Parul® University

NAAC A++

Stream cipher and Block Cipher

Chapter-3: Block Ciphers and the Data Encryption Standard

Mohammad Asif
Assistant Professor
Department of Computer Science and Engineering

Content

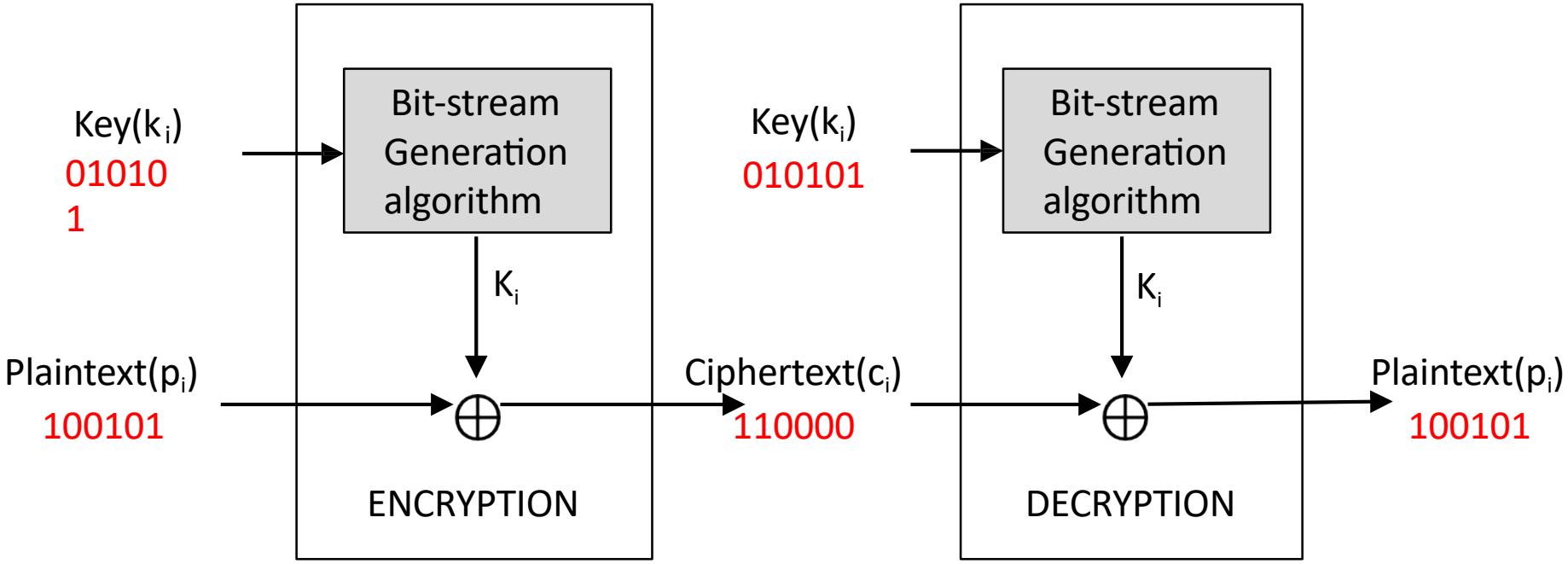
Stream cipher and Block Cipher:

A stream cipher is one that encrypts a digital data stream one bit or one byte at a time.

Examples:

Autokeyed Vigenère cipher
A5/1
RC4
Vern
am
ciph
er.

Stream cipher and Block Cipher:



Stream cipher and Block Cipher:

A block cipher is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used.

Examples:

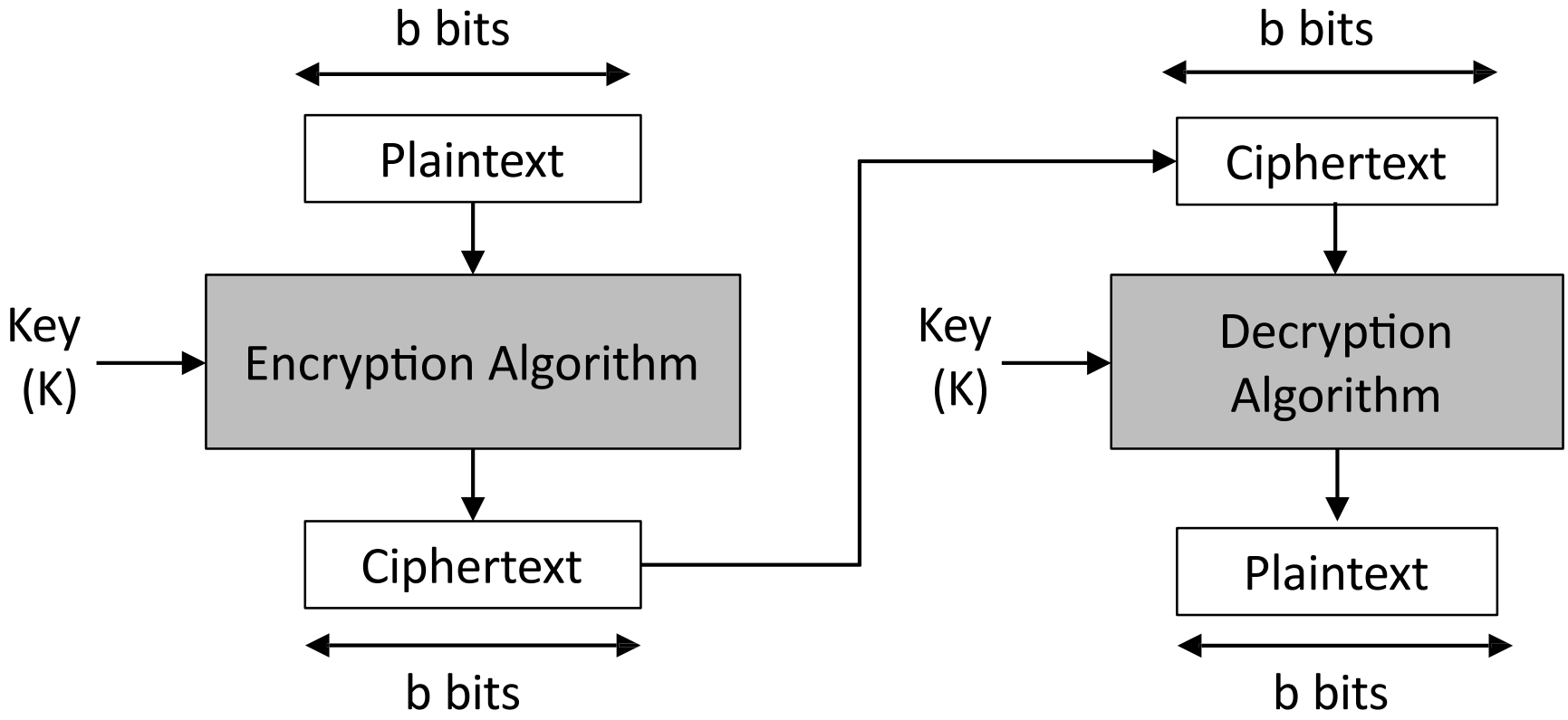
Feistel cipher

DES

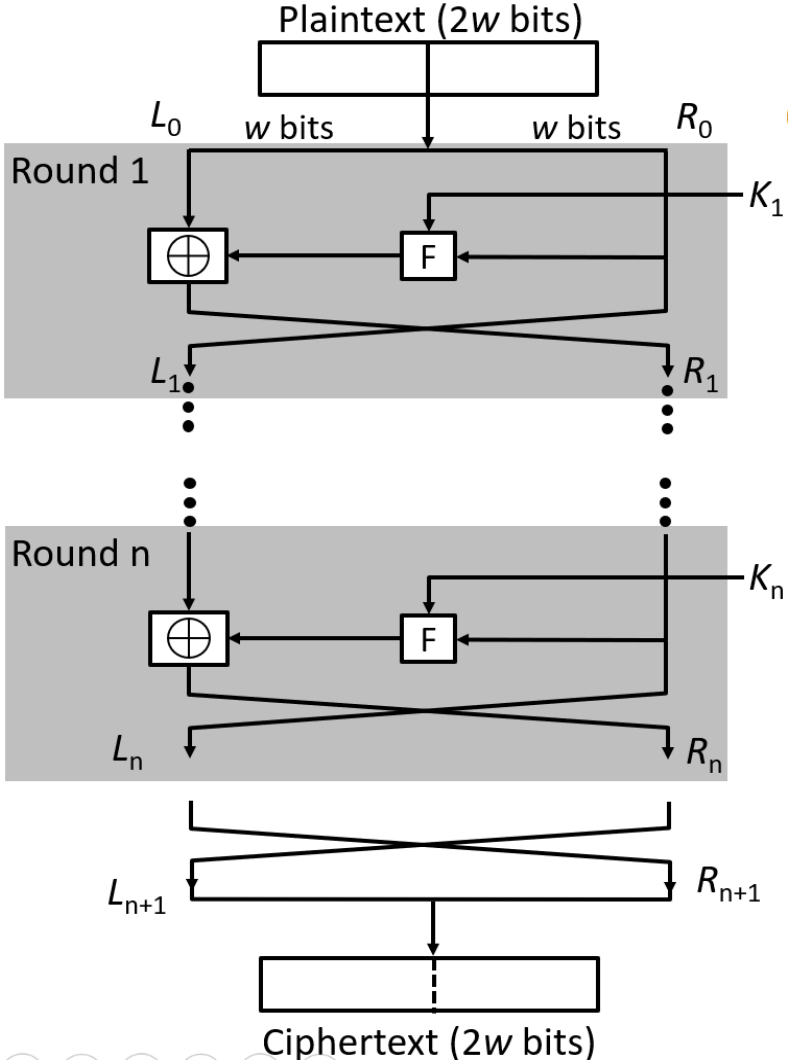
Triple DES

AES

Stream cipher and Block Cipher:



Block Cipher Principle – Fiestel Structure



1. Plaintext is split into 32-bit halves L_i and R_i
2. R_i is fed into the function F.
3. The output of function F is then XORed with L_i
4. Left and right half are swapped.

