3

Hybrid Cryptography

To paraphrase Lao Tse, you cannot create trust with cryptography, no matter how much cryptography you use.

— Jon Callas.

3.1 Introduction

Although electronic documents have been greatly improved in the transferring speed and processing speed, it would also result that electronic documents information is disclosed, counterfeited, tampered, repudiated and so on. Therefore due to the rapid usages of data communication, security is becoming a more crucial issue. [85] The fundamental requirements for security include authentication, confidentiality, integrity, and non-repudiation. To provide such security services, cryptography is the foundation and most system uses two major classes of cryptographic algorithms namely private-key and public-key algorithms. In private-key algorithms, same key is used for both encryption and decryption. They usually operate at relatively high speed and are suitable for bulk encryption of messages. Public-key algorithms are based on the idea of separating the key used to encrypt a message from the one used to decrypt it. They are relatively slow and therefore unsuitable for encryption of large bulky messages. [7,15,10,90].

According to the Merriam-Webster dictionary, hybrid means “something that is formed by combining two or more things”. As the name “Hybrid cryptography” depicts that it is a mixed up approach of the both types of cryptographies namely the symmetric one as well as the asymmetric one. Thus to solve the problems encountered when trying to process long messages quickly, hybrid cryptography uses two technologies together in a complementary manner, with each performing a different function.
Symmetric and asymmetric ciphers each have their own advantages and disadvantages (explained later in this chapter). Symmetric ciphers are 100 times faster than asymmetric ciphers [7,48], but have some drawbacks like lack of scalability, difficult key management and provides only confidentiality. The asymmetric algorithms allow public key infrastructure and key exchange systems, but at the cost of speed. A hybrid cryptosystem is a protocol using multiple ciphers of different types together, each to its best advantage. One common approach is to generate a random secret key for a symmetric cipher, and then encrypt this key via an asymmetric cipher using the recipient’s public key. The message itself is then encrypted using the symmetric cipher and the secret key. Both the encrypted secret key and the encrypted message are then sent to the recipient. The recipient decrypts the secret key first using his private key and then uses that key to decrypt the message [91,48].

An asymmetrical key is utilized to encrypt the message or block of bytes and it is passed on the network with the help of an arbitrary key which is generated. This arbitrary key is used to encrypt the rest of the messages with the help of the symmetrical key encryption. In such type of cryptography both the features are utilized one is the pace and efficiency of the symmetrical algorithms and another one is the exchanging of private key securely. This type of cryptography is utilized in the Secure Socket Layer (SSL), SSH (Secure Socket Hashing), and PGP.

The ISO/IEC JTC1/SC27 standardisation committee suggest that hybrid cryptography can be defined as the branch of asymmetric cryptography that makes use of convenient symmetric techniques to remove some of the problems inherent in normal asymmetric cryptosystems. [92] But this definition is not technically correct because S-boxes of symmetric technique cannot be used in hybrid system. According to Alexander W. Dent [34] it is better to define hybrid cryptography as the branch of asymmetric cryptography
that makes use of keyed symmetric cryptosystems as black-box algorithms with certain security properties. The critical point of this definition is that it is the properties of the symmetric cryptosystem that are used to construct the asymmetric scheme, rather than the technical details about the way in which the symmetric algorithm achieves these security properties. We specify the use of keyed symmetric algorithms to make sure that an asymmetric cryptosystems that makes use of hash functions (as almost all asymmetric cryptosystems seem to do) are not automatically classed as hybrid schemes.

Traditionally, hybrid cryptography has been concerned with building asymmetric encryption schemes. In these cryptosystems a symmetric encryption scheme is used to overcome the problems typically associated with encrypting long messages using “pure” asymmetric techniques. This is typically achieved by encrypting the message with a symmetric encryption scheme and a randomly generated symmetric key. This random symmetric key is then somehow encrypted using an asymmetric encryption scheme. This approach has been successfully used for many years [18,93,94]; although it has been shown that a naive implementation may derive a breakable scheme [25]. In 2005 Dent introduced the idea of signcryption to hybrid cryptography and proposed the concept of hybrid signcryption.

One recent advance in hybrid cryptography is the development of the KEM–DEM model for hybrid encryption algorithms which was first introduced by Cramer and Shoup [93,95,96] until 2000. This model splits a hybrid encryption scheme into two distinct components: an asymmetric key encapsulation mechanism (KEM) and a symmetric data encapsulation mechanism (DEM). Whilst the KEM–DEM model does not model all possible hybrid encryption schemes, and there are several examples of hybrid encryption schemes that do not fit into the KEM–DEM model, it does have the advantage of allowing the security requirements of the asymmetric and symmetric parts of the scheme to be completely separated and studied.
Hybrid Cryptography

independently. This approach has proven to be very popular and several emerging standards are strongly supporting its use [34,33,96]. Shoup’s KEM–DEM model demonstrates what should be an overriding principle of hybrid cryptography: it is not necessary for an asymmetric scheme to fully involve itself in the details of providing a security service — the security service can be provided by a symmetric scheme provided the asymmetric scheme is in full control of that process (say, by generating the secret key that the symmetric scheme uses). Hence, we can fully separate the asymmetric and symmetric parts of the scheme [93].

In nutshell, hybrid cryptography is one which combines the convenience of public-key cryptosystem with the efficiency of a symmetric-key cryptosystem. A lot of research papers have been searched and reviewed related to the problem and found that the major scope of work has been done using RSA [97]. In the developed application RSAAPP, DES algorithm of private key cryptosystem and RSA algorithm of public key cryptosystem are being used to implement the hybrid cryptography, so both symmetric-key cryptosystem and public-key cryptosystem are being explained, including DES and RSA algorithms, in details below.

3.2 Symmetric Key Cryptosystems

It is also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the development of public key cryptosystem in the 1970s. In secret key cryptosystems, a single key is used for both encryption and decryption. As shown in Figure 5, the sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption. With this form of cryptography, it is obvious that the key must be known to both the
sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key. The Figure 5 below explains the simplified model of symmetric cryptosystems –

![Figure 5: Simplified Model of Symmetric Encryption](image)

Classical cryptography used two major types of ciphers, namely substitution ciphers (such as shift ciphers and affine ciphers) and transposition ciphers. It relied heavily on the use of secret keys for security.

Modern cryptography uses the same basic ideas as traditional cryptography but in a different way. Transpositions and substitutions are now implemented as simple circuits called P-Box and S-Box. In block ciphers, the S-boxes and P-Boxes are used to make the relation between the plaintext and the ciphertext difficult to understand. [98]

### 3.2.1 P-Box (Permutation boxes)

P-boxes are typically classified as compression, expansion, and straight, according as the number of output bits is less than, greater than, or equal to the number of input bits. Only straight P-boxes are invertible. These are devices that accept a certain length bit string and produce a 'transposition' or permutation of the bits. By appropriate internal wiring, a P-box can produce any permutation of the input bit string. The following diagram shows a P-box...
that accepts an 8-bit string and, if the bits are designated from top to bottom as 01234567, then it outputs the bits in the order 36071245.

