Chapter 4 The Data Encryption Standard
Most widely used encryption scheme is based on DES adopted by National Bureau of Standards (now National Institute of Standards and Technology) in 1977
Most widely used encryption scheme is based on DES adopted by National Bureau of Standards (now National Institute of Standards and Technology) in 1977.

Algorithm known as Data Encryption Algorithm (DEA)
Most widely used encryption scheme is based on DES adopted by National Bureau of Standards (now National Institute of Standards and Technology) in 1977.

Algorithm known as Data Encryption Algorithm (DEA).

Data is encrypted in 64-bit blocks using a 56-bit key.
Most widely used encryption scheme is based on DES adopted by National Bureau of Standards (now National Institute of Standards and Technology) in 1977

Algorithm known as Data Encryption Algorithm (DEA)

Data is encrypted in 64-bit blocks using a 56-bit key

Output is also 64 bits in size
Most widely used encryption scheme is based on DES adopted by National Bureau of Standards (now National Institute of Standards and Technology) in 1977

Algorithm known as Data Encryption Algorithm (DEA)

Data is encrypted in 64-bit blocks using a 56-bit key

Output is also 64 bits in size

The DES is widely used, but has also been the subject of controversy about how secure it is. Let’s do a quick history lesson on the DES so we can appreciate the nature of this controversy.
In the late 1960’s, IBM set up a research project in computer technology led by Horst Feistel.
History of DES

- In the late 1960’s, IBM set up a research project in computer technology led by Horst Feistel
- Conclusion: creation of LUCIFER in 1971
In the late 1960’s, IBM set up a research project in computer technology led by Horst Feistel.

Conclusion: creation of LUCIFER in 1971

Sold to Lloyd’s of London for use in cash dispensing system.
History of DES

- In the late 1960’s, IBM set up a research project in computer technology led by Horst Feistel
- Conclusion: creation of LUCIFER in 1971
- Sold to Lloyd’s of London for use in cash dispensing system
- LUCIFER is a Feistel block cipher operating on 64 bits with a key of 128 bits
In the late 1960’s, IBM set up a research project in computer technology led by Horst Feistel

Conclusion: creation of LUCIFER in 1971

Sold to Lloyd’s of London for use in cash dispensing system

LUCIFER is a Feistel block cipher operating on 64 bits with a key of 128 bits

IBM wanted a more marketable product that could fit on a single chip
In the late 1960’s, IBM set up a research project in computer technology led by Horst Feistel

Conclusion: creation of LUCIFER in 1971

Sold to Lloyd’s of London for use in cash dispensing system

LUCIFER is a Feistel block cipher operating on 64 bits with a key of 128 bits

IBM wanted a more marketable product that could fit on a single chip

1973: IBM submitted this to NBS during search for national encryption standard
In the late 1960’s, IBM set up a research project in computer technology led by Horst Feistel.

Conclusion: creation of LUCIFER in 1971.

Sold to Lloyd’s of London for use in cash dispensing system.

LUCIFER is a Feistel block cipher operating on 64 bits with a key of 128 bits.

IBM wanted a more marketable product that could fit on a single chip.

1973: IBM submitted this to NBS during search for national encryption standard.

Accepted and adopted as DES in 1977.
Problems

Before it was adopted, however, the proposed DES was subject to intense scrutiny, which still has not subsided today.
Problems

Before it was adopted, however, the proposed DES was subject to intense scrutiny, which still has not subsided today.

1. The key length of the original LUCIFER algorithm was 128 bits, but that of the proposed system was only 56 bits (every 8th bit is used as a parity check, reducing the length of the key that is used from 64 to 56). Critics feared that the length was too short to withstand brute force attack.
Before it was adopted, however, the proposed DES was subject to intense scrutiny, which still has not subsided today.

1. The key length of the original LUCIFER algorithm was 128 bits, but that of the proposed system was only 56 bits (every 8\textsuperscript{th} bit is used as a parity check, reducing the length of the key that is used from 64 to 56). Critics feared that the length was too short to withstand brute force attack.

