feistel who did pioneering research while working for IBM (USA); it is also commonly known as a Feistel network [72]. A large proportion of block ciphers use the scheme, including the Data Encryption Standard (DES). The Feistel structure has the advantage that encryption and decryption operations are very similar, even identical in some cases, requiring only a reversal of the key schedule [73]. Therefore the size of the code or circuitry required to implement such a cipher is nearly halved. A Feistel network is an iterated cipher with an internal function called a round function. The working principle is as follows:

Let be the round function and let $K_0, K_1, \ldots K_n$ be the sub-keys for the rounds 0, 1, …n respectively. Then the basic operation is as follows:

Split the plaintext block into two equal pieces, $(L_0, R_0)$

For each round $I = 0, 1, \ldots n$ compute

\[
L_{i+1} = R_i \quad \text{............................................}(5.11)
\]

\[
R_{i+1} = L_i \oplus F(R_i, K_i) \quad \text{............................................}(5.12)
\]

Then the ciphertext is $(R_{n+1}, L_{n+1})$
Decryption of a ciphertext \((R_{n+1}, L_{n+1})\) is accomplished by computing for \(I = n, n-1, \ldots, 0\)

\[ R_i = L_{i+1} \quad \ldots \quad (5.13) \]

\[ L_i = R_{i+1} \oplus F(L_{i+1}, K_i) \quad \ldots \quad (5.14) \]

Then \((L_0, R_0)\) is the plaintext again.

One advantage of the Feistel model compared to a substitution-permutation network is that the round function \(F\) does not have to be invertible.

Figure 5.9, Figure 5.10 illustrates encryption, decryption process respectively, for the proposed algorithm. Note the reversal of the subkey order for decryption; this is the only difference between encryption and decryption.

Fig. 5.9 Feistel Encryption model for Hybrid AES-DES
The encryption process starts with AES sub-byte transformation. The output of substitution step would be divided into two parts, namely 64-bits right part and 64-bits left part. According Feistel cipher equation 5.12, the right part of \( i \)th round will become left part of for \( i+1 \) round. The right part of the \( i+1 \) round is calculated by XORing left part with \( F(R_i, K_i) \). The function \( F \) is discussed in following section.

![Feistel Decryption model for Hybrid AES-DES](image)

In decryption process the cipher text is processed by AES inverse substitution transformation. The output of substitution step is divided into two parts, namely 64-bits left part and 64-bits left part. According Feistel cipher equation 5.14, the right part of \( i+1 \) round will become left part of for \( i \)th round. The right part of the \( i \)th round is calculated XORing left part with \( F(R_i+1, K_i+1) \). Here \( F(R_i, K_i) \) is a function performed on \( R_i+1 \), key \( K_i+1 \). The function \( F \) is discussed in following section.
5.4.3.1 Function F in Feistel Cipher

The Function F is graphically shown in Figure 5.11. In F function 64-bits Right part is expanded, XORed with Key, substituted using DES S-Box and finally XORed with 64-bit left part as per Feistel equation 5.12. Each of this operation is explained as follows:

5.4.3.1.1 Expansion Function
The 64-bit right half is expanded to 128-bits using expansion permutation function as shown in Figure 5.12. Initially 64-bits are expanded to 80-bits by dividing 64-bits into 8-subhalf of 8-bits each. In this eight bit the last and first bit is appended from previous and next 8-bit subhalf in round-robin fashion as shown. With the generated 80-bits, DES 48-bits key is appended, to generate 128-bits.
5.4.3.1.1 XOR Operation

The XOR operation performed on 128-bits generated from-bits last step (80-bits expanded with 64-bits DES key) as shown in Figure 5.13.
The purpose of previous expansion was to make the right half 128-bits. The 128-bits from expansion permutation rounds are XORed with AES 128-bits key.

5.4.3.1.2 Substitution

The 128-bits are converted to 64-bits using DES S-Box as shown in Figure 5.14. The operation uses sixteen S-Boxes. Each box takes eight bits as input and produces 4-bits as output. The operation of individual S-Box is discussed below and shown in Figure 5.15.
Given an 8-bit input, the 4-bit output is found by selecting the row using the outer four bits (the first and last bits), and the column using the inner four bits. For example, an input "00011011" has outer bits "0011" and inner bits "0110"; the corresponding output would be "0010".

5.4.3.1.3 Permutation

![Fig. 5.16 Permutation](image)

Figure 5.16 Shows the P-Box permutation which is nothing but the scrambling of 64-bits inputs to produce 64-bits scrambled output.

5.4.4 Final Step: Add Round Key

The final step of the proposed algorithm is AES AddRoundKey. The AddRoundKey transformation\textsuperscript{[67]} adds a round key to the state by using a simple bitwise XOR operation. Each round key is used in this transformation derived from the secret key employing the AES-key schedule is described earlier. Each round key has the same size as the state. Figure 5.17 show the effect of the AddRound Key operation on the state.
The encryption, decryption routines are illustrated below:

**Procedure 5.7** Cipher(byte in[4 * Nb], byte out[4 * Nb], word w[Nb * (Nr + 1)])

1: byte s[4, Nb]
2: s = in
3: SubBytes(s)
4: for round = 1 to Nr-1 do
   5: Des FesiteloOp(s)
   6: end for
7: Add(s, w[round * Nb, (round + 1) * Nb -1])
8: out = s

**InvCipher**

**Procedure 5.8** InvCipher(byte in[4 * Nb], byte out[4 * Nb], word w[Nb * (Nr + 1)])

1: byte s[4, Nb]
2: s = in
3: InvSubBytes(s)
4: for round = Nr-1 to 1 do
   5: Des Inv of FesiteloOp(s)
   6: end for
7: Inv of AddRound Key(s, w[round * Nb, (round + 1) * Nb -1])
8: out = s

**5.4.5 Results**

The proposed pseudo code for the hybrid AES-DES algorithm is implemented in Java. For the experiment, the current work has used a laptop PentiumV 2.4 GHz CPU, in which performance data is collected. The performance is analyzed for different data
files. The encryption/decryption time required for specific amount of data is observed. The encryption/decryption time is used to calculate the throughput of an encryption scheme. It indicates the speed of encryption. The throughput of the encryption scheme is calculated as the total plaintext in bytes encrypted divided by the encryption time \(^{122}\). The performance is shown in Table 5.2.