![Figure 6: P-box (Permutation Box)](image)

### 3.2.2 S-Box (Substitution-box)

In cryptography, an S-Box (Substitution-box) is a basic component of symmetric key algorithms which performs substitution. In block ciphers, they are typically used to obscure the relationship between the key and the ciphertext — Shannon's property of confusion. In general, an S-Box takes some number of input bits, m, and transforms them into some number of output bits, n, where n is not necessarily equal to m. [98]

These are devices that accept a certain length bit string; the output is the result of applying a substitution cipher to the bit string. The following diagram shows an S-box that accepts a 3-bit string (note there are eight possible 3-bit strings); the 3-bit input selects one of the eight lines exiting from the first stage and sets it to 1 - the other seven exit lines are set to 0. The second stage is a P-box, and hence outputs an eight bit string with exactly one bit equal to 1. The third stage encodes this eight bit string into a 3-bit binary string again. Changing the wiring of the central P-box appropriately enables any substitution to be achieved.
### 3.2.3 Product Cipher

It is also called a mixing transformation. A Substitution-Permutation Network has sets of S-boxes linked by P-boxes. Using several of these P-boxes and S-boxes one after the other in some manner enables the construction of powerful ‘product ciphers’.

In modern terminology, the systems where encryption and decryption are done using same key are called Private Key Cryptosystems or Symmetric Cryptosystems. Such a system, however, has the following drawbacks:

- The sender and recipient of a message must somehow agree on the key to be used for both encryption and decryption without anyone else finding out what it is.
• Keys must be distributed secretly. They are as valuable as the message that they encrypt.
• Keys cannot be compromised which not only means loss of security but also false messages from an intruder who has the knowledge of the key to fool the other users.
• Unmanageable key-space. If the use of a separate key for each pair of users in the network is assumed then total number of keys is proportional to n^2, which is very large when n gets large.

Many private key algorithms exist today like DES and its many forms, IDEA, RC6, RIJNDEL, AES (which recently made it as standard algorithm for US NIST), TWOFISH etc. However, since the DES algorithm was the most popular and widely used and forms a part of the developed application, only it will be discussed.

3.2.4 Data Encryption Standard (DES)

In 1972, the National Institute of Standards and Technology (called the National Bureau of Standards at the time) decided that a strong cryptographic algorithm was needed to protect non-classified information. The algorithm was required to be cheap, widely available, and very secure. NIST envisioned something that would be available to the general public and could be used in a wide variety of applications. So they asked for public proposals for such an algorithm. In 1974 IBM submitted the Lucifer algorithm, which appeared to meet most of NIST's design requirements.

NIST enlisted the help of the National Security Agency to evaluate the security of Lucifer. At the time many people distrusted the NSA due to their extremely secretive activities, so there was initially a certain degree of skepticism regarding the analysis of Lucifer. One of the greatest worries was that the key length, originally 128 bits, was reduced to just 56 bits, weakening it significantly. The NSA was also accused of changing the algorithm to plant a "back door" in it that would allow agents to decrypt any information without
having to know the encryption key. But these fears proved unjustified and no such back door has ever been found.

The modified Lucifer algorithm was adopted by NIST as a federal standard on November 23, 1976. Its name was changed to the Data Encryption Standard (DES). The algorithm specification was published in January 1977, and with the official backing of the government it became a very widely employed algorithm in a short amount of time.

Unfortunately, over time various shortcut attacks were found that could significantly reduce the amount of time needed to find a DES key by brute force. And as computers became progressively faster and more powerful, it was recognized that a 56-bit key was simply not large enough for high security applications. As a result of these serious flaws, NIST abandoned their official endorsement of DES in 1997 and began work on a replacement, to be called the Advanced Encryption Standard (AES). Despite the growing concerns about its vulnerability, DES is still widely used by financial services and other industries worldwide to protect sensitive on-line applications.

To highlight the need for stronger security than a 56-bit key can offer, RSA Data Security has been sponsoring a series of DES cracking contests since early 1997. In 1998 the Electronic Frontier Foundation won the RSA DES Challenge II-2 contest by breaking DES in less than 3 days. EFF used a specially developed computer called the DES Cracker, which was developed for under $250,000. The encryption chip that powered the DES Cracker was capable of processing 88 billion keys per second. More recently, in early 1999, Distributed. Net used the DES Cracker and a worldwide network of nearly 100,000 PCs to win the RSA DES Challenge III in a record breaking 22 hours and 15 minutes. The DES Cracker and PCs combined were testing 245 billion keys per second when the correct key was found. In addition, it has been shown that for a cost of one million dollars a dedicated hardware device can be built that can search all possible DES keys in about 3.5 hours. This just
serves to illustrate that any organization with moderate resources can break through DES with very little effort these days.

DES encrypts and decrypts data in 64-bit blocks, using a 64-bit key (although the effective key strength is only 56 bits, as explained below). It takes a 64-bit block of plaintext as input and outputs a 64-bit block of ciphertext. Since it always operates on blocks of equal size and it uses both permutations and substitutions in the algorithm, DES is both a block cipher and a product cipher.

DES has 16 rounds, meaning the main algorithm is repeated 16 times to produce the ciphertext. It has been found that the number of rounds is exponentially proportional to the amount of time required to find a key using a brute-force attack. So as the number of rounds increases, the security of the algorithm increases exponentially.

The practical implementation of DES in C for this application is based on following simplified step-by-step description:

**Key Scheduling**

Although the input key for DES is 64 bits long, the actual key used by DES is only 56 bits in length. The least significant (right-most) bit in each byte is a parity bit, and should be set so that there are always an odd number of 1s in every byte. These parity bits are ignored, so only the seven most significant bits of each byte are used, resulting in a key length of 56 bits.

The first step is to pass the 64-bit key through a permutation called Permuted Choice 1, or PC-1 for short. The table for this is given below. Note that in all subsequent descriptions of bit numbers, 1 is the left-most bit in the number, and n is the rightmost bit.
Table 1: Permuted Choice 1 (PC-1)

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>57</td>
<td>49</td>
<td>41</td>
<td>33</td>
<td>25</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>58</td>
<td>50</td>
<td>42</td>
<td>34</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>2</td>
<td>59</td>
<td>51</td>
<td>43</td>
<td>35</td>
<td>27</td>
</tr>
<tr>
<td>22</td>
<td>19</td>
<td>11</td>
<td>3</td>
<td>60</td>
<td>52</td>
<td>44</td>
<td>36</td>
</tr>
<tr>
<td>29</td>
<td>63</td>
<td>55</td>
<td>47</td>
<td>39</td>
<td>31</td>
<td>23</td>
<td>15</td>
</tr>
<tr>
<td>36</td>
<td>7</td>
<td>62</td>
<td>54</td>
<td>46</td>
<td>38</td>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td>43</td>
<td>14</td>
<td>6</td>
<td>61</td>
<td>53</td>
<td>45</td>
<td>37</td>
<td>29</td>
</tr>
<tr>
<td>50</td>
<td>21</td>
<td>13</td>
<td>5</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>

For example, we can use the PC-1 table to figure out how bit 30 of the original 64-bit key transforms to a bit in the new 56-bit key. Find the number 30 in the table, and notice that it belongs to the column labeled 5 and the row labeled 36. Add up the value of the row and column to find the new position of the bit within the key. For bit 30, 36 + 5 = 41, so bit 30 becomes bit 41 of the new 56-bit key. Note that bits 8, 16, 24, 32, 40, 48, 56 and 64 of the original key are not in the table. These are the unused parity bits that are discarded when the final 56-bit key is created.

