Chapter 2  System Design

2.1  Novel Modification to the Algorithm

As we have seen in Hill cipher decryption, it requires the inverse of a matrix. So while one problem arises that is: Inverse of the matrix doesn’t always exist. Then if the matrix is not invertible then encrypted text cannot be decrypted. In order to overcome this problem author suggests the use of self repetitive matrix. This matrix if multiplied with itself for a given mod value (i.e. mod value of the matrix I taken after every multiplication) will eventually result in an identity matrix after N multiplications. So, after N+ 1 multiplication the matrix will repeat itself. Hence, it derives its name i.e. self repetitive matrix. It should be non singular square matrix.

Note: Historically, there are already many available methods, like self invertible matrix etc. But this method as suggested by the author is both easy to compute and simpler to implement.

2.1.1  Modular Arithmetic: A brief Analysis

The analysis presented here for generation of self repetitive matrix is valid for matrix of positive integers that are the residues of modulo arithmetic on a prime number. So in analysis the arithmetic operations presented here are addition, subtraction, Unary operation, Multiplication and division.

The modulo operator have the following properties:
1. \( a \equiv b \pmod{p} \) if \( n \mid (a-b) \)
2. \( a \equiv b \pmod{p} \) implies \( b \equiv a \pmod{p} \)
3. \( a \equiv b \pmod{p} \) and \( b \equiv a \pmod{p} \) imply \( a \equiv c \pmod{p} \)

The modulo arithmetic have the following properties:

Let \( z = \{0,1,\ldots,p-1\} \), the set residues (mod p) operator maps all integers into the set \( z \). If modular arithmetic is performed within the set \( z \),

The following equations present the arithmetic identities:

1. Addition: \( (a + b) \mod{p} = [(a \mod{p}) + (b \mod{p})] \mod{p} \)
2. Subtraction: \( (a - b) \mod{p} = [(a \mod{p}) - (b \mod{p})] \mod{p} \)
3. Multiplication: \( (a \times b) \mod{p} = [(a \mod{p}) \times (b \mod{p})] \mod{p} \)
4. Negation: \( -a \mod{p} = p - (a \mod{p}) \)
5. Division: \( (a/b) \mod{p} = c \) when \( a = (c \times b) \mod{p} \)
6. Multiplicative inverse: \((a-1) = c\) if there exists \((c*z) \mod p = 1\)

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Property</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Commutative Law</td>
<td>((w + x) \mod p = (x + w) \mod p)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>((w<em>x) \mod p = (x</em>w) \mod p)</td>
</tr>
<tr>
<td>2</td>
<td>Associative Law</td>
<td>([(w + x) + y] \mod p = [w + (x + y)] \mod p)</td>
</tr>
<tr>
<td>3</td>
<td>Distributive Law</td>
<td>([w*(x + y)] \mod p = [w<em>x + w</em>y] \mod p)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>([w*(x<em>y)] \mod p = [(w</em>x) \mod p]<em>[(w</em>y) \mod p] \mod p)</td>
</tr>
<tr>
<td>4</td>
<td>Identities</td>
<td>((0+a) \mod p = a \mod p) and ((1*a) \mod p = a \mod p)</td>
</tr>
<tr>
<td>5</td>
<td>Inverse</td>
<td>For each (X) belongs to (zp), there exists (y) such that ((x + y) \mod p = 0) then (y = -x)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For each (X) belongs to (zp), there exists (y) such that ((x*y) \mod p = 1)</td>
</tr>
</tbody>
</table>

Table 2.1 Exhibits the properties of modulo arithmetic

2.1.2 Complex modular numbers

Used while calculating negative or non-existent square roots.

Say, for example, for mod 7

\[\sqrt{1} = 1;\] \[\sqrt{2} = 4;\] \[\sqrt{4} = 2;\]

But square root of 3, 5, and 6 doesn’t exist

So \(\sqrt{6} = \sqrt{-1} \sqrt{-6} = j\)

where \(j\) stand for square of -1 in modular arithmetic

Similarly, \(\sqrt{5} = 4j \& \sqrt{3} = 2j\)

Extensive no of simulation exercises were carried out to understand the properties of modular arithmetic.