2. The design criteria for the internal structure of the DES, the S-boxes, were classified. So, users in this system could not be sure that the internal structure of the DES was free of any hidden weak points and would enable the NSA to decrypt messages without the benefit of a key.
Problems

Before it was adopted, however, the proposed DES was subject to intense scrutiny, which still has not subsided today.

1. The key length of the original LUCIFER algorithm was 128 bits, but that of the proposed system was only 56 bits (every 8\textsuperscript{th} bit is used as a parity check, reducing the length of the key that is used from 64 to 56). Critics feared that the length was too short to withstand brute force attack.

2. The design criteria for the internal structure of the DES, the S-boxes, were classified. So, users in this system could not be sure that the internal structure of the DES was free of any hidden weak points and would enable the NSA to decrypt messages without the benefit of a key.

IBM participants have said that the only changes that had been made to the proposal were changes to the S-boxes, suggested by the NSA, that removed vulnerabilities identified during the evaluation process.
Usage Today

- DES still used in financial applications
Usage Today

- DES still used in financial applications
- NIST (1999) issued a new version, the triple DES
Usage Today

- DES still used in financial applications
- NIST (1999) issued a new version, the triple DES
- They say DES should only be used for legacy systems
DES still used in financial applications
NIST (1999) issued a new version, the triple DES
They say DES should only be used for legacy systems

So, DES is ‘semi-obsolete’, but is worth looking at to make it clear that it is not easy to understand.
Horst Feistel was one of the first non-military researchers in the field of cryptography and can be considered the father of modern block ciphers.
Horst Feistel was one of the first non-military researchers in the field of cryptography and can be considered the father of modern block ciphers.

In 1973 he published an article with the title ‘Cryptography and Computer Privacy’ in a magazine called ‘Scientific American’, in which he tried to cover the most important aspects of machine encryption and introduced what is today known as the Feistel Network.
Horst Feistel was one of the first non-military researchers in the field of cryptography and can be considered the father of modern block ciphers.

In 1973 he published an article with the title ‘Cryptography and Computer Privacy’ in a magazine called ‘Scientific American’, in which he tried to cover the most important aspects of machine encryption and introduced what is today known as the Feistel Network.

A Feistel network is a cryptographic technique used in the construction of block cipher-based algorithms and mechanisms. A Feistel network is also known as a Feistel cipher.
A Feistel network implements a series of iterative ciphers on a block of data and is generally designed for block ciphers that encrypt large quantities of data.

- Split data into two equal pieces
A Feistel network implements a series of iterative ciphers on a block of data and is generally designed for block ciphers that encrypt large quantities of data.

- Split data into two equal pieces
- Apply encryption in multiple rounds
A Feistel network implements a series of iterative ciphers on a block of data and is generally designed for block ciphers that encrypt large quantities of data.

- Split data into two equal pieces
- Apply encryption in multiple rounds
- Each round implements permutations and combinations derived from a primary key or function
A Feistel network implements a series of iterative ciphers on a block of data and is generally designed for block ciphers that encrypt large quantities of data.

- Split data into two equal pieces
- Apply encryption in multiple rounds
- Each round implements permutations and combinations derived from a primary key or function
- Number of rounds varies for each cipher implementing a Feistel network
Feistel Networks

A Feistel network implements a series of iterative ciphers on a block of data and is generally designed for block ciphers that encrypt large quantities of data.

- Split data into two equal pieces
- Apply encryption in multiple rounds
- Each round implements permutations and combinations derived from a primary key or function
- Number of rounds varies for each cipher implementing a Feistel network
- Feistel ciphers are also symmetric and sometimes the exact same key is used to encrypt and decrypt.
A Feistel network implements a series of iterative ciphers on a block of data and is generally designed for block ciphers that encrypt large quantities of data.

- Split data into two equal pieces
- Apply encryption in multiple rounds
- Each round implements permutations and combinations derived from a primary key or function
- Number of rounds varies for each cipher implementing a Feistel network
- Feistel ciphers are also symmetric and sometimes the exact same key is used to encrypt and decrypt.