Now that we have the 56-bit key, the next step is to use this key to generate 16 48-bit subkeys, called K[1]-K[16], which are used in the 16 rounds of DES for encryption and decryption. The procedure for generating the subkeys - known as key scheduling - is fairly simple:

1. Set the round number R to 1.
2. Split the current 56-bit key, K, up into two 28-bit blocks, L (the left-hand half) and R (the right-hand half).
3. Rotate L left by the number of bits specified in the table below, and rotate R left by the same number of bits as well.
4. Join L and R together to get the new K.
5. Apply Permuted Choice 2 (PC-2) to \( K \) to get the final \( K[R] \), where \( R \) is the round number we are on.

6. Increment \( R \) by 1 and repeat the procedure until we have all 16 subkeys \( K[1]-K[16] \).

Here are the tables involved in these operations:

<table>
<thead>
<tr>
<th>Round Number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bits to rotate</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Plaintext Preparation**

Once the key scheduling has been performed, the next step is to prepare the plaintext for the actual encryption. This is done by passing the plaintext through a permutation called the Initial Permutation, or IP for short. This table also has an inverse, called the Inverse Initial Permutation, or IP\(^{-1}\). Sometimes IP\(^{-1}\) is also called the Final Permutation. Both of these tables are shown below.
Table 4: Initial Permutation (IP)

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>58</td>
<td>50</td>
<td>42</td>
<td>34</td>
<td>26</td>
<td>18</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>52</td>
<td>44</td>
<td>36</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>62</td>
<td>54</td>
<td>46</td>
<td>38</td>
<td>30</td>
<td>22</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>64</td>
<td>56</td>
<td>48</td>
<td>40</td>
<td>32</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>33</td>
<td>57</td>
<td>49</td>
<td>41</td>
<td>33</td>
<td>25</td>
<td>17</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>59</td>
<td>51</td>
<td>43</td>
<td>35</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>49</td>
<td>61</td>
<td>53</td>
<td>45</td>
<td>37</td>
<td>29</td>
<td>21</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>57</td>
<td>63</td>
<td>55</td>
<td>47</td>
<td>39</td>
<td>31</td>
<td>23</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 5: Inverse Initial Permutation (IP⁻¹)

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>8</td>
<td>48</td>
<td>16</td>
<td>56</td>
<td>24</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>39</td>
<td>7</td>
<td>47</td>
<td>15</td>
<td>55</td>
<td>23</td>
<td>63</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>38</td>
<td>6</td>
<td>46</td>
<td>14</td>
<td>54</td>
<td>22</td>
<td>62</td>
<td>30</td>
</tr>
<tr>
<td>25</td>
<td>37</td>
<td>5</td>
<td>45</td>
<td>13</td>
<td>53</td>
<td>21</td>
<td>61</td>
<td>29</td>
</tr>
<tr>
<td>33</td>
<td>36</td>
<td>4</td>
<td>44</td>
<td>12</td>
<td>52</td>
<td>20</td>
<td>60</td>
<td>28</td>
</tr>
<tr>
<td>41</td>
<td>35</td>
<td>3</td>
<td>43</td>
<td>11</td>
<td>51</td>
<td>19</td>
<td>59</td>
<td>27</td>
</tr>
<tr>
<td>49</td>
<td>34</td>
<td>2</td>
<td>42</td>
<td>10</td>
<td>50</td>
<td>18</td>
<td>58</td>
<td>26</td>
</tr>
<tr>
<td>57</td>
<td>33</td>
<td>1</td>
<td>41</td>
<td>9</td>
<td>49</td>
<td>17</td>
<td>57</td>
<td>25</td>
</tr>
</tbody>
</table>

These tables are used just like PC-1 and PC-2 were for the key scheduling. By looking at the table is becomes apparent why one permutation is called the inverse of the other. For example, let's examine how bit 32 is transformed under IP. In the table, bit 32 is located at the intersection of the column labeled 4 and the row labeled 25. So this bit becomes bit 29 of the 64-bit block after the permutation. Now let's apply IP⁻¹. In IP⁻¹, bit 29 is located at the intersection of the column labeled 7 and the row labeled 25. So this bit becomes bit 32 after the permutation. And this is the bit position that we
started with before the first permutation. So $IP^{-1}$ really is the inverse of IP. It does the exact opposite of IP. If you run a block of plaintext through IP and then pass the resulting block through $IP^{-1}$, you'll end up with the original block.

**DES Core Function**

Once the key scheduling and plaintext preparation have been completed, the actual encryption or decryption is performed by the main DES algorithm. The 64-bit block of input data is first split into two halves, L and R. L is the left-most 32 bits, and R is the right-most 32 bits. The following process is repeated 16 times, making up the 16 rounds of standard DES. We call the 16 sets of halves L[0]-L[15] and R[0]-R[15].

1. R[I-1] - where I is the round number, starting at 1 - is taken and fed into the E-Bit Selection Table, which is like a permutation, except that some of the bits are used more than once. This expands the number R[I-1] from 32 to 48 bits to prepare for the next step.
2. The 48-bit R[I-1] is XORed with K[I] and stored in a temporary buffer so that R[I-1] is not modified.
3. The result from the previous step is now split into 8 segments of 6 bits each. The left-most 6 bits are B[1], and the right-most 6 bits are B[8]. These blocks form the index into the S-boxes, which are used in the next step. The Substitution boxes, known as S-boxes, are a set of 8 two-dimensional arrays, each with 4 rows and 16 columns. The numbers in the boxes are always 4 bits in length, so their values range from 0-15. The S-boxes are numbered S[1]-S[8].
4. Starting with B[1], the first and last bits of the 6-bit block are taken and used as an index into the row number of S[1], which can range from 0 to 3, and the middle four bits are used as an index into the column number, which can range from 0 to 15. The number from this position in the S-box is retrieved and stored away. This is repeated with B[2] and S[2], B[3] and
S[3], and the others up to B[8] and S[8]. At this point, you now have eight 4-bit numbers, which when strung together one after the other in the order of retrieval, give a 32-bit result.

5. The result from the previous stage is now passed into the P Permutation.

6. This number is now XORed with L[I-1], and moved into R[I]. R[I-1] is moved into L[I].

7. At this point we have a new L[I] and R[I]. Here, we increment I and repeat the core function until I = 17, which means that 16 rounds have been executed and keys K[1]-K[16] have all been used.

When L[16] and R[16] have been obtained, they are joined back together in the same fashion they were split apart (L[16] is the left-hand half, R[16] is the right-hand half), then the two halves are swapped, R[16] becomes the left-most 32 bits and L[16] becomes the right-most 32 bits of the pre-output block and the resultant 64-bit number is called the pre-output.