2.1.3 Generation of a self repetitive Matrix A for a Given N

The initial conditions for the existence of a self repetitive matrix are:

1. The matrix should be square.
2. It should be non-singular.

It is trying to find out the value of \(N\) (the value where the matrix becomes a identity matrix) through the method of brute force may not be the best idea always; because the matrix is of dimension greater than 5*5 and with mod index (i.e.) greater
than 91 then the brute force technique might take very long time and N value may be in the range of millions. A normal Pentium 4 machine might hang if asked to do the computations for 15*15 matrixes or more. Hence, it would be comfortable to know the value of N and then generate a random matrix accordingly. This can be done as follows:

1. First a diagonal matrix A is chosen, and then the values powers of each individual element when they reach unity is calculated and denoted as \( n_1, n_2, n_3, \ldots \). Now LCM of these values is taken to given the value of N.
2. Now the next step is generate a random square matrix whose N value is same as the N calculated in the previous step.
3. Pick up any random invertible square matrix B
4. Generate \( C = B^{-1}AB \)
5. The N value of \( C \) is also N

Mathematical proof:
\[
(B^{-1}AB)^N = (B^{-1})^N * (A)^N * (B)^N
\]
\[
A^N = I \text{ as calculated before as it is a diagonal matrix and N is the LCM of all elements}
\]
\[
(B^{-1}B) * (B^{-1}B) \ldots \ldots n \text{ times} = I
\]

2.1.4 Another Line of Investigation

In pursuit of finding a many one relationship of N value with matrices led us towards investigating the correlation of N values with \( l \) (lambda) Eigen values. A relationship established like this could easily solve the key distribution problem. The relationship worked out but not conclusively implemented is like this:

NOTE: ALL operations are modular.

Say A is any square matrix and B is the diagonal matrix containing the Eigen values of A. Let \( P(l) \) is the characteristic Eigen equation of the A.

Then, \( l^N - 1 \) will be divisible by \( P(l) \). (N is the n value of A) only real l values

NOTE: These methods shown and proved here have been used to simulate a transmitter and a receiver. Care has been taken to properly compress the code as well. The algorithm and the code include in the implementation section.
2.2 Data Encryption Standard

Data encryption standard is the most widely used method of data encryption using a secret key. There are 72,000,000,000,000,000 (72 quadrillion) or more possible encryption keys that can be used. For each given message, the key is chosen at random from among this enormous number of keys. Like other private key cryptographic methods, both the sender and the receiver must know and use the same private key. It was developed in the 1970s by the National Bureau of Standards with the help of the National Security Agency. Its purpose is to provide a standard method for sensitive commercial and unclassified data. IBM created the first draft of the algorithm, calling it LUCIFER (H. Feistel). DES officially became a standard in November of 1976. Data encryption algorithm has a 64-bit block size and uses a 56-bit key during execution (8 parity bits are stripped of from the full 64-bit key). The DEA can also be used for single user encryption, such as to store files in a hard in encrypted form. National Institute of Standards and Technology (NIST) re-certifies DES every 5 years. DES has been in world wide use for over 20 years, and due to the fact that it is a defined standard that any system implementing DES can communicate with any other system using it. DES is an approved cryptographic algorithm as required by FIPS (Federal Information Processing Standards). FIPS provides a complete description of DES as a mathematical algorithm for encryption and decryption for binary coded information.

2.2.1 DES Encryption

DES is a symmetric, block-cipher algorithm with a key length of 64 bits, and the algorithm operates on successive 64 bit blocks of plain text. Due to symmetric, the same key is used for encryption and decryption, and also uses the same algorithm for encryption and decryption. Initially a transposition is carried out according to a set table (the initial permutation), the 64-bit plaintext block is then split into two 32-bit blocks, and 16 identical operations called rounds are carried out on each half. The two halves are joined back together, and the reverse of the initial permutation is carried out. The purpose of the first transposition is clear; it does not affect the security of the algorithm, but is through for the purpose of allowing plaintext and cipher text to be loaded into 8-bit chip in byte-sized pieces. In any round, only one half of the original
64-bit block is operated on. The rounds alternate between the two halves. One round in DES consists of the following:

2.2.1.1 Key Transformation

The keys reduced from 64-bit to 56-bit by removing every eighth bit, which are sometimes used for error checking. 16 different 48 bit sub-keys are then generated i.e. one for each round. This is achieved by splitting the 56-bit key into the two halves, and then circularly shifting them left by one or two bits, depending on the round. After this, 48 of the bits are selected. Because they are shifted, different groups of key bits are used in each sub key. This process is called compression permutation due to transposition of bits & reduction of the overall size.

2.2.1.2 Expansion Permutation

Whichever half of the block is being operated on undergoes a permutation after key transformation. In this operation, the expansion & the transposition are achieved simultaneously by allowing the first and fourth bits in each block for bit block to appear twice in the output that is the fourth input bit becomes the fifth and the seventh output bit. The expansion permutation achieves 3 things. Those are described below:

1. It increases the size of the half block from 32 to 48 bit, the same number of bit as in compressed key subset, which is important as the next operation is to XOR the two together.
2. It produces a long string of data for the substitution operation that subsequently comprises it.
3. Because in the subsequent substitutions on the first & 4th bits appearing in 2 S-boxes, they affect two substitutions. The effect of this is the dependency of the output on the input bits is that the dependency of the output on the input bits increases rapidly.