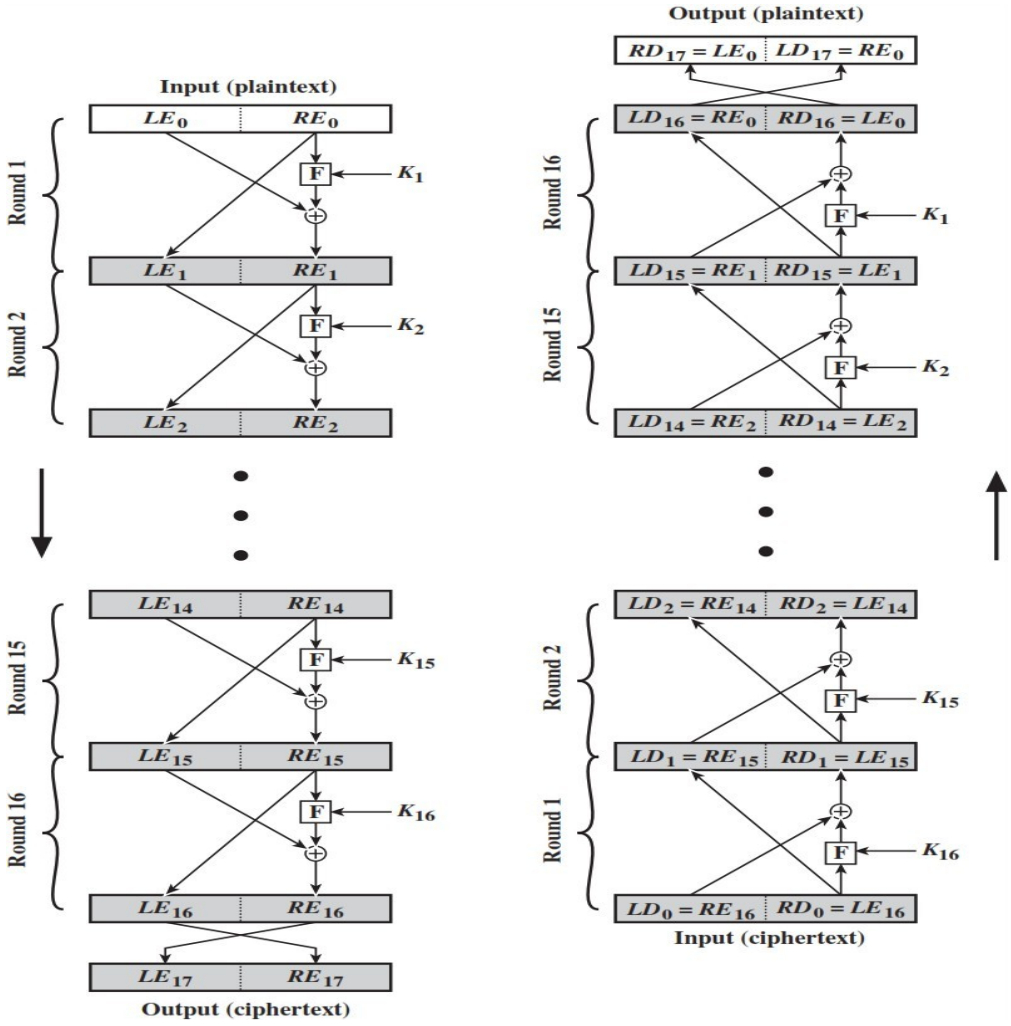
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

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

Block Cipher Principle – Fiestel Structure

1. **Block size:** Common block size of 64-bit. However, the new algorithms uses a 128-bit, 256-bit block size.
2. **Key size:** Key sizes of 64 bits or less are now widely considered to be insufficient, and 128 bits has become a common size.
3. **Number of rounds**: A typical size is 16 rounds.
4. **Round function F:** This phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. Again, greater complexity generally means greater resistance to cryptanalysis.
5. **Subkey generation algorithm:** For each of the sixteen rounds, a different subkey (Ki) derived from main key by the combination of a left circular shift and a permutation. Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

Block Cipher Principle – Fiestel Structure



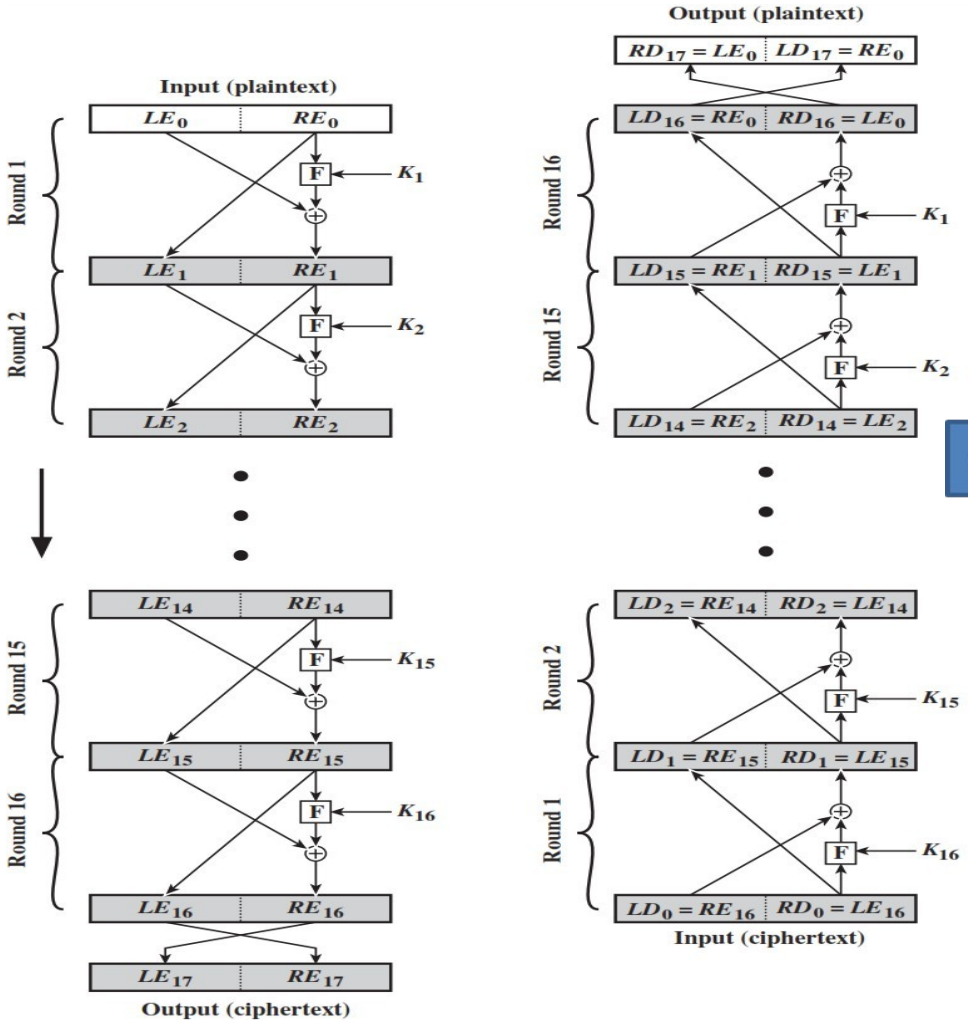
Prove that o/p of first round of Decryption is equal to 32-bit swap o i/p of 16th round of Encryption
LD1=RE15 & RD1=LE15

On Encryption Side:

$$LE_{16} = RE_{15}$$

$$RE_{16} = LE_{15} \oplus F(RE_{15}, K_{16})$$

Block Cipher Principle – Fiestel Structure



On Decryption Side:

$$LD_1 = RD_0 = LE_{16} = RE_{15}$$

$$RD_1 = LD_0 \oplus F(RD_0, K_{16})$$

$$= RE_{16} \oplus F(RE_{15}, K_{16})$$

$$= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16})$$

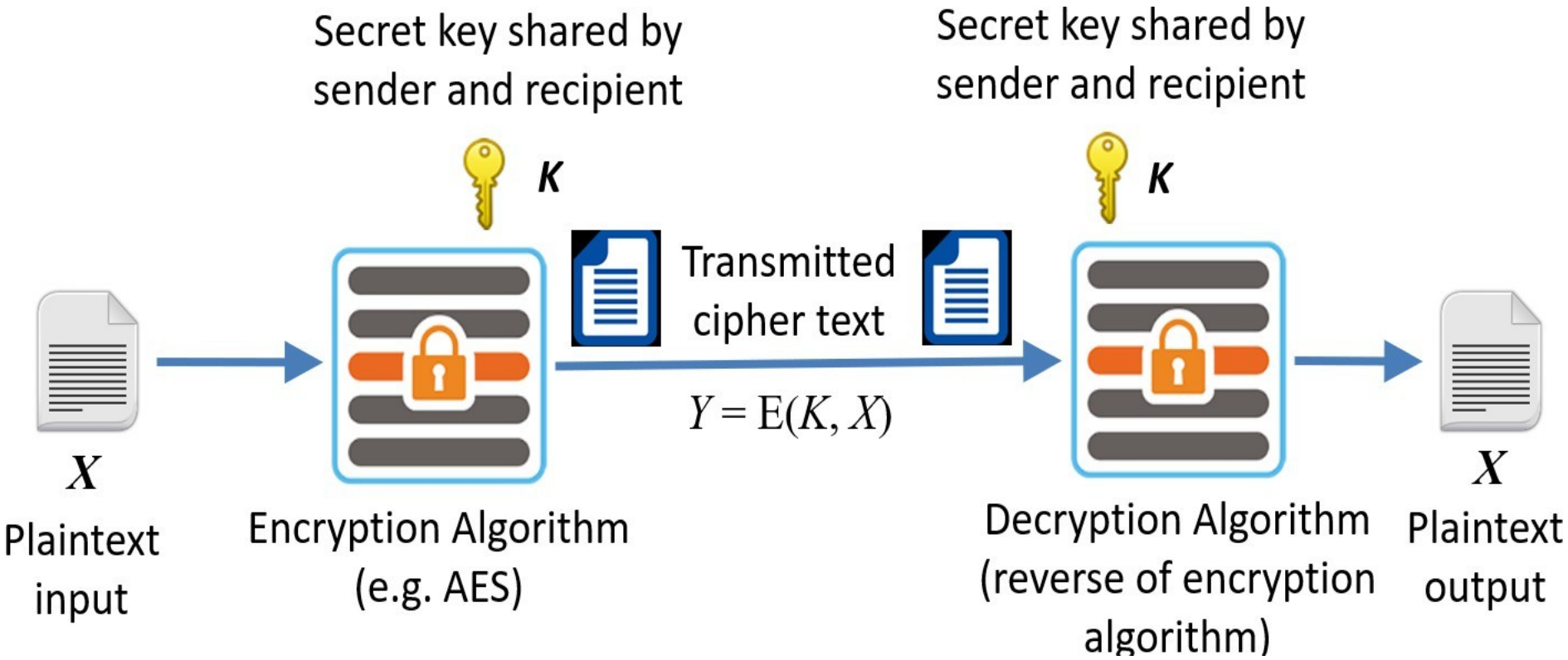
Thus,

$$LD_1 = RE_{15} \;\&\; RD_1 = LE_{15}$$

XOR Associativity Property

$$\because [A \oplus B] \oplus C = A \oplus [B \oplus C]$$

Symmetric Cipher Model



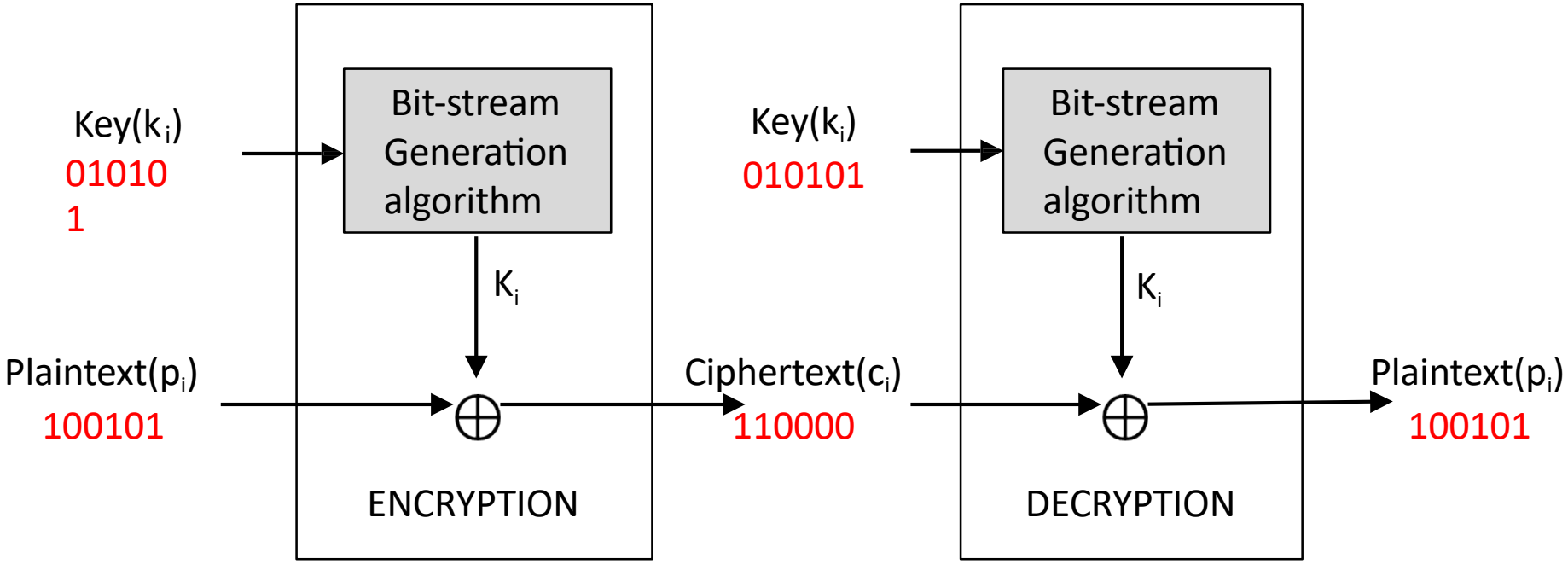
Stream cipher and Block Cipher:

A stream cipher is one that encrypts a digital data stream one bit or one byte at a time.

Examples:

Autokeyed Vigenère cipher
A5/1
RC4
Vern
am
ciph
er.

Stream cipher and Block Cipher:



Stream cipher and Block Cipher:

A block cipher is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used.

Examples:

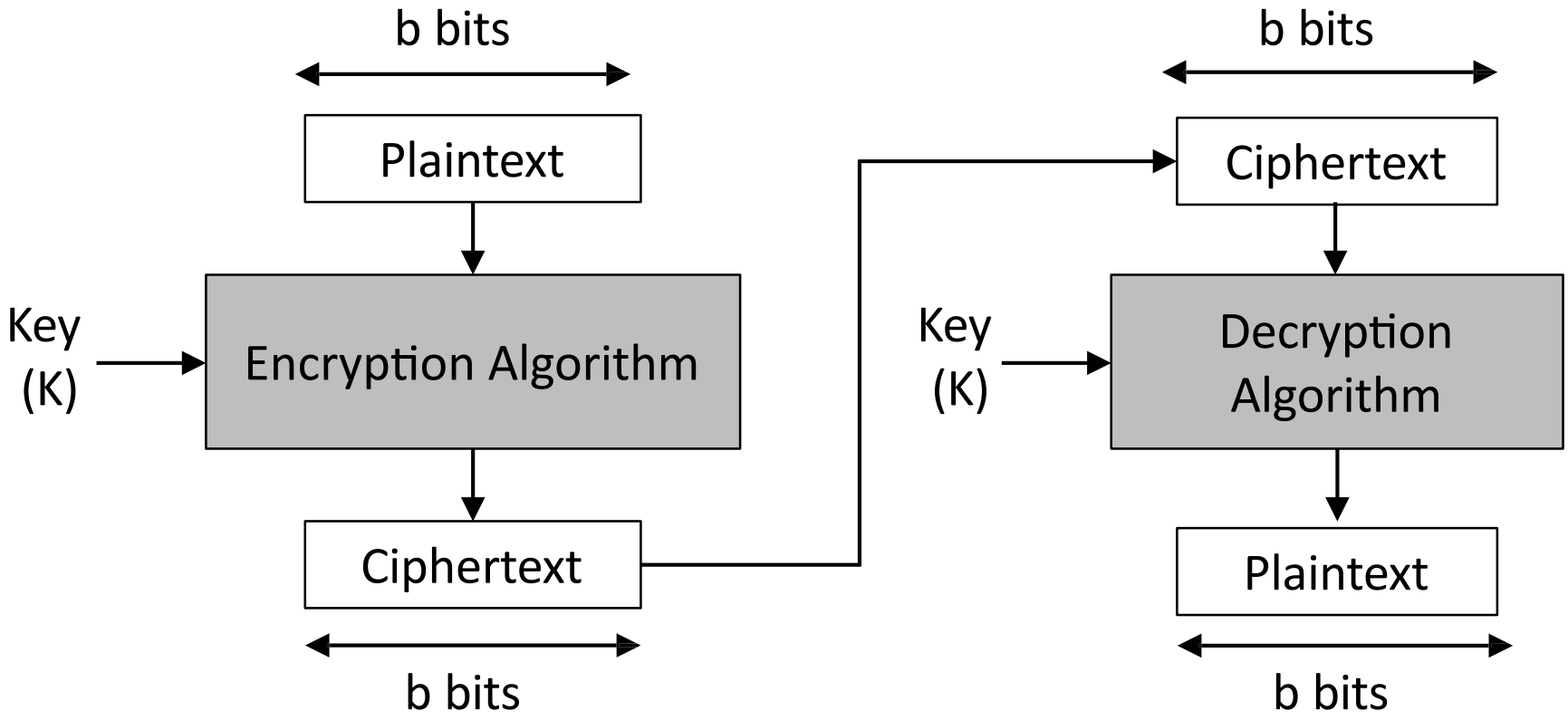
Feistel cipher

DES

Triple DES

AES

Stream cipher and Block Cipher:



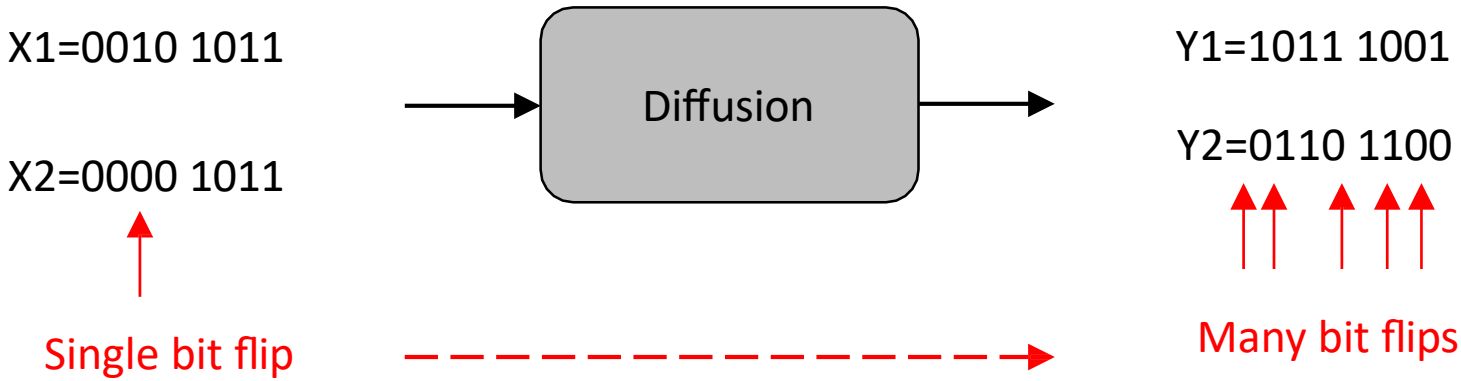
Confusion & Diffusion:

Confusion

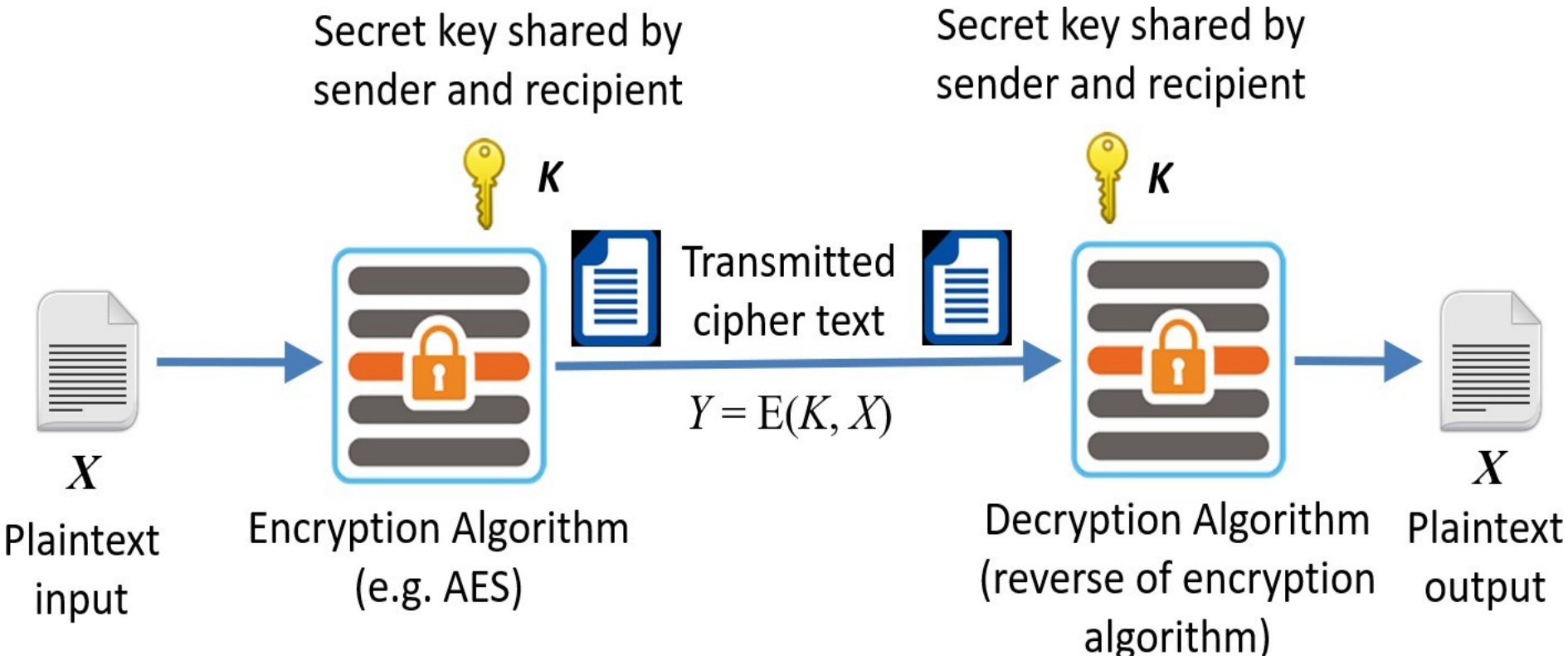
- Confusion hides the relationship between the cipher text and the key.
- This is achieved by the use of a complex substitution algorithm.