The Figure 9: Feistel function (F function) for calculation of $f(R, K)$ below explains the Feistel function for calculation of $f(R, K)$.

$$f(R_{i-1}, K_i) = P(S(E(R_{i-1} \oplus K_i)))$$

Figure 9: Feistel function (F function) for calculation of $f(R, K)$
Tables used in the DES Core Function

Table 6: E-Bit Selection Table

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>19</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>25</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>31</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>37</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>43</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7: Permutation (P)

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>7</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>29</td>
<td>12</td>
<td>28</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>15</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>18</td>
<td>31</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>8</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>21</td>
<td>32</td>
<td>27</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>13</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>29</td>
<td>22</td>
<td>11</td>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>
### Table 8: Substitution Box 1 (S-Box 1)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>2</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>14</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

### Table 9: Substitution Box 2 (S-Box 2)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>1</td>
<td>8</td>
<td>14</td>
<td>6</td>
<td>11</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>13</td>
<td>12</td>
<td>0</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>13</td>
<td>4</td>
<td>7</td>
<td>15</td>
<td>2</td>
<td>8</td>
<td>14</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>14</td>
<td>7</td>
<td>11</td>
<td>10</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>12</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>8</td>
<td>10</td>
<td>1</td>
<td>3</td>
<td>15</td>
<td>4</td>
<td>2</td>
<td>11</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>5</td>
<td>14</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

### Table 10: Substitution Box 3 (S-Box 3)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
<td>9</td>
<td>14</td>
<td>6</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>1</td>
<td>13</td>
<td>12</td>
<td>7</td>
<td>11</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>7</td>
<td>0</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>14</td>
<td>12</td>
<td>11</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>15</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>5</td>
<td>10</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>10</td>
<td>13</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>15</td>
<td>14</td>
<td>3</td>
<td>11</td>
<td>5</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

### Table 11: Substitution Box 4 (S-Box 4)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>13</td>
<td>14</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>11</td>
<td>12</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>8</td>
<td>11</td>
<td>5</td>
<td>6</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>12</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>0</td>
<td>12</td>
<td>11</td>
<td>7</td>
<td>13</td>
<td>15</td>
<td>1</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>15</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>12</td>
<td>7</td>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>
Table 12: Substitution Box 5 (S-Box 5)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>13</td>
<td>0</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>11</td>
<td>2</td>
<td>12</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>13</td>
<td>7</td>
<td>8</td>
<td>15</td>
<td>9</td>
<td>12</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>8</td>
<td>12</td>
<td>7</td>
<td>1</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>6</td>
<td>15</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 13: Substitution Box 6 (S-Box 6)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>1</td>
<td>10</td>
<td>15</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>14</td>
<td>7</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>15</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>12</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>13</td>
<td>14</td>
<td>0</td>
<td>11</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>14</td>
<td>15</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>12</td>
<td>9</td>
<td>5</td>
<td>15</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 14: Substitution Box 7 (S-Box 7)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>11</td>
<td>2</td>
<td>14</td>
<td>15</td>
<td>0</td>
<td>8</td>
<td>13</td>
<td>3</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>10</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>0</td>
<td>11</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>14</td>
<td>3</td>
<td>5</td>
<td>12</td>
<td>2</td>
<td>15</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>3</td>
<td>7</td>
<td>14</td>
<td>10</td>
<td>15</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>11</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 15: Substitution Box 8 (S-Box 8)

<table>
<thead>
<tr>
<th>Row / Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>15</td>
<td>11</td>
<td>1</td>
<td>10</td>
<td>9</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>0</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>15</td>
<td>13</td>
<td>8</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>12</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>11</td>
<td>4</td>
<td>1</td>
<td>9</td>
<td>12</td>
<td>14</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>13</td>
<td>15</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>14</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>8</td>
<td>13</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>
Method to use the S-Boxes

The purpose of this example is to clarify how the S-boxes work. Suppose we have the following 48-bit binary number:

\[011101000101110101000111101000111001011011001011101\]

In order to pass this through steps 3 and 4 of the Core Function as outlined above, the number is split up into 8 6-bit blocks, labeled B[1] to B[8] from left to right:

\[011101 000101 110101 000111 101000 011100 101101 011101\]

Now, eight numbers are extracted from the S-boxes - one from each box:

- B[1] = S[1](01,1110) = S[1][1][14] = 3 = 0011
- B[6] = S[6](00,1110) = S[6][0][14] = 5 = 0101
- B[8] = S[8](01,1110) = S[8][1][14] = 9 = 1001

In each case of S[n][row][column], the first and last bits of the current B[n] are used as the row index, and the middle four bits as the column index. The results are now joined together to form a 32-bit number which serves as the input to stage 5 of the Core Function (the P Permutation):

\[00110100110011011010010110101001\]

Ciphertext Preparation

The final step is to apply the permutation IP^{-1} to the pre-output. The result is the completely encrypted ciphertext.

Encryption and Decryption

The same algorithm can be used for encryption or decryption. The method described above will encrypt a block of plaintext and return a block of ciphertext. In order to decrypt the ciphertext and get the original plaintext
Again, the procedure is simply repeated but the subkeys are applied in reverse order, from $K_{16}$-$K_1$. That is, stage 2 of the Core Function as outlined above changes from $R_{I-1}$ XOR $K[I]$ to $R_{I-1}$ XOR $K[17-I]$. Other than that, decryption is performed exactly the same as encryption. Figure 5 below show the structure of DES.

Figure 10: DES enciphering algorithm
The details of the DES described above can be summarized as follows

**Key Schedule:**

\[ C[0]D[0] = PC1(key) \]

for \(1 \leq i \leq 16\)

\[ C[i] = LS[i](C[i-1]) \]

\[ D[i] = LS[i](D[i-1]) \]

\[ K[i] = PC2(C[i]D[i]) \]

**Encipherment:**

\[ L[0]R[0] = IP \text{ (plain block)} \]

for \(1 \leq i \leq 16\)

\[ L[i] = R[i-1] \]

\[ R[i] = L[i-1] \oplus f(R[i-1],K[i]) \]

\[ \text{cipher block} = FP(R[16]L[16]) \]

**Decipherment:**

\[ R[16]L[16] = IP \text{ (cipher block)} \]

for \(1 \leq i \leq 16\)

\[ R[i-1] = L[i] \]

\[ L[i-1] = R[i] \oplus f(L[i],K[i]) \]

\[ \text{plain block} = FP(L[0]R[0]) \]

The mode of operation is an important factor for block ciphers, which specifies how a block cipher with a fixed block size (8 bytes for DES, 16 for AES) can be extended to process messages of arbitrary length. These modes have different error propagation, pattern concealment, and integrity protection properties. There are several “modes of operation” that DES can use, some of them better than others the following four are the most important [99]:

**ECB (Electronic Code Book)**

This is the regular DES algorithm, exactly as described above. Data is divided into 64-bit blocks and each block is encrypted one at a time. Separate
encryptions with different blocks are totally independent of each other. This means that if data is transmitted over a network or phone line, transmission errors will only affect the block containing the error. It also means, however, that the blocks can be rearranged, thus scrambling a file beyond recognition, and this action would go undetected. ECB is the weakest of the various modes because no additional security measures are implemented besides the basic DES algorithm. However, ECB is the fastest and easiest to implement, making it the most common mode of DES seen in commercial applications. This is the mode of operation used by Private Encryptor.

**CBC (Cipher Block Chaining)**

In this mode of operation, each block of ECB encrypted ciphertext is XORed with the next plaintext block to be encrypted, thus making all the blocks dependent on all the previous blocks. This means that in order to find the plaintext of a particular block, you need to know the ciphertext, the key, and the ciphertext for the previous block. The first block to be encrypted has no previous ciphertext, so the plaintext is XORed with a 64-bit number called the Initialization Vector, or IV for short. So if data is transmitted over a network or phone line and there is a transmission error (adding or deleting bits), the error will be carried forward to all subsequent blocks since each block is dependent upon the last. If the bits are just modified in transit (as is the more common case) the error will only affect all of the bits in the changed block, and the corresponding bits in the following block. The error doesn't propagate any further. This mode of operation is more secure than ECB because the extra XOR step adds one more layer to the encryption process.