2.2.1.3 XOR

XOR operation performed with the appropriate subset key for that round & the resulting 48 block.
2.2.1.4 Substitution

After XOR operation the next operation is to perform substitution on the expanded block. There are 8 substitution boxes called S-boxes. The first S-Box operates on the first 6 bits of the 48 bit expanded block, the second S-box on the next 6 & so on. Each S-box operates from a table of 4 rows & 16 columns; each entry on a table is a 4 bit number. The 6 bit number the S-box takes as input is used to look up the appropriate entry in the table in the following way. The first and the 6 bits combine to form a two bit number corresponding to a row number, the second and fifth bit combine to form a 4 bit number corresponding to a particular column. The net result of the substitution phase is 8, 4 bit blocks that are then combined to form a 32 bit block. It is the non-linear relationship of the S-boxes that really provides DES with its security.

2.2.1.5 Permutation

The 32 bit output of a substitution phase then undergoes a straight forward transposition using a table called P-Box. After all the round has been completed, the two half blocks of 32 bits are recombined to form a 64 bit output. The final permutation is performed on it, and the resulting 64 bit block is the desired DES encrypted cipher text of the input plain text block.

2.2.2 DES Decryption

If one has the correct key decrypting DES is very easy. The decryption algorithm is identical to the encryption algorithm. The only change is to decrypt DES cipher text; the subsets of the keys use in each round are used in reverse.
### 2.2.2.1 The Avalanche Effect

The avalanche effect is evident if, when an input is changed slightly (for example, flipping a single bit) the output changes significantly (e.g., half the output bits flip). In the case of quality block ciphers, such a small change in either the key or the plaintext should cause a drastic change in the ciphertext.

If a block cipher or cryptographic hash function does not exhibit the avalanche effect to a significant degree, then it has poor randomization, and thus a cryptanalyst can make predictions about the input, being given only the output. This may be sufficient to partially or completely break the algorithm. Thus, the avalanche effect is a desirable condition from the point of view of the designer of the cryptographic algorithm or device.

### 2.2.3 Security of DES

Security of DES fall into two categories: key size and the nature of the algorithm. With the key length of 56 bits, there are $2^{56}$ possible keys, which are approximately $7.2 \times 10^{16}$ keys. Thus on the face of it, a brute force attack appears impractical. Assuming that on average half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher.

Another possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Internal working of algorithm were not made public in such a way cryptanalysis is possible for an opponent who knows the weakness in the S-boxes, over the years a number of regularities and unexpected behaviours of the S-boxes have been discovered.

DES can no longer be considered a sufficiently secured algorithm if the DES secured message can be broken in minutes by a super computer. Then the rapidly increasing power of computer means, it’ll be trivial to break DES in future. An extension of DES called DESX is considered virtually immune to key search.

### 2.3 International Data Encryption Algorithm (IDEA)

The international data encryption algorithm is a symmetric block cipher developed by Xuejia Lai & James Massey in the Swiss federal institute of Technology.
in 1990 and was called the proposed encryption standard PES. In 1991, Lai & Massey strengthened the algorithm against differential crypt analysis and called the result improved PES. The IPES name was changed to International Data Encryption Algorithm in 1992. IDEA is one of a number of conventional encryption algorithms that have been proposed in recent years to replace DES. In terms of adoption, IDEA is one of the most successful of these proposals. IDEA is best known for its use in PGP (pretty good privacy) in network protocol (Lai, Xuejia, and Massey, James L Springer-Verlag, 1991).

2.3.1 The Algorithm

IDEA algorithm with the key length of 128 bits, a block size of the 64 bits and as with DES, the same algorithm provides encryption and decryption. IDEA consists of 8 rounds using 52 sub keys. Each round uses 6 sub keys with the remaining 4 being used for output transformation. The Sub-keys are created as follows:

1) The 128 bit key is divided into 8 16 bit keys to provide the first 8 sub keys.
2) The bits of the original key are then shifted 25 bits to the left, and then it is again split in 8 sub keys.
3) The shifting & splitting is repeated until all 52 sub keys (SK1-SK52) have been created. The 64 bit plain text is first split into four blocks.