Diffusion

- Diffusion hides the relationship between the cipher text and the plaintext.
- This is achieved by changing one plaintext digit which affect the value of many cipher text digits.

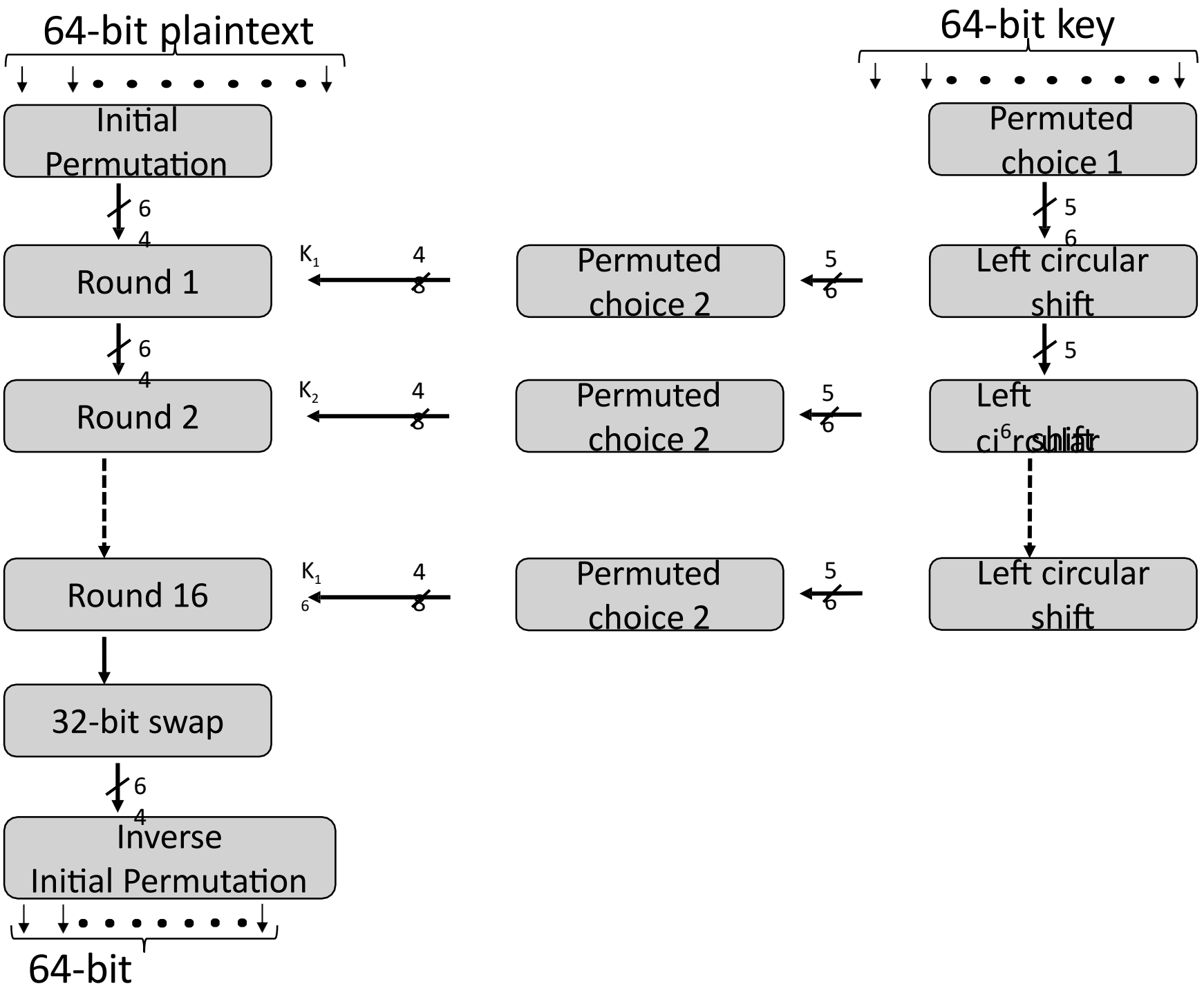


Symmetric Cipher Model

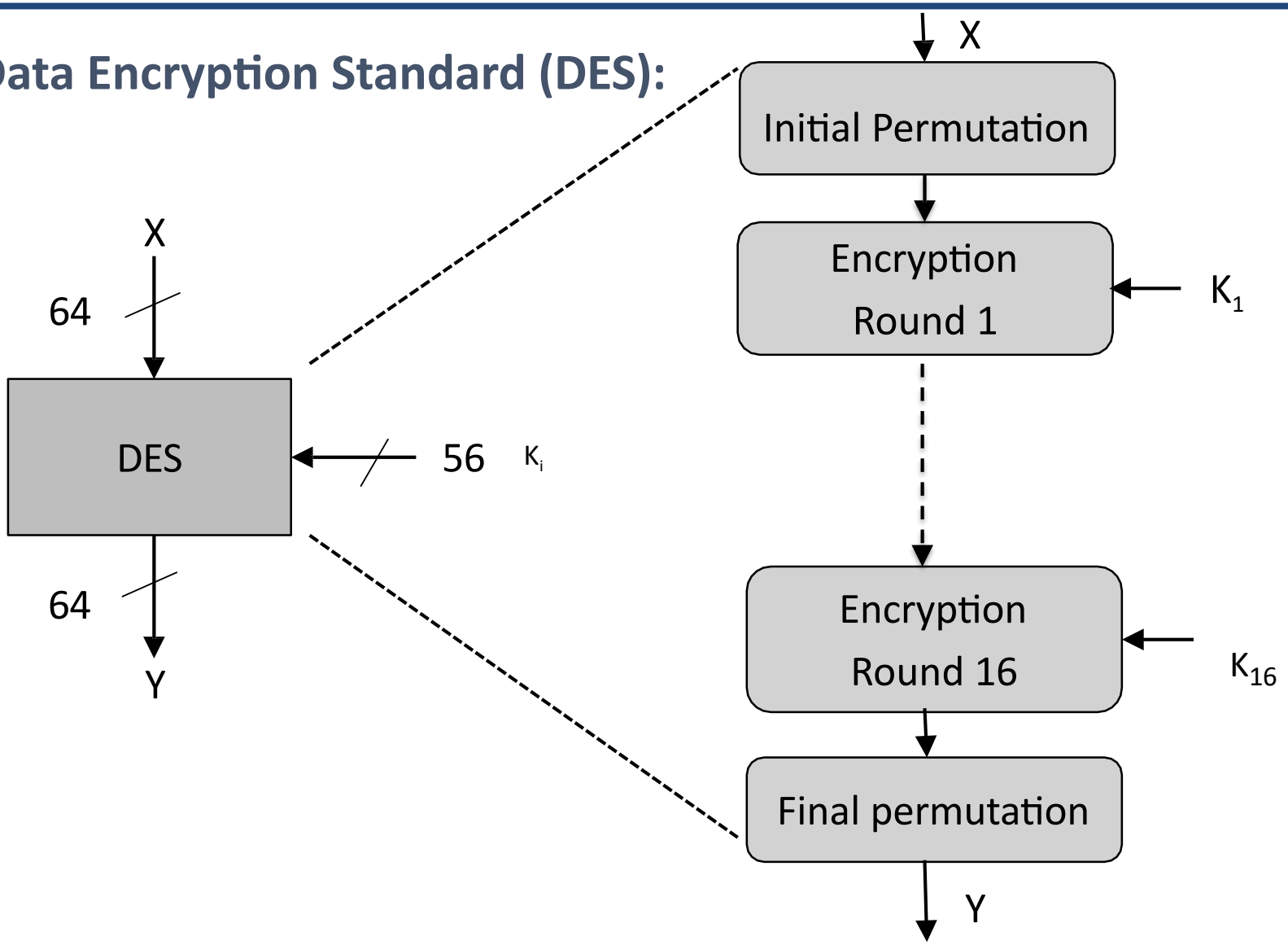


Data Encryption Standard (DES):

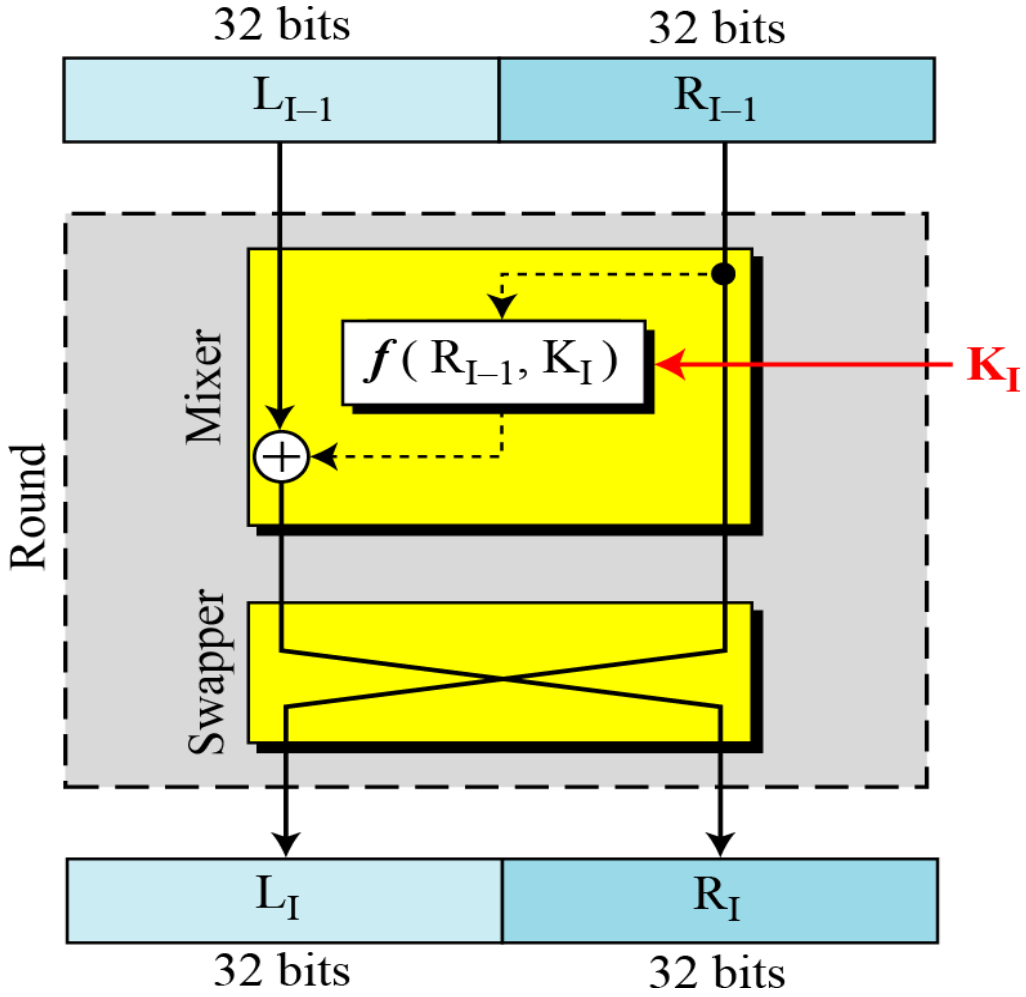
Type: Block Cipher
Block Size : 64-bit
Key Size: 64-bit,
with only 56-bit
effective
Number of
Rounds: 16