**CFB (Cipher Feedback)**

In this mode, blocks of plaintext those are less than 64 bits long can be encrypted. Normally, special processing has to be used to handle files whose size is not a perfect multiple of 8 bytes, but this mode removes that necessity (Private Encryptor handles this case by adding several dummy bytes to the end
of a file before encrypting it). The plaintext itself is not actually passed through the DES algorithm, but merely XORed with an output block from it, in the following manner: A 64-bit block called the Shift Register is used as the input plaintext to DES. This is initially set to some arbitrary value, and encrypted with the DES algorithm. The ciphertext is then passed through an extra component called the M-box, which simply selects the left-most M bits of the ciphertext, where M is the number of bits in the block we wish to encrypt. This value is XORed with the real plaintext, and the output of that is the final ciphertext. Finally, the ciphertext is fed back into the Shift Register, and used as the plaintext seed for the next block to be encrypted. As with CBC mode, an error in one block affects all subsequent blocks during data transmission. This mode of operation is similar to CBC and is very secure, but it is slower than ECB due to the added complexity.

*OFB (Output Feedback)*

This is similar to CFB mode, except that the ciphertext output of DES is fed back into the Shift Register, rather than the actual final ciphertext. The Shift Register is set to an arbitrary initial value, and passed through the DES algorithm. The output from DES is passed through the M-box and then fed back into the Shift Register to prepare for the next block. This value is then XORed with the real plaintext (which may be less than 64 bits in length, like CFB mode), and the result is the final ciphertext. Note that unlike CFB and CBC, a transmission error in one block will not affect subsequent blocks because once the recipient has the initial Shift Register value; it will continue to generate new Shift Register plaintext inputs without any further data input. However, this mode of operation is less secure than CFB mode because only the real ciphertext and DES ciphertext output is needed to find the plaintext of the most recent block. Knowledge of the key is not required.
3.2.5 Advantages of Symmetric-key cryptosystems

1. Symmetric-key ciphers can be designed to have high rates of data throughput. Some hardware implementations achieve encrypts rates of hundreds of megabytes per second, while software implementations may attain throughput rates in the megabytes per second range.
2. Keys for symmetric-key ciphers are relatively short.
3. Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions, and computationally efficient digital signature schemes, to name just a few.
4. Symmetric-key ciphers can be composed to produce stronger ciphers. Simple transformations which are easy to analyze, but on their own it can be used to construct strong product ciphers.
5. Symmetric-key encryption is perceived to have an extensive history, although it must be acknowledged that, notwithstanding the invention of rotor machines earlier, much of the knowledge in this area has been acquired subsequent to the invention of the digital Computer, and, in particular, the design of the Data Encryption Standard in the early 1970s.

3.2.6 Disadvantages of symmetric-key cryptosystems

1. In a two-party communication, the key must remain secret at both ends.
2. In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted TTP.
3. In a two-party communication between entities A and B, sound cryptographic practice dictates that the key be changed frequently and perhaps for each communication session.
4. Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a TTP.

3.3 Public Key Cryptosystems

The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography [10]. The concept of public key cryptography evolved from an attempt to attack the most difficult problem associated with conventional symmetric cryptosystems: the distribution of private keys and the lack of secrecy thereof. Public-key cryptography provides a radical departure from all that has gone before. For one thing, public-key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

In 1976, two researchers at Stanford University, Diffie and Hellman [5], proposed a radically new kind of cryptosystem, one in which the encryption and decryption keys are different and the decryption key could not be derived from the encryption key. In their proposal the keyed encryption algorithm $E$ and the keyed decryption algorithm $D$ had to meet the following three requirements:

1. $D(E(P)) = P$
2. It is exceedingly difficult to deduce $D$ from $E$.
3. $E$ cannot be broken by a chosen plaintext attack.

The systems where the encryption algorithm relies on one key and the decryption algorithm depends on a different but related key are called Public Key Cryptosystems or Asymmetric Cryptosystems. Public key algorithms have the following important characteristics:
1. It is computationally infeasible to determine the decryption key given only the knowledge of the cryptographic algorithm and the encryption key.

2. In addition some algorithm such as RSA also exhibit that either of the two related keys can be used for encryption with the other used for decryption. This property is exploited in the usage of digital signature for authentication.

![Diagram](image)

Figure 11: An asymmetrical encryption algorithm requires the end user to generate public and private keys, and to provide the public key to the folks performing the encryption.

![Diagram](image)

Figure 12: Public Key Cryptosystem

The working of the public key cryptosystem as shown in Figure 11 and Figure 12 can be described as follows:
1. Each end system in a network generates a pair of keys to be used for encryption and decryption of messages that it will receive.

2. Each system publishes its encryption key by placing it in a public register or file. This is public key. The companion key is kept secret and is called private key.

3. If A wishes to send message to B, it encrypts the message using B’s public key.

4. When B receives the encrypted message, B decrypts it using B’s private key. So no other recipients can decrypt the message because only B knows B’s private key.

With this approach, all participants have access to public key. Private keys are generated locally by each participant and therefore needs no key distribution. As long as a system controls its private key, its incoming communication is safe and secure. At any time a system can change its private key and publish the companion public key to replace the old public key.

A number of algorithms have been proposed for public-key cryptography. Some of these, though initially promising but turned out to be breakable. One of the first successful responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978 [24]. The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption [10].

### 3.3.1 The RSA Algorithm

The RSA algorithm (named after the inventors Rivest, Shamir and Adleman) was one of the first cryptographic algorithms that met the requirements for public key systems as stated by Diffie and Hellman [5]. Since then it has reigned supreme as the only widely accepted and implemented general purpose approach to public key systems.
The RSA algorithm is divided into three stages namely Key Generation, Encryption and Decryption which are described below:

**Key Generation**
- a. Generate two large random primes, p and q, of approximately equal size such that their product \( n = p \times q \) is of the required bit length, e.g. 1024 bits and \( p \neq q \).
- b. Calculate \( n = p \times q \).
- c. Calculate Euler totient function \( \Phi(n) = (p-1) \times (q-1) \).
- d. Select integer \( e, 1 < e < \Phi(n) \) such that \( \gcd(e, \Phi(n)) = 1 \).
- e. Compute \( d, 1 < d < \Phi(n) \) such that \( ed \equiv 1 \pmod{\Phi(n)} \). Extended Euclidean Algorithm is used to calculate \( d \equiv e^{-1} \pmod{\Phi(n)} \). This is known as modular inversion. The modular inverse \( d \) is defined as the integer value such that \( ed \equiv 1 \pmod{\Phi(n)} \). It only exists if \( e \) and \( \Phi(n) \) have no common factors.
- f. Public key \( K_U = \{e, n\} \).
- g. Private key \( K_R = \{d, n\} \).

**Encryption**
- a. Plaintext : \( M \)
- b. Ciphertext : \( C = M^e \pmod{n} \)

**Decryption**
- a. Ciphertext : \( C \)
- b. Plaintext : \( M = C^d \pmod{n} \).

Since \( ed \equiv 1 \pmod{\Phi} \), there exists an integer \( k \) such that \( ed = 1 + k\Phi \).