A round then consists of the following steps:

Output Block-1 (OB1) = B1 * SK1 (multiply 1st Sub-Block with 1st Sub Key)
   (OB2) = B2 + SK2 (add 2nd Sub-Block with 2nd Sub Key)
   (OB3) = B3 + SK3 (add 3rd sub-block with 3rd sub key)
   (OB4) = bB4 * SK4 (multiply 4th sub-block with 4th sub key)
   (OB5) = OB1 XOR OB3 (XOR results of 1 & 3)
   (OB6) = OB2 XOR OB4
   (OB7) = OB5 * SK5
   (OB8) = OB6 + OB7
   (OB9) = OB8 * SK6
   (OB10) = OB7+OB9
   (OB11) = OB1 XOR OB9
(OB12) = OB3 XOR OB9
(OB13) = OB2 XOR OB10
(OD14) = OB4 XOR OB10

The input to the next round is the four sub blocks OB11, OB12, OB13, and OB14 in that order. After the eighth round, the four final output blocks (F1-F4) are used in a final transformation to produce 4 sub blocks of cipher text c1-c4 that are then rejoined to form final 64 bit block of cipher text.

C1=F1 * SK49
C2=F2 + SK50
C3=F3 + SK51
C4=F4 + SK52

Cipher Text= C1 C2 C3 C4

2.3.2 Security provided by IDEA

IDEA is approximately twice as fast as DES and is also considerably more secure. Using a brute force approach there are 2128 possible keys. If a billion chips that could each test 1 billion keys a second would try and crack an IDEA encrypted message, it would take them 1013 years. Being a fairly new algorithm, it is possible a better attack than brute force will be found, when coupled with more powerful machines in the future may be able to crack a message. The security of IDEA depends upon the fundamental tenet of cryptography, breaking IDEA by brute-force attack requires an unbelievable computing cost. However, a long way into future IDEA seems to be a very secure algorithm.

Similarities between IDEA and DES

- IDEA runs in rounds. It has 17 rounds, while DES has 16 rounds.
- IDEA takes 64 bits block of plaintext as input and produces 64- bits ciphertext block as output.
- IDEA also has a complex cipher function f_k. There is no need to run this function in the reverse order for decryption. Instead, the cipher function runs in the same direction for both processes of encryption and decryption.
2.4 Blowfish

Blowfish is a variable-length key block cipher developed by Bruce Schneier. It does not meet all the requirements for a new cryptographic standard discussed above: It is only suitable for applications where the key does not change often, like a communications link or an automatic file encrypt or. It is significantly faster than DES when implemented in 32-bit microprocessors with large data caches, such as the Pentium and the Power PC. Blowfish provides a good encryption rate in software and no effective cryptanalysis of it has been found to date. However, the Advanced Encryption Standard now receives more attention. Schneier designed Blowfish as a general-purpose algorithm, intended as an alternative to the ageing DES and free of the problems and constraints associated with other algorithms (Bruce Schneier, 1993).

2.4.1 Description of the Algorithm

Blowfish is a variable-length key, 64-bit block cipher. The algorithm consists of two parts; a key-expansion part and a date-encryption part. Key expansion converts a key of at most 448 bits (Bruce Schneier, 1993) into several sub-key arrays totalling 4168 bytes. Data encryption occurs via a 16-round Feistel network. Each round consists of a key dependent permutation, and a key-and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round.

2.4.2 Sub-keys

Blowfish uses a large number of sub-keys. These keys must be precomputed before any data encryption or decryption.

1. The P-array consists of 18, 32-bit sub-keys:

   \[ P_1, P_2 \ldots P_{18} \]

2. There are four 32-bit S-boxes with 256 entries each:

   \[ S_1, 0, \quad S_1, 1 \ldots \quad S_1, 255; \]
   \[ S_2, 0, \quad S_2, 1\ldots \quad S_2, 255; \]
   \[ S_3, 0, \quad S_3, 1\ldots \quad S_3, 255; \]
   \[ S_4, 0, \quad S_4, 1\ldots \quad S_4, 255; \]

The exact method used to calculate these sub-keys will be described later.
2.4.3 Encryption and Decryption

Blowfish is a Fiestal (H. Feistel, *May 73,*) network consisting of 16 rounds. The input is a 64-bit data element x.
Divide x into two 32-bit halves: xL, xR.