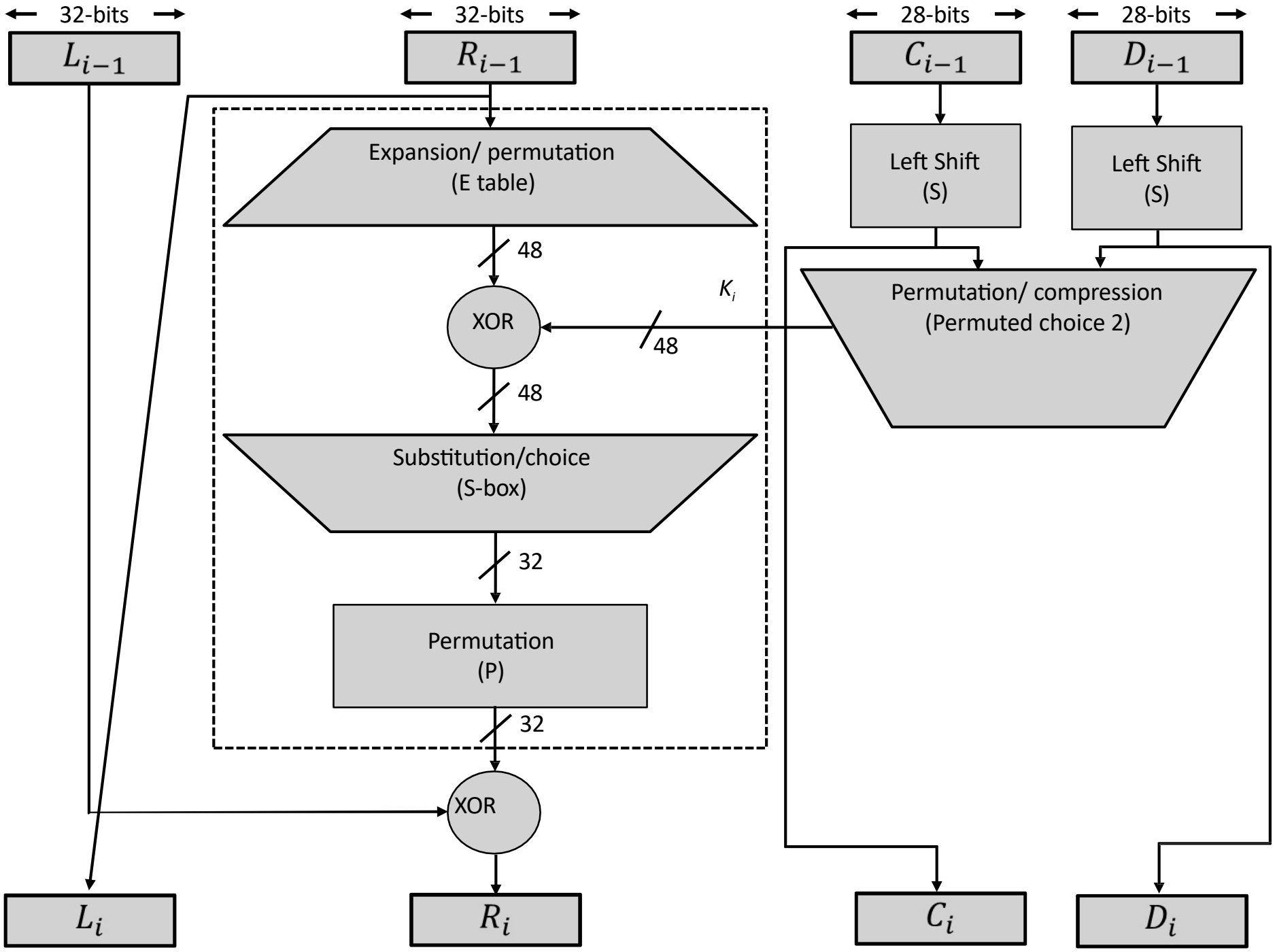

64-bit plaintext
64-bit key
Initial Permutation
Permuted choice 1
64
56
Round 1
Permuted choice 2
Left circular shift
48
56
K1
Round 2
Permuted choice 2
Left circular shift
48
56
K2
Round 16
Permuted choice 2
Left circular shift
48
56
K16
32-bit swap
64
Inverse Initial Permutation
64-bit

Data Encryption Standard (DES):



Data Encryption Standard (DES) – Single round of DES:



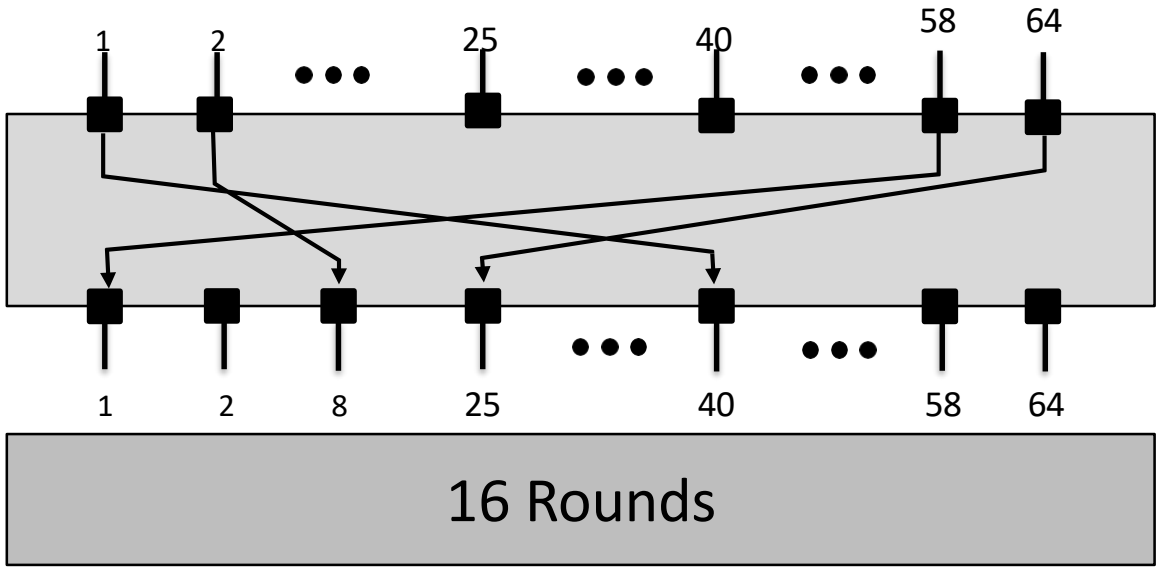

32-bits
32-bits
28-bits
28-bits
L_{i-1}
R_{i-1}
C_{i-1}
D_{i-1}
Expansion/ permutation
(E table)
Left Shift
(S)
Left Shift
(S)
48
K_i
XOR
48
Permutation/ compression
(Permuted choice 2)
48
Substitution/choice
(S-box)
32
Permutation
(P)
32
XOR
L_i
R_i
C_i
D_i

Data Encryption Standard (DES):

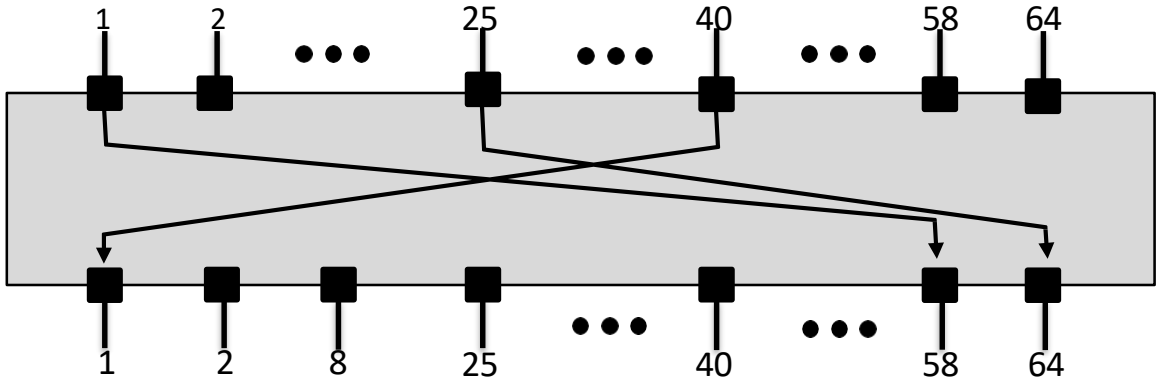
1. Initial permutation: First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the permuted input.
2. The F function: This phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions.
3. Swap: L and R swapped again at the end of the cipher, i.e., after round 16 followed by a final permutation.
4. Inverse (Final) permutation: It is the inverse of the initial permutation.
5. Subkey generation: For each of the sixteen rounds, a different subkey (Ki) derived from main key by the combination of a left circular shift and a permutation.

Data Encryption Standard (DES): - Initial Permutation

The initial permutation of the DES algorithm changes the order of the plaintext prior to the first round of encryption



The final permutation occurs after the sixteen rounds of DES are completed. It is the inverse of the initial permutation.