DES encryption consists of 16 rounds, which means repetition of a similar process. Each round is a Feistel network, which is guaranteed to be invertible and to be its own inverse.
Fix a positive integer $n$, in this case $n = 32$. 
Fix a positive integer $n$, in this case $n = 32$.

Given a string of $2n$ bits, group them in two parts, the left and the right halves (L and R).
Idea of the DES

- Fix a positive integer $n$, in this case $n = 32$.
- Given a string of $2n$ bits, group them in two parts, the left and the right halves (L and R).
- We can view L and R as vectors of length $n$ with entries reduced modulo 2.
Idea of the DES

- Fix a positive integer $n$, in this case $n = 32$.
- Given a string of $2n$ bits, group them in two parts, the left and the right halves (L and R).
- We can view L and R as vectors of length $n$ with entries reduced modulo 2.
- Let $f$ be any function at all that accepts as inputs $n$ bits and produces an output of $n$ bits. The corresponding Feistel network $F_j$ takes the $2n$-bit pieces L and R as inputs and produces $2n$ bits of output by

$$F_j(L, R) = (L \oplus f(R), R)$$

where the $\oplus$ used here means vector (component-wise) addition and then reduces modulo 2.
Idea of the DES

Example

\[(1, 1, 1, 0, 0) \oplus (1, 0, 1, 1, 1)(\text{mod } 2) = (0, 1, 0, 1, 1)\]
Idea of the DES

Example

\[(1, 1, 1, 0, 0) \oplus (1, 0, 1, 1, 1)(\text{mod } 2) = (0, 1, 0, 1, 1)\]

The key property of a Feistel network is that if you do the same thing twice with the same \(f\), you get back the same thing.

\[F_f(F_f(L, R)) = F_f(L \oplus f(R), R) = ((L \oplus f(R)) \oplus f(R), R) = (L, R)\]
Idea of the DES

Example

\[(1, 1, 1, 0, 0) \oplus (1, 0, 1, 1, 1) \pmod{2} = (0, 1, 0, 1, 1)\]

The key property of a Feistel network is that if you do the same thing twice with the same \(f\), you get back the same thing.

\[F_f(F_f(L, R)) = F_f(L \oplus f(R), R) = ((L \oplus f(R)) \oplus f(R), R) = (L, R)\]

So, no matter how bizarre or complex this function \(f\) is, we don’t have to worry about invertibility or about finding the inverse. If we repeat this process with some simple mixing in-between, using some sort of tricky function \(f\) dependent on the key, then we would do what a DES does.
Overall Scheme of the DES

As with any encryption scheme, there are two inputs to the encryption function, the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.
Looking at the left hand side, we see that the plaintext proceeds in three phases.
Looking at the left hand side, we see that the plaintext proceeds in three phases.

1. The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.
Looking at the left hand side, we see that the plaintext proceeds in three phases.

1. The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.

2. Next is a phase consisting of 16 rounds of the same function which involves both permutations and substitution functions.
Looking at the left hand side, we see that the plaintext proceeds in three phases.

1. The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.

2. Next is a phase consisting of 16 rounds of the same function which involves both permutations and substitution functions.
   - The output of the $16^{th}$ round consists of 64 bits that are a function of the input plaintext and the key.
Overall Scheme of the DES

Looking at the left hand side, we see that the plaintext proceeds in three phases.

1. The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.

2. Next is a phase consisting of 16 rounds of the same function which involves both permutations and substitution functions.
   1. The output of the 16th round consists of 64 bits that are a function of the input plaintext and the key.
   2. The left and right halves are swapped to produce the pre-output.

With the exception of the initial and final permutations, DES has the exact same structure of a Feistel cipher.
Overall Scheme of the DES

Looking at the left hand side, we see that the plaintext proceeds in three phases.

1. The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.

2. Next is a phase consisting of 16 rounds of the same function which involves both permutations and substitution functions.
   - The output of the 16\(^{th}\) round consists of 64 bits that are a function of the input plaintext and the key.
   - The left and right halves are swapped to produce the pre-output.