<table>
<thead>
<tr>
<th>Input size in (MB)</th>
<th>Proposed Hybrid AES-DES(128) Encryption</th>
<th>Proposed Hybrid AES-DES(128) Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.49</td>
<td>0.41</td>
</tr>
<tr>
<td>50</td>
<td>0.89</td>
<td>0.87</td>
</tr>
<tr>
<td>100</td>
<td>1.78</td>
<td>1.75</td>
</tr>
<tr>
<td>128</td>
<td>2.44</td>
<td>2.30</td>
</tr>
<tr>
<td>200</td>
<td>3.87</td>
<td>3.70</td>
</tr>
<tr>
<td>256</td>
<td>4.8</td>
<td>4.5</td>
</tr>
<tr>
<td>512</td>
<td>8.66</td>
<td>8.5</td>
</tr>
<tr>
<td>700</td>
<td>10.6</td>
<td>10.51</td>
</tr>
<tr>
<td>850</td>
<td>12.2</td>
<td>12.1</td>
</tr>
<tr>
<td>1000</td>
<td>13.75</td>
<td>13.45</td>
</tr>
<tr>
<td>Average</td>
<td>5.94</td>
<td>5.58</td>
</tr>
<tr>
<td>Throughput(MB/S)</td>
<td>53.94</td>
<td>54.33</td>
</tr>
</tbody>
</table>

The encryption, decryption performance is graphically shown in Figure 5.18. The x-axis shows the time required in seconds. The y-axis shows the size of the data files in MB. The calculated throughput and average from the observed values are also shown. The observations are

1. The proposed algorithm is used to encrypt the data files from 1MB to 1GB.
2. The algorithm is showing acceptable results.
3. These results are compared with standard AES (128-bit key) algorithm in chapter6.
5.5 Cryptanalysis

AES and DES has a simple, elegant and efficient design. It was designed to resist all well known attacks, which include two classic cryptanalytic method: differential cryptanalysis\cite{113, 114, 115} and linear cryptanalysis\cite{116, 117, 118}. Linear cryptanalysis and differential cryptanalysis are two powerful techniques in block cipher cryptanalysis. These two attacks need to be taken into account while designing any new cipher. Resistance against differential cryptanalysis and linear cryptanalysis are most important criteria in the design of AES, DES. Here the work has evaluated both the cryptanalysis for the analyzing the proposed hybrid AES-DES algorithm.

Several recent papers have dealt with provable security against differential and linear cryptanalysis for block ciphers based on the substitution-permutation network (SPN) structure \cite{114, 116, 117, 118}. Most of these results apply directly to the Advanced
Encryption Standard (AES)\textsuperscript{[118]}. Demonstrating provable security against differential/linear cryptanalysis involves proving that the maximum expected differential/linear probability (MEDP / MELP) is sufficiently small over T core rounds—this is because the data complexity of the attack (the number of plaintext-ciphertext pairs required) is proportional to the inverse of the MEDP/MELP. Since in general it is difficult to compute the MEDP/MELP exactly, researchers have focused on bounds. A series of progressively smaller upper bounds has been obtained for the AES; the best of these is $1.161 \times 2^{-111}$ (MEDP) / $1.064 \times 2^{-106}$ (MELP) for $T \geq 4$\textsuperscript{[114]}. Many such bounds are based on careful examination of the case $T = 2$\textsuperscript{[120]}. Based on these assumptions the analysis is done for proposed algorithm.

5.5.1 Differential and Linear Cryptanalysis
Differential cryptanalysis, proposed by Biham and Shamir\textsuperscript{[118]} in 1990, is chosen plaintext attack. The basic idea is to analyze the relations between the differences of plaintext pairs and the differences of the resultant ciphertext pairs, which are the result of encryptions under the same secret key. These differences are used to assign probabilities to the possible keys and identify most probable one.

According to \textsuperscript{[119]} the MEDP for standard AES(128) is $1.161 \times 2^{-111}$, the proposed hybrid AES-DES has 194-bits key(AES-128 + DES-56). Therefore the MEDP bound for the proposed hybrid AES-DES would definitely be more than standard AES(128)MEDP.

In linear cryptanalysis is a known plaintext attack and is discovered by Masuti\textsuperscript{[118]} in 1993. In linear cryptanalysis, the adversary studies the linear approximation of the plaintext, ciphertext and key bits to recover the key. The MELP for standard AES(128) is $1.064 \times 2^{-106}$, the proposed hybrid AES-DES has 194-bits key(AES-128 + DES-56). Therefore the MELP bound for the proposed hybrid AES-DES would definitely be more than standard AES(128)MELP.

The observation is, With this calculation both linear and differential cryptanalysis are quite complex for the proposed algorithm hybrid AES-DES (128+56) as compare to standard AES (128).
5.6 Summary

In this chapter DiskTrust: disk security model is experimentally implemented using Java programming. It implemented improved AES with variations in S-Box implementation. The chapter also implemented novel hybrid AES-DES architecture. All the algorithms are tested on different data sets.