Data Encryption Standard (DES): Initial and Final Permutation

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

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

Data Encryption Standard (DES): The f Function

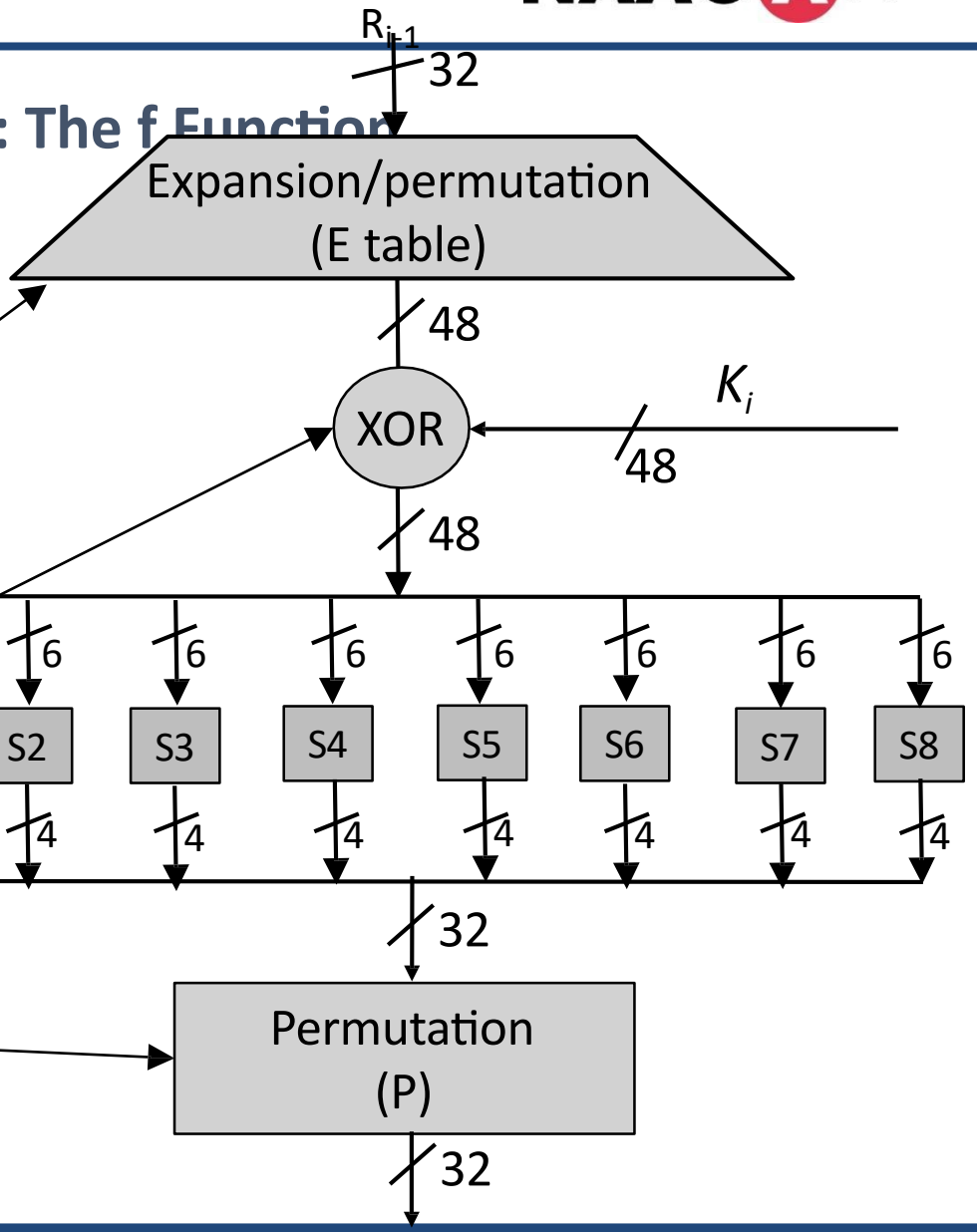
1. Main operation of DES
- f-function inputs:
  Ri-1 and round key *ki*
- **4 Steps**:

1. Expansion E

2. XOR with round key

3. S-box substitution

4. Permutation



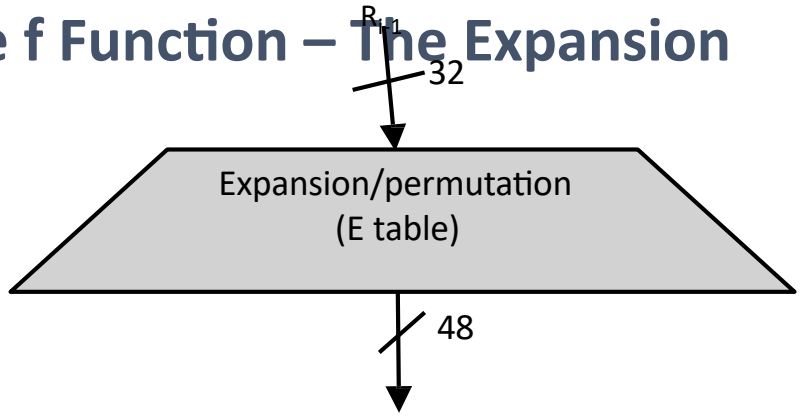
Data Encryption Standard (DES): The f Function – The Expansion Function

Main purpose: **Increases diffusion**

Since Ri-1 is a 32-bit input and Ki is a 48-bit key, we first need to expand Ri-1 to 48 bits.

Input: (8 blocks, each of them consisting 4 bits) - 32 bits

Output: (8 blocks, each of them consisting 6 bits) – 48 bits



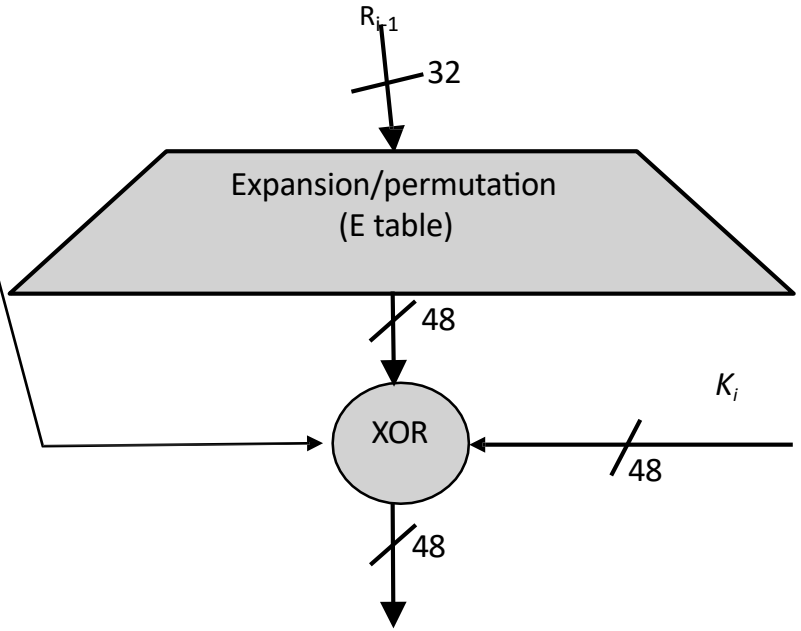
Expansion Table E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Data Encryption Standard (DES): XOR round Key

XOR Round Key

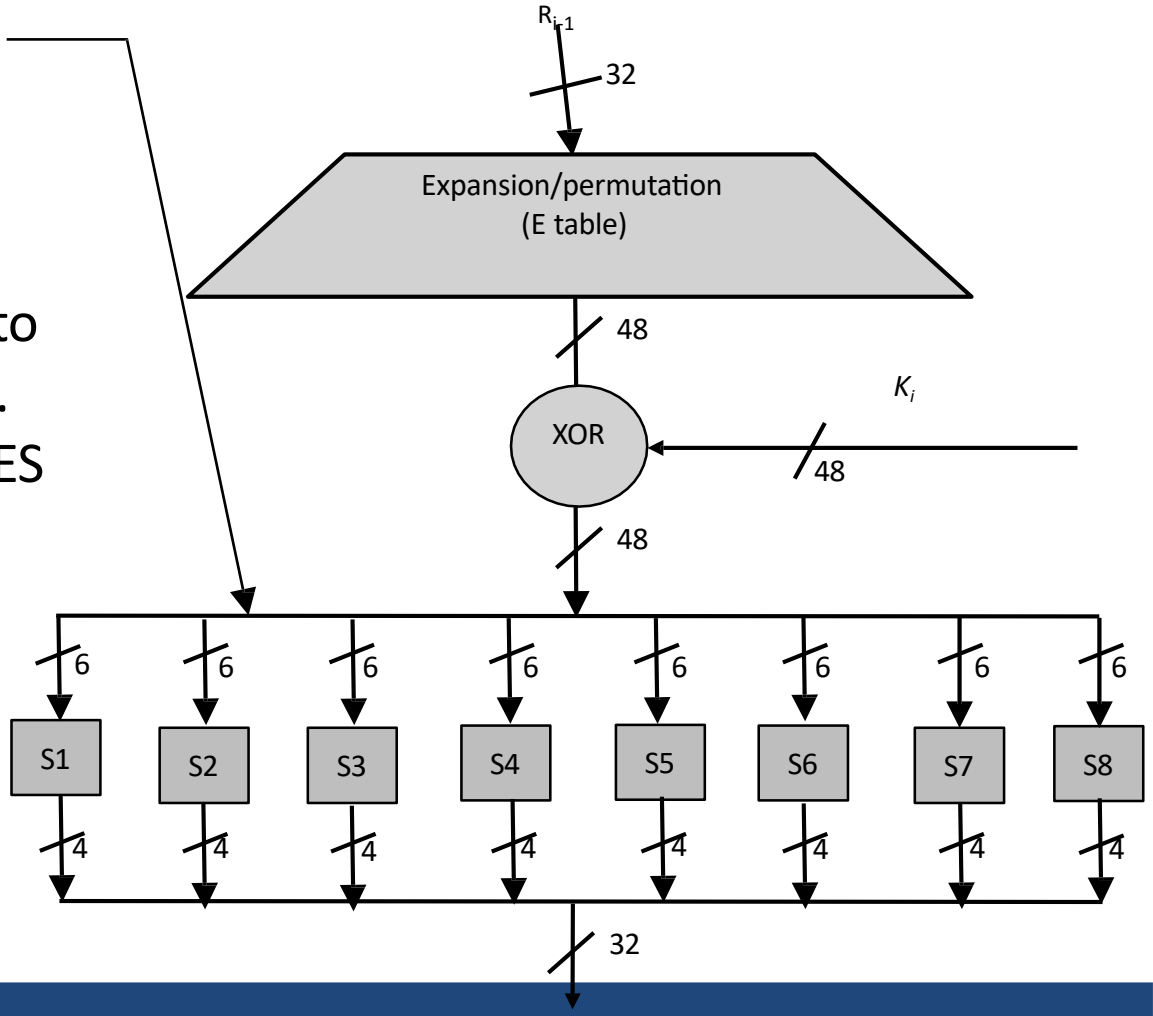
After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key.

Note that both the right section and the key are 48-bits in length now.



Data Encryption Standard (DES): S-Box substitution

- Eight substitution tables.
- 6 bits of input
- 4 bits of output.
- Convert 48 bits to 32 bits
- Non-linear and resistant to differential cryptanalysis.
- Crucial element for DES security!
- Introduces confusion.