3. Finally, the pre-output is passed through a permutation ($IP^{-1}$) that is the inverse of the initial permutation function to produce the 64-bit ciphertext.
Looking at the left hand side, we see that the plaintext proceeds in three phases.

1. The 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.
2. Next is a phase consisting of 16 rounds of the same function which involves both permutations and substitution functions.
   1. The output of the 16th round consists of 64 bits that are a function of the input plaintext and the key.
   2. The left and right halves are swapped to produce the pre-output.
3. Finally, the pre-output is passed through a permutation ($IP^{-1}$) that is the inverse of the initial permutation function to produce the 64-bit ciphertext.

With the exception of the initial and final permutations, DES has the exact same structure of a Feistel cipher.
The right-hand portion shows the way in which the 56-bit key is used.

1. Initially the key is passed through a permutation function.
The right-hand portion shows the way in which the 56-bit key is used.

1. Initially the key is passed through a permutation function.
2. Then, for each of the 16 rounds, a subkey $K_i$ is produced by the combination of the left circular shift and a permutation.
Overall Scheme of the DES

The right-hand portion shows the way in which the 56-bit key is used.

1. Initially the key is passed through a permutation function.
2. Then, for each of the 16 rounds, a subkey \( K_i \) is produced by the combination of the left circular shift and a permutation.
3. The permutation function is the same for each round but a different subkey is produced because of the repeated shifts of the key bits.
### Table 1: Initial Permutation (IP)

<table>
<thead>
<tr>
<th>58</th>
<th>50</th>
<th>42</th>
<th>34</th>
<th>26</th>
<th>18</th>
<th>10</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>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>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>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>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>61</td>
<td>52</td>
<td>45</td>
<td>37</td>
<td>29</td>
<td>21</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<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>
### DES Permutation Tables

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

Table 2: Inverse Initial Permutation ($IP^{-1}$)
## DES Permutation Tables

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

Expansion Permutation $(E)$
### DES Permutation Tables

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

**Permutation Function** ($P$)
### DES Permutation Tables

<table>
<thead>
<tr>
<th></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>
</tr>
</thead>
<tbody>
<tr>
<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>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>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>
# DES Permutation Tables

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

Input Key
# DES Permutation Tables

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

Permuted Choice 1 ($PC - 1$)
### DES Permutation Tables

#### Permuted Choice 2 ($PC - 2$)

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>17</td>
<td>11</td>
<td>24</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>21</td>
<td>10</td>
<td>23</td>
<td>19</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>8</td>
<td>16</td>
<td>7</td>
<td>27</td>
<td>20</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>41</td>
<td>52</td>
<td>31</td>
<td>37</td>
<td>47</td>
<td>55</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>51</td>
<td>45</td>
<td>33</td>
<td>48</td>
<td>44</td>
<td>49</td>
<td>39</td>
<td>56</td>
</tr>
<tr>
<td>34</td>
<td>53</td>
<td>46</td>
<td>42</td>
<td>50</td>
<td>36</td>
<td>29</td>
<td>32</td>
</tr>
</tbody>
</table>

This table represents the Permuted Choice 2 ($PC - 2$) permutation of the DES (Data Encryption Standard) algorithm.
### Schedule of Left Shifts

<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>Bits Rotated</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The input to a table consists of 64 bits numbered from 1-64. The 64 entries in the permutation table contain a permutation of the numbers from 1-64. There is a pattern in each row, the value decreases by 8 and wraps around if they reach 0. But, this pattern is not true from row to row.
The input to a table consists of 64 bits numbered from 1-64. The 64 entries in the permutation table contain a permutation of the numbers from 1-64. There is a pattern in each row, the value decreases by 8 and wraps around if they reach 0. But, this pattern is not true from row to row.

The meaning of the notation is that the $58^{th}$ bit of the key goes into the first bit of the rearrangement, the $50^{th}$ bit of the key goes into the second, etc.
The input to a table consists of 64 bits numbered from 1-64. The 64 entries in the permutation table contain a permutation of the numbers from 1-64. There is a pattern in each row, the value decreases by 8 and wraps around if they reach 0. But, this pattern is not true from row to row.