Now, if \( \gcd(m, p) = 1 \) then by Fermat’s theorem,

\[
M^{p-1} \equiv 1 \pmod{p}.
\]

Raising both sides of the congruence to the power \( k(q-1) \) and then multiply both sides by \( m \) yields

\[
M^{1 + k(p-1)(q-1)} \equiv m \pmod{p}.
\]
On the other hand, if gcd(m, p) = p, then this last congruence is again valid since each side is congruent to 0 modulo p. Hence, in all cases – 

\[ M^{ed} \equiv m \pmod{p}. \]

By the same argument,

\[ M^{ed} \equiv m \pmod{q}. \]

Finally, both p and q are distinct relatively primes, it follows that

\[ M^{ed} \equiv m \pmod{n}. \]

and, hence,

\[ c^d = (M^e)^d \equiv m \pmod{n}. \]

**Illustration with a real example**

1. Select two prime numbers, p = 7 and q = 17.
2. Calculate \( n = p \times q = 7 \times 17 = 119 \).
3. Calculate \( \Phi(n) = (p-1) \times (q-1) = 6 \times 16 = 96 \).
4. Select e such that e is relatively prime to \( \Phi(n) = 96 \) and less than \( \Phi(n) \) so let us assume that e = 5.
5. Determine d such that ed \equiv 1 \pmod{96} and d < 96. The value of d = 77, as 77 \times 5 = 385 = 4 \times 96 + 1.
6. Public key KU = \{5, 119\}
7. Private Key KR = \{77, 119\}
8. Say M = 19
9. Encryption: \( 19^5 = 2476099 \Rightarrow 2476099 / 119 = 20807 \) with the remainder of 66, so C = 66.
10. Decryption: \( 66^{77} = 1.27\ldots \times 10^{140} \Rightarrow (1.27\ldots \times 10^{140}) / 119 = 1.06\ldots \times 10^{136} \) with remainder of 19, so M = 19.

**Computation aspect of RSA algorithm**

**Key generation:** It consists of the following two tasks:

1. Determine two prime numbers, p and q,
2. Selecting e and calculating the d.

**Determine two prime numbers, p and q-**
First consider the selection of $p$ and $q$ because the value $n = p \times q$ will be known to any potential opponent, to prevent discovery of $p$ and $q$ by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e. $p$ and $q$ must be large numbers ~ 100 digits). But the method used for finding large primes must be reasonably efficient.

The following procedure is usually adopted for picking prime numbers:

1. Pick an odd integer at random using a pseudorandom number generator.
2. Pick an integer $a < n$ at random.
3. Perform the probabilistic primality test, such as Miller-Rabin. If $n$ fails the test, reject $n$ and go to step 1.
4. If $n$ has passed a sufficient number of tests, accept $n$; otherwise go to step 2.

This is tedious procedure but it is performed only when a new key pair \{KU, KR\} in needed.

Cryptographic applications typically make use of algorithmic techniques for random number generation. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness [10]. Such numbers are referred to as pseudorandom numbers. The traditional method of generating random numbers is to pass the results of a pseudorandom number generator through a hash function like MD5 or SHA and use the hash result. This is however much slower and instead faster approach was adopted in the development of this application. It uses the shuffling method as outlined in Computer Generated Random Numbers (http://world.std.com/~franl/crypto/random-numbers.html) by David W. Deley. Additionally it uses around 16 random entropy sources including different pseudorandom number generators. The basic method is outlines below:
(i) Random Number Generation

- Start with an array of dimension around 200.
- Seed all the pseudorandom number generators by cascading one random number to seed to the next one. The first generator is seeded with system time.
- Fill the first 100 elements with the results of the first pseudorandom number generator.
- Fill the next 100 elements with the results of the second pseudorandom number generator.
- When the program wants a random number, randomly choose one from the array and send it to the program.
- Replace the number chosen in the array with a new random number from the main random number generator which uses all the 16 randomness entropy sources.

(ii) Primality Test

The random number values generated are made odd and then checked for primality using the probabilistic Miller-Rabin algorithm [58,100] which is typically used to test a large number for primality [10].

MILLER-RABIN(n, t)

INPUT: an odd number n \geq 3 and security parameter t \geq 1.
OUTPUT: an answer “PRIME” or “COMPOSITE” to the question: “Is n Prime?”

1. Write \( n - 1 = 2^s r \) such that r is odd.
2. For I from 1 to t do the following
   
   2.1. Choose a random number integer a, \( 2 \leq a \leq n - 2 \)
   
   2.2. Compute \( y = a^r \mod n \)
   
   2.3. If \( y \neq 1 \) and \( y \neq n - 1 \) then do the following:

   \( j \leftarrow 1. \)

   While \( j \leq s - 1 \) and \( y \neq n - 1 \) do the following:

   Compute \( y \leftarrow y^2 \mod n \)
If \( y = 1 \) then return ("COMPOSITE")
\[
j \leftarrow j + 1
\]
if \( y \neq n - 1 \) then return ("COMPOSITE")
3. Return("PRIME").

Selecting \( e \) and calculating the \( d \):

(i) Calculation of private key

Having determined prime numbers \( p \) and \( q \), we select \( e \) and calculate \( d \) using the extended Euclid’s algorithm which is used for computing polynomial greatest common divisor and also allows to compute the multiplicative inverse \([101,55]\):

EXTENDED_EUCLID(U,V)

INPUT: 2 nonnegative integers \( u \) and \( v \).

OUTPUT: a vector \( (u_1, u_2, u_3) \) such that \( uu_1 + vv_1 = u_3 = \gcd(u, v) \)

REMARK: Temporary vectors \( (v_1, v_2, v_3) \) and \( (t_1, t_2, t_3) \) are used in such a way that the following hold throughout the calculation:
\[
\begin{align*}
    ut_1 + vt_2 &= t_3 \\
nu_1 + vu_2 &= u_3 \\
    uv_1 + uv_2 &= v_3
\end{align*}
\]

1. Initialize: set \( (u_1, u_2, u_3) \leftarrow (1, 0, u) \) and \( (v_1, v_2, v_3) \leftarrow (0, 1, v) \)
2. If \( v_3 = 0 \) stop.
3. Set \( q \leftarrow \frac{u_3}{v_3} \) and then set:
\[
\begin{align*}
    (t_1, t_2, t_3) &\leftarrow (u_1, u_2, u_3) - (v_1, v_2, v_3)q \\
    (u_1, u_2, u_3) &\leftarrow (v_1, v_2, v_3) \\
    (v_1, v_2, v_3) &\leftarrow (t_1, t_2, t_3)
\end{align*}
\]
Return to step 2

Encryption & Decryption

Both encryption and decryption involve raising an integer to an integer power, \( \mod n \), for which we make use of the following property of modular arithmetic:
\[
[(a \mod n) \times (b \mod n)] \mod n = (a \times b) \mod n
\]
Now suppose we wish to find \( x^{16} \), we can repeatedly take the square of each partial result successively forming \( x^2, x^4, x^8 \) and \( x^{16} \). This method is used as illustrated below to calculate \( a^b \mod n \).