Then for i = 1 to 16:

\[
\begin{align*}
xL &= xL \oplus P_i \\
xR &= F(xL) \oplus xR \\
\text{Swap } xL \text{ and } xR \\
\text{Next } i \\
\text{Swap } xL \text{ and } xR \text{ (Undo the last swap)}
\end{align*}
\]

Then \( xR = xR \oplus P_{17} \) and \( xL = xL \oplus P_{18} \).
Now combine xL and xR Function F:

\[
F(xL) = ((S1, a + S2, b + \text{mob } 2^{32}) \oplus S3, c) + S4, d \text{ MOB } 2^{32}
\]

Decryption is exactly the same as encryption, except that P1, P2 … P18 are used in the reverse order. Implementations of Blowfish that require the fastest speeds should unroll the loop and ensure that all sub keys are stored in cache.

Generating the Sub keys

The sub keys are calculated using the Blowfish algorithm. The exact method is as follows:

1. Initialize first the P-array and then the four S-boxes, in order with fixed string. This string consists of the hexadecimal digits of fractional part of Pi (less the initial 3).

*For example:*

\[
\begin{align*}
P1 &= 243f6a88 \\
P2 &= 85a308d3 \\
P3 &= 13198a2e \\
P4 &= 03707344
\end{align*}
\]

2. XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle
through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc equivalent keys.)

3. Encrypt the all-zero string with the Blowfish algorithm using the sub keys described in steps (1) and (2).
4. Replace P1 and P2 with the output of step (3).
5. Encrypt the output of step (3) using the Blowfish algorithm with the modified sub-keys.
6. Replace P3 and P4 with the output of step (5).
7. Continue the process, replacing all entries of the P-array, and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

In total, 521 iterations are required to generate all required sub keys. Applications can store the sub-keys rather than derivation process multiple times.

2.5 RC Cipher

The RC algorithms are a set of symmetric-key encryption algorithms invented by Ron Rivest. The "RC" may stand for either Rivest's cipher or, more informally, Ron's code. Despite the similarity in their names, the algorithms are for the most part unrelated. Different RC Ciphers described briefly below.

2.5.1 RC2

Also known as ARC2, It was designed as a quick-fix replacement for DES that is more secure. It is a block cipher with a variable key size that has proprietary algorithm RC2 is a variable-key length cipher. RC2 is a 64-bit block cipher with a variable size key. Its 18 rounds are arranged as a source-heavy Feistel network, with 16 rounds of one type (MIXING) punctuated by two rounds of another type (MASHING).

2.5.2 RC4

It was developed by Ron Rivest in 1987. It is a variable-key-size stream cipher. The details of the algorithm have not been officially published. The algorithm
is extremely easy to describe and program just like RC2, 40 bit RC4 is supported by the Microsoft base Cryptography provider, and the enhanced provider allows keys in the range of 40 to 128 bits in 8-bit increments. The two main reasons which helped its use over such a big range of applications are its speed and simplicity. Uses of RC4 in both software and hardware are extremely easy to develop. The RC4 (W. Stallings, 2005) encryption algorithm is started with a different key length, usually between 40 and 256 bits, using the key-scheduling algorithm (KSA).

2.5.3 RC5
RC5 is a block designed for speed. It allows a user defined key length, data block size, and number of encryption rounds. In particular, the key size can be as large as 2,048 bits. Unlike many schemes, RC5 has a variable block size (32, 64 or 128 bits), key size (0 to 2040 bits) and number of rounds (0 to 255). The original suggested choice of parameters was a block size of 64 bits, a 128-bit key and 12 rounds. A key feature of RC5 is the use of data-dependent rotations; one of the goals of RC5 was to prompt the study and evaluation of such operations as a cryptographic primitive.

2.5.4 RC6
RC6 is a new block cipher submitted to NIST for consideration as the new Advanced Encryption Standard (AES). The design of RC6 began with a consideration of RC5 as a potential candidate for an AES submission. Modifications were then made to meet the AES requirements, to increase security, and to improve performance. The inner loop, however, is based on the same "half-round" found in RC5. The standard mode RC6 operates on 128-bit input/output blocks (32-bit words) with variable-length keys -- but the RC6 design offers great inherent flexibility in the word-size of the basic computational unit, and the number of rounds to be specified, as well as in the length of the encryption key.

2.6 Conclusion
There are several methods of conventional cryptography, and since it is not possible to present all the methods, very important and popular methods were
presented. It is seen that the modified Hill cipher Encryption and Decryption requires generating random Matrix, which is essentially the power of security. As we know in Hill cipher Decryption requires inverse of the matrix. Hence while decryption one problem arises that is, Inverse of the matrix does not always exist. Then if the matrix is not invertible then encrypted text cannot be decrypted. But this drawback is completely eliminated in modified Hill cipher algorithm. At the same time, this method requires the cracker to find the inverse of many square matrices which is not computationally easy. So this modified Hill-Cipher method is both easy to implement and difficult to crack.