The meaning of the notation is that the $58^{th}$ bit of the key goes into the first bit of the rearrangement, the $50^{th}$ bit of the key goes into the second, etc.

To see that the first two are inverses of each other, notice the position of the 1 in IP and the value in the 1, 1 position in $IP^{-1}$...
Details of a Single Round

Here, we will look at the internal structure of a single round.
Begin by focusing on the LHS of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled $L$ and $R$. 

As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

The round key $K_i$ is 48 bits. The $R$ input is first expanded to 48 bits by using a table that defines a permutation plus and expansion that involves duplication of 16 of the $R$ bits (table E). The resulting 48 bits are Xored with $K_i$. This 48 bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by table $P$. 

Details of a Single Round
Begin by focusing on the LHS of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled $L$ and $R$.

As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

\[
L_i = R_{i-1}
\]
\[
R_i = L_{i-1} \oplus F(R_{i-1}, K_i)
\]
Details of a Single Round

Begin by focusing on the LHS of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled $L$ and $R$.

As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

\[
L_i = R_{i-1} \\
R_i = L_{i-1} \oplus F(R_{i-1}, K_i)
\]

The round key $K_i$ is 48 bits. The $R$ input is first expanded to 48 bits by using a table that defines a permutation plus and expansion that involves duplication of 16 of the $R$ bits (table E). The resulting 48 bits are Xored with $K_i$. This 48 bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by table P.
The role of the S-boxes (substitution boxes) are to give the security to the DES. They are also the most confusing part.
There are 8 $S$-boxes, each of which takes a 6-bit input and produces a 4-bit output. The 48 bits are broken into 8 pieces of 6 bits and fed to the 8 $S$-boxes. (The first 6 bits are acted upon by the first $S$-box, the next 6 by the second, etc.). The outputs are stuck back together to again give a 32-bit total output.
Details of a Single Round

There are 8 S-boxes, each of which takes a 6-bit input and produces a 4-bit output. The 48 bits are broken into 8 pieces of 6 bits and fed to the 8 S-boxes. (The first 6 bits are acted upon by the first S-box, the next 6 by the second, etc.). The outputs are stuck back together to again give a 32-bit total output.

Each of the S-boxes can be described by a table with 4 rows and 16 columns. Each entry in the table is a 4-bit number, meaning it is in the range 0-15, which when written in binary, will be the output of the S-box. The 6-bit input to the S-box specifies the row and column as follows:
Details of a Single Round

There are 8 S-boxes, each of which takes a 6-bit input and produces a 4-bit output. The 48 bits are broken into 8 pieces of 6 bits and fed to the 8 S-boxes. (The first 6 bits are acted upon by the first S-box, the next 6 by the second, etc.). The outputs are stuck back together to again give a 32-bit total output.

Each of the S-boxes can be described by a table with 4 rows and 16 columns. Each entry in the table is a 4-bit number, meaning it is in the range 0-15, which when written in binary, will be the output of the S-box. The 6-bit input to the S-box specifies the row and column as follows:

Let the 6 bits be \( b_1, b_2, \cdots, b_6 \). Then, (Note: these are the binary expansions)

\[
\text{row} = 2 \cdot b_1 + b_6 \\
\text{column} = 8 \cdot b_2 + 4 \cdot b_3 + 2 \cdot b_4 + b_5
\]

where the indexing of rows and columns starts in the upper left and begins with 0.
For example, the 6 bits 011001 would specify row 01 → 1 and the column 1100 → 12. The value in row 1, column 12 is 9, so the output is 1001.
Details of a Single Round

For example, the 6 bits 011001 would specify row 01 $\rightarrow$ 1 and the column 1100 $\rightarrow$ 12. The value in row 1, column 12 is 9, so the output is 1001.

Each row of an $S$-box defines a general reversible substitution.
For example, the 6 bits 011001 would specify row 01 → 1 and the column 1100 → 12. The value in row 1, column 12 is 9, so the output is 1001.

Each row of an S-box defines a general reversible substitution.