**Repeated Square and Multiply Algorithm for Exponentiation**

**INPUT:** \( a \) and an integer \( 0 \leq k < n \) whose binary representation is \( k = \sum_{i=0}^{t} k_i 2^i \).

**OUTPUT:** \( a^k \mod n \).

1. Set \( b \leftarrow 1 \). If \( k = 0 \) then return \( b \).
2. Set \( A \leftarrow a \).
3. If \( k_0 = 1 \) then set \( b \leftarrow a \).
4. For I from 1 to \( t \) do the following:
   4.1. Set \( A \leftarrow A^2 \mod n \).
   4.2. If \( k_i = 1 \) then set \( b \leftarrow A \cdot b \mod n \).
5. Return \( b \).

**Practical applications of the RSA algorithm**

To this day the RSA together with the AES algorithm is the most used algorithm in commercial systems. It is used:

(i) to protect web traffic, in the SSL protocol (Security Socket Layer)

(ii) to guarantee email privacy and authenticity in PGP (Pretty Good Privacy)

(iii) to guarantee remote connection in SSH (Secure Shell).

(iv) Furthermore it plays an important role in the modern payment systems through SET protocol (Secure Electronic Transaction).

**Security of RSA algorithm**

The security of the RSA cryptosystem is based on two mathematical problems: the problem of factoring large numbers and the RSA problem. Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for
solving them. Providing security against partial decryption may require the addition of a secure padding scheme.

According to William Stallings [10] Four possible approaches to attacking the RSA algorithm are:

- **Brute force**: This involves trying all possible private keys.
- **Mathematical attacks**: There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks**: These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks**: This type of attack exploits properties of the RSA algorithm.

The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, to use a large key space. Thus, the larger the number of bits in \(d\), the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run. Thus, we need to be careful in choosing a key size for RSA, a key size in the range of 1024 to 2048 bits seems reasonable.

In fact, these issues have solutions; the only downside is that any device implementing RSA would have to have much more hardware and software to counter certain types of attacks or attempts at eavesdropping.

A very major threat to RSA would be a solution to the Riemann hypothesis [102]. Thus a solution has neither been proven to exist nor to not exist. Development on the Riemann hypothesis is currently relatively stagnant. However, if a solution were found, prime numbers would be too easy to find, and RSA would fall apart.
3.3.2 *Advantages of public key cryptosystems*

1. Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).
2. The administration of keys on a network requires the presence of only a functionally trusted TTP as opposed to an unconditionally trusted TTP. Depending on the mode of usage, the TTP might only be required in an “off-line” manner, as opposed to in real time.
3. Depending on the mode of usage, a private key/public key pair may remain unchanged for considerable periods of time, e.g., many sessions (even several years).
4. Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.
5. In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

3.3.3 *Disadvantages of public key cryptosystems*

1. Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric key schemes.
2. Key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.
3. No public-key scheme has been proven to be secure (the same can be said for block ciphers). The most effective public-key encryption schemes found to date have their security based on the presumed difficulty of a small set of number-theoretic problems.
4. Public-key cryptography does not have as extensive a history as symmetric-key encryption, being discovered only in the mid-1970.
3.4 Comparison of Symmetric and Public-Key Cryptosystems

However, the public-key algorithm is not a perfect substitute for the symmetric algorithm because they have their own unique problems-
1. They are very slow as symmetric algorithms have been observed to perform 100 times faster than asymmetric-key algorithm [7].
2. They found vulnerable to chosen-plaintext attacks.

Table 16: Comparison of Symmetric and public-key cryptosystem

<table>
<thead>
<tr>
<th>Features</th>
<th>Private key Cryptosystem</th>
<th>Public key Cryptosystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of keys</td>
<td>Single key</td>
<td>Pair of keys</td>
</tr>
<tr>
<td>Types of keys</td>
<td>Key is secret</td>
<td>One key is private (Secret) while other is public</td>
</tr>
<tr>
<td>Protection of keys</td>
<td>Disclosure and modification</td>
<td>Disclosure and modification for private keys and modification of public keys.</td>
</tr>
<tr>
<td>Relative Speeds</td>
<td>Faster</td>
<td>Slower</td>
</tr>
</tbody>
</table>

3.5 KEM–DEM Hybrid Encryption Schemes

If we define a hybrid encryption scheme as an asymmetric encryption scheme which makes use of some keyed symmetric cryptography as a “black-box” then it is very difficult to make any statements about the security of hybrid encryption schemes as there are far too many ways in which the symmetric scheme could be used within the cryptosystem. However, almost all the practical examples of hybrid encryption schemes [93,94,29,103] are based on a simple idea:

1. an asymmetric section generates a suitable symmetric key and encrypts that key using an asymmetric encryption scheme, and
2. a symmetric section encrypts the message using the randomly generated symmetric key.
Cramer and Shoup [93] have shown that these two parts of a hybrid encryption scheme can be separated and security criteria defined for both sections individually. The asymmetric section is known as the key encapsulation mechanism or KEM. The symmetric section is known as the data encapsulation mechanism or DEM. The beauty of the KEM–DEM construction is that it allows us to consider KEMs and DEMs separately [34].

A Key Encapsulation Mechanism (KEM) is a triple of algorithm consisting of:

1. The key generation algorithm, \( \text{Gen} \), which takes as input a security parameter \( 1^k \) and outputs a public/private key pair \( (pk, sk) \).
2. A probabilistic encapsulation algorithm, \( \text{Encap} \), which takes as input a public key \( pk \) and outputs a key \( K \) and an encapsulation of that key \( C \). We denote this as \( (K, C) = \text{Encap}(pk) \).
3. A deterministic decapsulation algorithm, \( \text{Decap} \), which takes as inputs the private key \( pk \) and an encapsulation \( C \), and outputs a symmetric key \( K \) or the error symbol \( \bot \). We denote this as \( K = \text{Decap}(sk, C) \).

Hence, KEMs are very similar to asymmetric encryption schemes; the only difference is that, unlike an encryption algorithm, the KEM’s encapsulation algorithm does not take any form of message as input but rather randomly generates its own “message” — the symmetric key \( K \). Just as with an asymmetric encryption scheme, it is important that a KEM satisfies a correctness property, i.e. that for all public/private key pair \( (pk, sk) \) we have that \( K = \text{Decap}(C, sk) \) for all \( (K, C) = \text{Encap}(pk) \).

The security criterion for a KEM [34] is phrased in terms of a game played between a hypothetical challenger and two-stage attacker \( A = (A_1, A_2) \). The attack goal for a KEM is to distinguish the real key corresponding to an encapsulation from a randomly generated key. This is known as the IND-CCA game and, for a given security parameter \( k \), works as follows:
1. The challenger generates a valid public/private key pair \((pk, sk)\) by running \(Gen(1^k)\).

2. The attacker runs \(A1\) on the input \(pk\). It terminates by outputting some state information \(state\). During its execution \(A1\) may query a decapsulation oracle that will, when given an encapsulation \(C\), return \(Decap(sk,C)\).

3. The challenger generates a valid encapsulation \((K_0,C^*)\) by running \(Encap(pk)\). It also generates a random key \(K_1\) of the same length as \(K_0\). Next it chooses a bit \(b \in \{0, 1\}\) uniformly at random and sets \(K^* = K_b\). The challenge encapsulation is \((K^*,C^*)\).