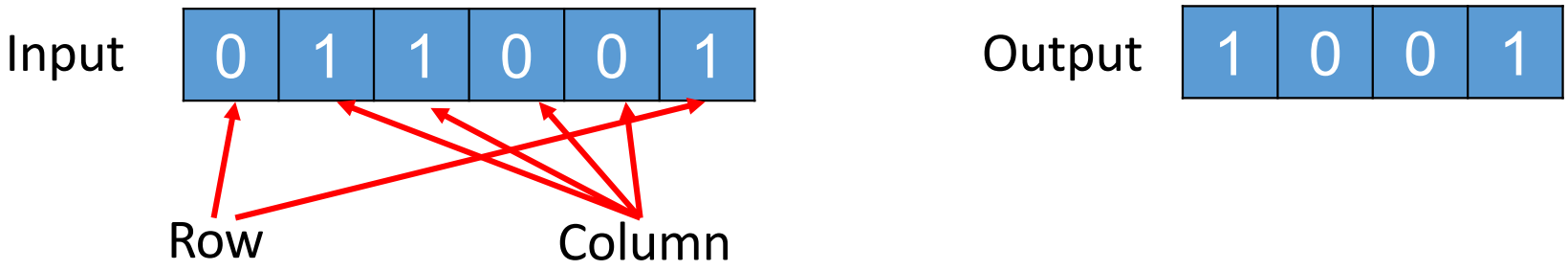
Data Encryption Standard (DES): S-Box substitution

The outer two bits of each group select one row of an S-box.
Inner four bits selects one column of an S-box.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

S-box 1

- Example:

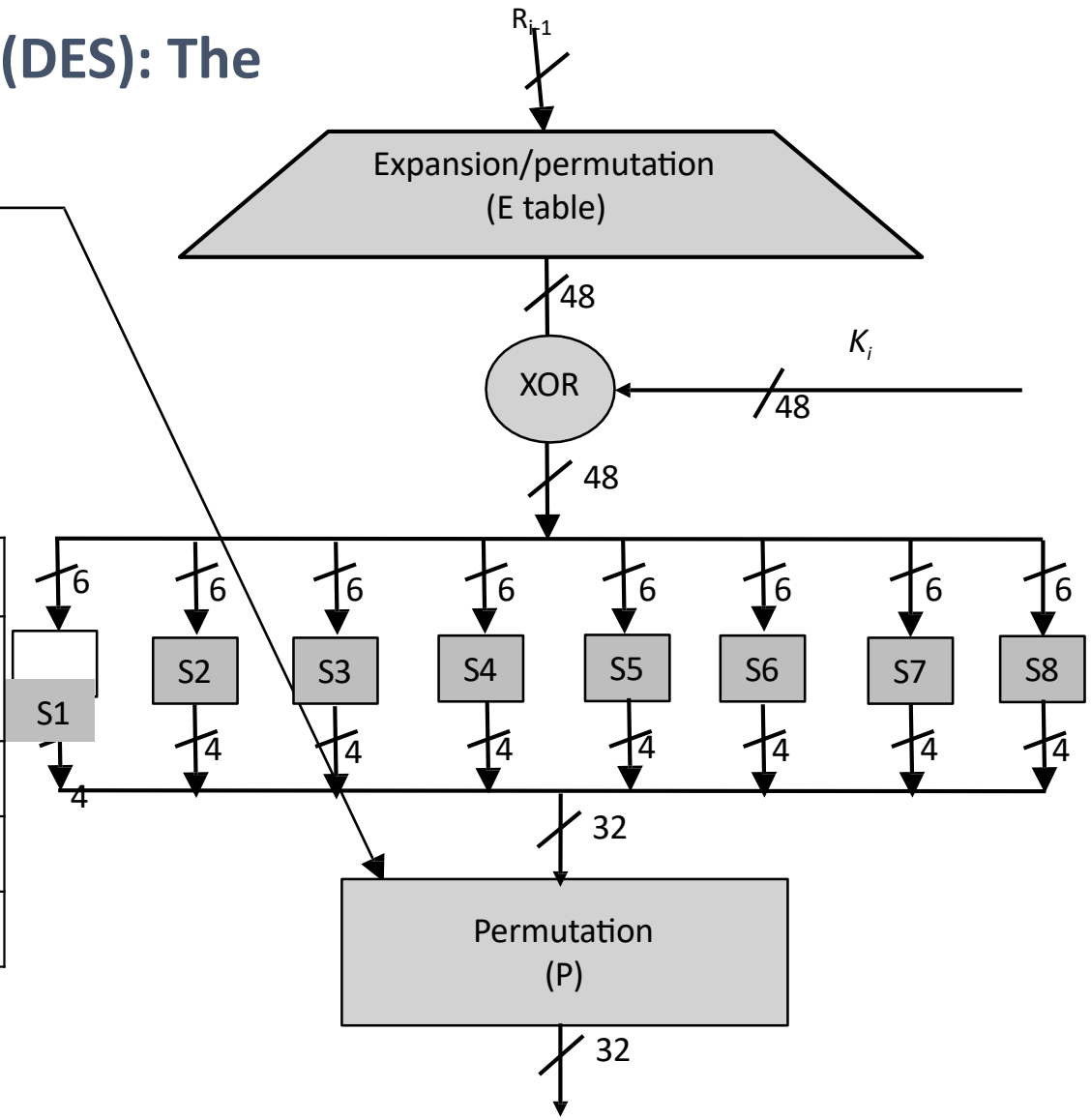


Data Encryption Standard (DES): The Permutation

Permutation P

- Bitwise permutation.
- **Introduces diffusion.**

Permutation Table P							
16	7	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25



Avalanche Effect

Desirable property of any encryption algorithm is that a change in one bit of the plaintext or of the key should produce a change in many bits of cipher text.
DES performs strong avalanche effect.

Plaintext: 0000000000000000	Key: 22234512987ABB23
Ciphertext: 4789FD476E82A5F1	
Plaintext: 0000000000000001	Key: 22234512987ABB23
Ciphertext: 0A4ED5C15A63FEA3	

Although the two plaintext blocks differ only in the rightmost bit, the cipher text blocks differ in 29 bits.
This means that changing approximately 1.5 % of the plaintext creates a change of approximately 45 % in the ciphertext.

Strength of DES

The use of 56-bit keys: 56-bit key is used in encryption, there are 256 possible keys. A brute force attack on such number of keys is impractical.

The nature of algorithm: Cryptanalyst can perform cryptanalysis by exploiting the characteristic of DES algorithm but no one has succeeded in finding out the weakness.

Design Principle of Block Cipher :

1. ConfusionPurpose: Make the relationship between the ciphertex and the encryption key as complex as possible.Achieved by: Using substitution operations (like S-boxes).
   Effect: Even a small change in the key or plaintext causes major,
   unpredictable changes in ciphertext.

2. DiffusionPurpose: Spread the influence of a single plaintext bit across many ciphertext bits. Achieved by: Using permutation and mixing operations.
   Effect: Changing one bit of the plaintext affects many bits of the ciphertext.

Design Principle of Block Cipher :

3.Kerckhoffs's Principle : A cipher should remain secure even if everything about the system (except the key) is public knowledge. Focuses security entirely on the secrecy of the key, not the algorithm.

4.Iterative Structure (Rounds)Instead of a single operation, block ciphers apply multiple rounds of transformations. Each round improves confusion and diffusion.
Example: AES uses 10, 12, or 14 rounds depending on key size.

5.Key Expansion The key schedule algorithm generates a different subkey for each round from the original key. Strong key expansion ensures better security.

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/

Program No: 1

Problem statement:

A program that creates a simple web server and serves a static HTML page.

Aim:

To solve this problem statement.

Problem Description:

Develop a Python program using Flask to create a simple web server serving a static HTML page. The objective is to use the flask command (flask --app <YourAppName> run) instead of running the script directly. Ensure proper project organization, write a basic HTML file, and confirm functionality by accessing http://127.0.0.1:5000/ in a web browser.

Algorithm:

Step 1: start
Step 2: Install Flask
Step 3: create a Project Structure
Step 4: Write a HTML file and save as a index.html file
Step 5: Write a Flask file and save as a app.py file
Step 6: Run that Flask file.
Step 7: Stop

HTML FILE:

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Static HTML Page</title>
</head>
<body>
    <h1>Hello, this is a static HTML page served by Flask!</h1>
```

```
</body>
</html>
```

FLASK FILE:

app.py

```
from flask import Flask,
render_template app = Flask(____name
_____________)
# Define a route for the root URL "/"
@app.route("/")
def home():
    # Render the static HTML page located in the "templates" folder
    return render_template("index.html")

# Run the app if this script is the main
program if__name_== "__main__":
    app.run(debug=True)
```

Output:

Conclusion:

Here We have successfully creates a simple web server and serves a static HTML page.

Program No: 2

Problem statement:

A program that creates a web application that allows users to register and login.

Aim:

To solve this problem statement.

Problem Description:

The web application will be built using a combination of front-end and back-end technologies. The front-end will be responsible for creating the user interface, while the back-end will handle user registration, login authentication, and data storage.

Algorithm:

Step 1: start
Step 2: create a Project Structure
Step 3: Write a HTML file and save as a index.html ,login.html & register.html file
Step 4: Write a Flask file and save as a app.py file
Step 5: Run that Flask file.
Step 6: Stop

HTML FILE:

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Static HTML Page</title>
  </head>
  <style>
```

```
@import url("https://fonts.googleapis.com/css2?
family=Poppins:wght@500&display=sw ap");
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
body {
  height: 100vh;
  width: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction:
  column; background:
  #ff5a5f;
}
h1 {
  font-family: "Poppins", sans-serif;
  color: #fff;
  margin: 30px
  50px; font-size:
  3rem;
}
input {
  padding: 10px 20px;
  border: 3px solid
  #fff; border-radius:
  10px;
  background: rgb(16, 208,
  16); font-size: 1.5rem;
  color: white;
  font-family: "Poppins", sans-
  serif; font-weight: 300;
  transition: .3s;
  &:hover{ backgr
  ound: #fff; color:
  #000; cursor:
  pointer;
   }
```

```
}
```

```
  </style>
  <body>
    <h1>Hello, this is a static HTML page served by Flask!</h1>
    <form action="{{ url_for('register') }}">
      <input type="submit" value="Register" />
    </form>
  </body>
</html>
```

login.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>User Login</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        height: 100vh;
        width: 100%;
        display: flex;
        align-items: center;
        justify-content: center;
        flex-direction:
        column;
        background: rgb(9, 9, 121);
        background: linear-gradient(
        30deg,
          rgba(9, 9, 121, 1) 0%,
          rgba(2, 0, 36, 1) 29%,
          rgba(0, 212, 255, 1) 100%
        );
      }
```

```
.container {
  display: flex;
  align-items: center;
  justify-content: space-
  evenly; flex-direction:
  column; width: 600px;
  border-radius:  20px;
  height:  500px;
  background: #ffffff5a;
  backdrop-filter:
  blur(20px); & h1 {
    font-family: Arial, Helvetica, sans-serif;
    color: #fff;
    margin: 30px 0;
  }
  & li {
    list-style: none;
  }
  & form {
    & label {
      color: white;
      font-family: Arial, Helvetica, sans-
      serif; font-size: 1.4rem;
      margin: 10px 20px;
    }
    & .log_button {
      color: #fff;
      background: red;
      border: none;
      outline: none;
      padding: 5px 10px;
      border-radius:
      10px; font-size:
      1.2rem; transition:
      0.3s;
      transform: translateX(130px);
      &:hover {
        background: #fff;
        color: #000;
        cursor: pointer;
```

```
        }
      }
      &  .password{ paddin
        g: 10px 20px;
        border-radius:
        20px; outline: none;
        border: none;
      }
      &  .username{ paddin
        g: 10px 20px;
        border-radius:
        20px; outline: none;
        border: none;
      }
      & input {
        margin: 10px 20px;
      }
      }
    }
    .error {
      color: red;
    }
    .success {
      color: green;
    }
    .default {
      color: black;
    }
  </style>
 </head>
 <body>
  <div class="container">
   <h1>User Login</h1>
   {% with messages = get_flashed_messages() %} {% if messages %}
   <ul>
    {% for message in messages %}
    <li
     class="{% if 'error' in message %}error{% elif 'success' in message
%}success{% else %}default{% endif %}"
```