Ignore for a moment the contribution of the key $K_i$. If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits and then become groups of 6 bits by taking the outer bits from the two adjacent groups.
For example, if part of the input word is

\[ efgh \ i j k l \ m n o p \]

This becomes

\[ defghi \ hijklm \ lmnopq \ldots \]
Details of a Single Round

For example, if part of the input word is

\[ efg h \ i j k l \ m n o p \]

This becomes

\[ d e f g h i \ j k l m \ l m n o p q \cdots \]

The outer two bits of each group select one of four possible substitutions (one row of the \( S \)-box). Then a 4-bit output value is substituted for a 4-bit input value (the middle 4 input bits). The 32-bit output from the 8 \( S \)-boxes is then permuted, so that on the next round, the output from each \( S \)-box immediately affects as many others as possible.
Returning to our first and second diagrams, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered 1-64; every 8th bit is ignored (separated off).
Returning to our first and second diagrams, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered 1-64; every $8^{th}$ bit is ignored (separated off).

The key is the first subjected to a permutation governed by $PC - 1$. The resulting 56-bit key is then treated as 2 28-bit quantities, labeled $C_0$ and $D_0$. At each round, $C_{i-1}$ and $D_{i-1}$ are separately subjected to a circular left rotation of 1 or 2 bits as given in the schedule.
Returning to our first and second diagrams, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered 1-64; every 8th bit is ignored (separated off).

The key is the first subjected to a permutation governed by $PC - 1$. The resulting 56-bit key is then treated as 2 28-bit quantities, labeled $C_0$ and $D_0$. At each round, $C_{i-1}$ and $D_{i-1}$ are separately subjected to a circular left rotation of 1 or 2 bits as given in the schedule.

These shifted values serve as inputs for the next round as well as the input to the part labeled Permutation Choice 2, which produces a 48-bit output that serves as the input to the function $F(R_{i-1}, K_i)$. 
One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. We will talk of the technique and the applicability to DES.
One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. We will talk of the technique and the applicability to DES.

**History**

- Differential cryptanalysis was not reported in open literature until 1990.
Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. We will talk of the technique and the applicability to DES.

History

- Differential cryptanalysis was not reported in open literature until 1990.
- The most publicized results for this approach have been those that have application to DES.
Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. We will talk of the technique and the applicability to DES.

History

- Differential cryptanalysis was not reported in open literature until 1990.
- The most publicized results for this approach have been those that have application to DES.
- Differential cryptanalysis is the first published attack capable of breaking DES in less than $2^{55}$ encryptions.
Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. We will talk of the technique and the applicability to DES.

History

- Differential cryptanalysis was not reported in open literature until 1990.
- The most publicized results for this approach have been those that have application to DES.
- Differential cryptanalysis is the first published attack capable of breaking DES in less than $2^{55}$ encryptions.
- This scheme can successfully cryptanalyze DES with an effort on the order of $2^{47}$ encryptions, requiring $2^{47}$ plaintexts.
Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. We will talk of the technique and the applicability to DES.

**History**

- Differential cryptanalysis was not reported in open literature until 1990.
- The most publicized results for this approach have been those that have application to DES.
- Differential cryptanalysis is the first published attack capable of breaking DES in less than $2^{55}$ encryptions.
- This scheme can successfully cryptanalyze DES with an effort on the order of $2^{47}$ encryptions, requiring $2^{47}$ plaintexts.
- Whereas $2^{47}$ is significantly smaller than $2^{55}$, finding $2^{47}$ plaintexts makes this attack only of theoretic interest.
So, this powerful method doesn’t do very well against DES. The reason is that the IBM team knew of differential cryptanalysis and strengthened DES against this type of attack when constructing the S-boxes.
So, this powerful method doesn’t do very well against DES. The reason is that the IBM team knew of differential cryptanalysis and strengthened DES against this type of attack when constructing the S-boxes.

Differential cryptanalysis is very complex. The rationale is observing the behavior of pairs of text blocks evolving along each round of the cipher instead of observing the evolution of a single block of text.