4. The attacker runs \(A2\) on the input \((K^*,C^*)\) and \(state\). It terminates by outputting a guess \(b'\) for \(b\). Again, during its execution \(A2\) may query a decapsulation oracle that will, when given an encapsulation \(C \neq C^*\), return \(Decap(sk,C)\).

The attacker wins the game if \(b = b'\). The attacker’s advantage is defined to be:

\[|Pr[b = b'] - 1/2|\]

A DEM is a pair of algorithms consisting of [34]:

1. A deterministic encryption algorithm, \(Enc\), which takes as input a message \(m \in \{0, 1\}^*\) of any length and a symmetric key \(K\) of some predetermined length. It outputs an encryption \(C = Enc_K(m)\).

2. A deterministic decryption algorithm, \(Dec\), which takes as input an encryption \(C \in \{0, 1\}^*\) and a symmetric key \(K\) of some predetermined length, and outputs either a message \(m \in \{0, 1\}^*\) or the error symbol \(\bot\).

Hence, DEMs are very similar to symmetric encryption schemes. Again, a DEM must satisfy a correctness property: that for every key \(K\) of the correct length, \(m = Dec_K(Enc_K(m))\).

It should now be easy to see that a KEM and a DEM can be ‘slotted together’ to form an asymmetric encryption algorithm. If \((Gen, Encap, Decap)\)
is a KEM and (Enc, Dec) is a DEM, and for any security parameter $k$ the length of the symmetric keys that are output by the KEM is equal to the length of the symmetric keys taken as input by the DEM, then we can form an asymmetric encryption scheme as follows:

- The key generation algorithm is given by Gen.
- The encryption algorithm for a message $m$ under a public key $pk$ is given by:
  1. Set $(K, C_1) = Encap(pk)$.
  2. Set $C_2 = Enc_K(m)$.
  3. Output $(C_1, C_2)$.
- The decryption algorithm for a ciphertext $C = (C_1, C_2)$ under a private key $sk$ is given by:
  1. Set $K = Decap(sk, C_1)$. If $K = \bot$ then output $\bot$ and stop.
  2. Set $m = Dec_K(C_2)$. If $m = \bot$ then output $\bot$ and stop.
  3. Output $m$.

Hybrid cryptography can take into account security and effectiveness in the application of cryptography. So it is increasingly being used in the real-world scenarios.

### 3.6 Developed hybrid encryption algorithm

In the real world, key management is the hardest part of cryptography. Keeping the keys secret is much harder. Cryptanalysts often attack both symmetric and public key cryptosystems through their key management. So the security of a cryptographic algorithm rests in the key. If someone is using a cryptographically weak process to generate keys then the whole system is weak [7,14,104]. Soundness of Cryptosystem relies upon two basic factors: (1) algorithm and (2) key. The algorithm is a mathematical function, and the key is a parameter used by that function. A key is often easier to protect because it
Hybrid Cryptography

is a small piece of information, and easier to change if compromised. Thus, the security of an encryption system in most cases relies on some key being kept secret but practically it is very difficult. An attacker who obtains the key by theft, extortion, dumpster dives or social engineering can recover the original message from the encrypted data.

Generally both the public-key and private-key of sender and receiver are saved on their computers. If someone gets unauthorized access to their computers then he will get their keys and break the security hurdles. Protection of public key is not much necessary because public-keys are shared to all the network members so that they can encrypt the message by using the public-key of the receiver but the protection of private-key is very necessary since the receiver decrypts the encrypted message using his private-key. So it is important to protect the private-key from unauthorized access. To do so hybrid cryptography is being used in the developed application. In the developed application DES algorithm of symmetric cryptography is used to encrypt the private-key so that in case the computer is hacked even though the hacker cannot access the private-key. The DES algorithm is used because it is very fast in operation so the overall encryption-decryption process remains faster and efficient. During the decryption process the DES rapidly decrypt the private-key and the decrypted private-key decrypts the received encrypted message quickly.

A hybrid encryption algorithm has the advantages of both the symmetric and asymmetric algorithms which enhance the speed as well as the security aspects up to a remarkable point. So implementation of this mechanism in the developed application RSAAPP involves the following steps:

1. Generate key pair i.e. Public key $K_U = \{e, n\}$ value and Private Key $K_R = \{d, n\}$ using RSA key generation
2. Save the Public Key $\{e, n\}$.
(3) The generated Private Key \(\{d, n\}\) are encrypted using DES algorithm of symmetric cryptography after taking 8-character password as user input and then Save it in a file for better security.

(4) Now encrypt the message by using the Public Key \(K_U\).

(5) At the time of decryption, first the private key is decrypted and then the encrypted message can be decrypted using private key \(K_R\).

The complete process can be viewed in the Figure 13

![Figure 13: Hybrid Cryptography in RSAAPP](image)

In the example of illustration of RSA algorithm the Public key, \(K_U\), was \(\{5, 119\}\) and the Private-key, \(K_R\), was \(\{77, 119\}\). Now in the developed application when save the private key then the application ask the user to input an 8-character password. This 8-character password, say 10101010, works as a key and DES algorithm uses this key to encrypt the private-key i.e. 77 to 152. Now the private key is enciphered and safe because if hacker hacks the
computer he cannot access the private key. In the decryption process the application decrypt 152 to 77 using key 10101010 by DES algorithm.

Cryptanalysts often attack both symmetric and public key cryptosystems through their key management. So the security of a cryptographic algorithm rests in the key. Therefore in the developed application the key generation process is made modular, efficient and fast enough so that it can generate the highly secured key-pairs in reasonable time and then the generated private key is being protected using DES which provides the advantages of hybrid cryptosystem in the RSAAPP.

**Chapter Summary**

At the end this chapter it is worth mentioning that the study of hybrid cryptography going to present is based on the study of hybridization. Study has been performed from 1993 to 2013. All the concepts related to hybrid cryptography have been analyzed and reached at the conclusion that the use of RSA is quite expanding. A lot of research papers have been searched and reviewed related to the problem and found that the major scope of work has been done using RSA [97].

The Data Encryption Standard is one of the first commercially developed ciphers. DES is the result of efforts done by IBM (International Business Machines) corporation, NBS (National Bureau of Standards) and NSA (National Security Agency). DES is a block cipher that encrypts 64-bit data blocks and encryption of the data is performed using a 56-bit secret key. DES consists of sixteen rounds and two permutation layers. DES uses a shared key both to encrypt and decrypt the message. The decryption process is the reverse of encryption process. DES possesses strong Avalanche effect and is flexible as it works in CBC, ECB, CFB and OFB modes.

RSA is a strong encryption algorithm that has stood a partial test of time. RSA implements a public-key cryptosystem that allows secure
communications and “digital signatures”, and its security rests in part on the difficulty of factoring large numbers. The authors urged anyone to attempt to break their code, whether by factorization techniques or otherwise, and nobody to date seems to have succeeded if the key size is large. This has in effect certified RSA, and will continue to assure its security for as long as it stands the test of time against such break-ins. The solution of Riemann hypothesis made primes numbers easy to find and RSA would fall apart. Undoubtedly, much more sophisticated algorithms than RSA will continue to be developed as mathematicians discover more in the fields of number theory and cryptanalysis.

The result analysis of the developed application RSAAPP shows that it is a high speed and efficient application up to a remarkable point in comparison to the other applications available related to the problem.