```
      >
        {{ message }}
      </li>
      {% endfor %}
    </ul>
    {% endif %} {% endwith %}
    <form method="post" action="{{ url_for('login') }}">
      <label for="username" class="username_label">Username:</label>
      <input type="text" name="username" class="username" required />
      <br />
      <label for="password" class="password_label">Password:</label>
      <input type="password" name="password" class="password" required />
      <br />
      <input type="submit" class="log_button" value="Log in" />
    </form>
    <p>
      Don't have an account?
      <a href="{{ url_for('register') }}">Register here</a>.
    </p>
  </div>
 </body>
</html>
```

register.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>User Registration</title>
  <style>
   * {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
   }
   body {
```

```
  height: 100vh;
  width: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction:
  column;
  background: rgb(9, 9, 121);
  background: linear-gradient(
  30deg,
    rgba(9, 9, 121, 1) 0%,
    rgba(2, 0, 36, 1) 29%,
    rgba(0, 212, 255, 1) 100%
  );
 }
 .container {
  display: flex;
  align-items: center;
  justify-content: space-
  evenly; flex-direction:
  column; width: 600px;
  border-radius:  20px;
  height:  500px;
  background: #ffffff5a;
  backdrop-filter:
  blur(20px); & h1 {
    font-family: Arial, Helvetica, sans-serif;
    color: #fff;
    margin: 30px 0;
  }
  & li {
    list-style: none;
  }
  & form {
    & label {
      color: white;
      font-family: Arial, Helvetica, sans-
      serif; font-size: 1.4rem;
      margin: 10px 20px;
    }
```

```
    & .register_button {
      color: #fff;
      background: red;
      border: none;
      outline: none;
      padding: 5px 10px;
      border-radius:
      10px; font-size:
      1.2rem; transition:
      0.3s;
      transform: translateX(130px);
      &:hover {
        background: #fff;
        color: #000;
        cursor: pointer;
      }
    }
    & .password {
      padding: 10px
      20px; border-
      radius: 20px;
      outline: none;
      border: none;
    }
    & .username {
      padding: 10px
      20px; border-
      radius: 20px;
      outline: none;
      border: none;
    }
    & input {
      margin: 10px 20px;
    }
  }
}
.error {
  color: red;
}
.success {
```

```
  color: green;
}
```

```
      .default {
        color: black;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h1>User Registration</h1>
      {% with messages = get_flashed_messages() %} {% if messages %}
      <ul>
        {% for message in messages %}
        <li
          class="{% if 'error' in message %}error{% elif 'success' in message
%}success{% else %}default{% endif %}"
        >
          {{ message }}
        </li>
        {% endfor %}
      </ul>
      {% endif %} {% endwith %}
      <form method="post" action="{{ url_for('register') }}">
        <label for="username" class="username_label">Username:</label>
        <input type="text" name="username" class="username" required />
        <br />
        <label for="password" class="password_label">Password:</label>
        <input type="password" name="password" class="password" required />
        <br />
        <input type="submit" class="register_button" value="Register" />
      </form>
      <p>
        Already have an account?
        <a href="{{ url_for('login') }}">Log in here</a>.
      </p>
    </div>
  </body>
</html>
```

FLASK FILE:

app.py

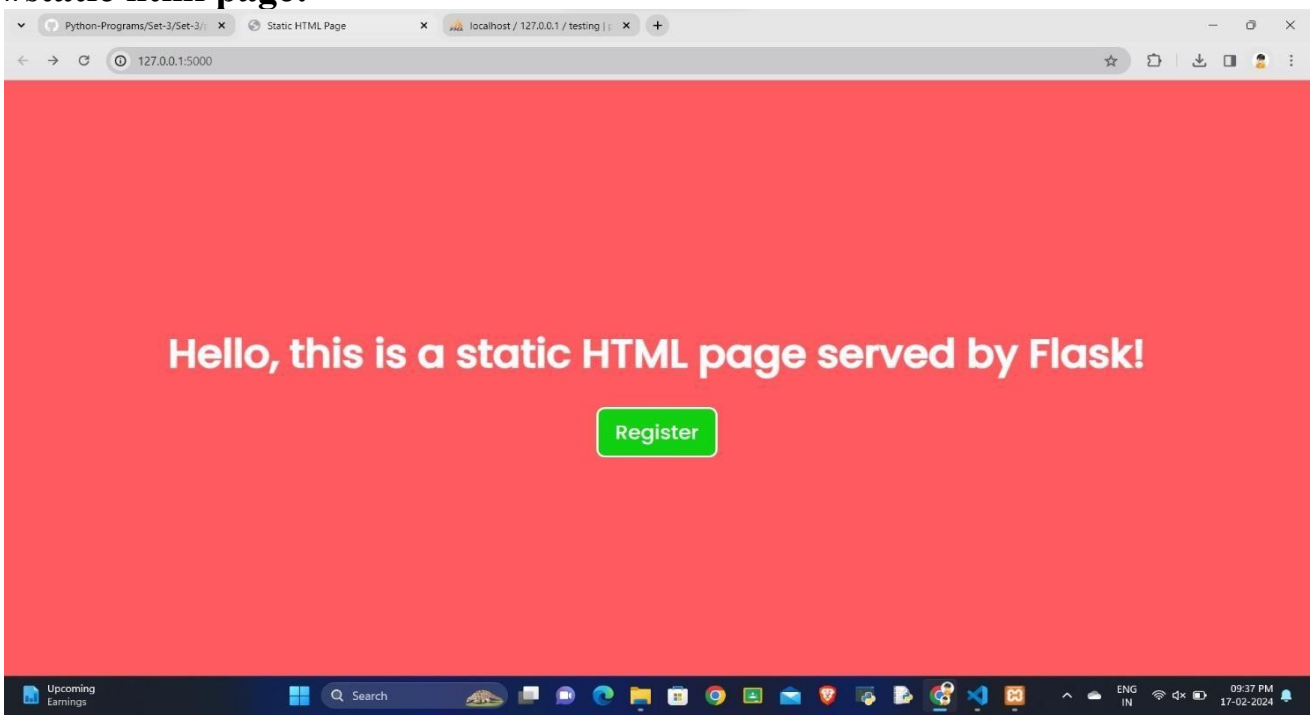
```
from flask import Flask, render_template, request, redirect, url_for, session,
flash
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
import secrets

# print(secrets.token_hex(16))

app = Flask(__name__)
app.secret_key = secrets.token_hex(16)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db' # SQLite
database, change for other databases
db = SQLAlchemy(app)

# Define the User model
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(256), nullable=False)

# Ensure the creation of all tables inside the application context
with app.app_context():
    db.create_all()

@app.route("/")
def home():
    # Render the static HTML page located in the "templates" folder
    return render_template("index.html")

# Define a route for the registration page
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method ==  'POST':
        username =
        request.form['username']
```

```
        password = request.form['password']

        # Check if the username is already taken
        if User.query.filter_by(username=username).first():
            flash('Username already taken. Please choose another.', 'error')
        else:
            # Hash the password before storing it
            hashed_password = generate_password_hash(password,
method='pbkdf2:sha256')

            # Create a new user instance
            new_user = User(username=username, password=hashed_password)

            # Add the user to the database
            db.session.add(new_user)
            db.session.commit()

            flash('Registration successful. You can now log in.', 'success')
            return redirect(url_for('login'))

    return render_template('register.html')

# Define a route for the login page
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method  ==  'POST':
        username = request.form['username']
        password = request.form['password']

        # Query the user from the database
        user = User.query.filter_by(username=username).first()

        # Check if the username exists and the password is correct
        if user and check_password_hash(user.password, password):
            session['username'] = username
            flash('Login successful!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid username or password. Please try again.', 'error')
```

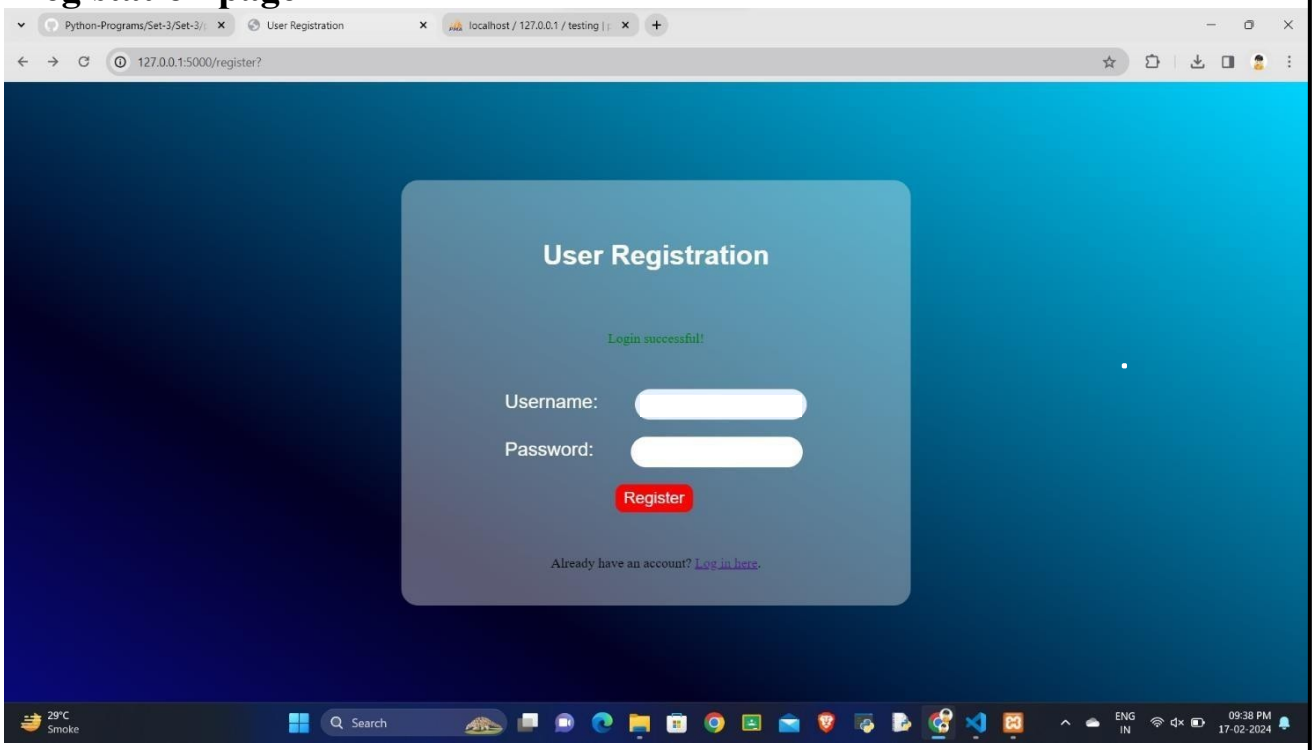
```
    return render_template('login.html')

# Define a route for the dashboard (a protected route accessible only after login)
@app.route('/dashboard')
def dashboard():
    if 'username' in session:
        return f'Welcome to the dashboard, {session["username"]}!'
    else:
        flash('Please log in to access the dashboard.', 'info')
        return redirect(url_for('login'))

# Logout route
@app.route('/logout')
def logout():
    session.pop('username', None)
    flash('You have been logged out.', 'info')
    return redirect(url_for('login'))

if__name__== '__main
    __': app.run(debug=True)
```

Output:

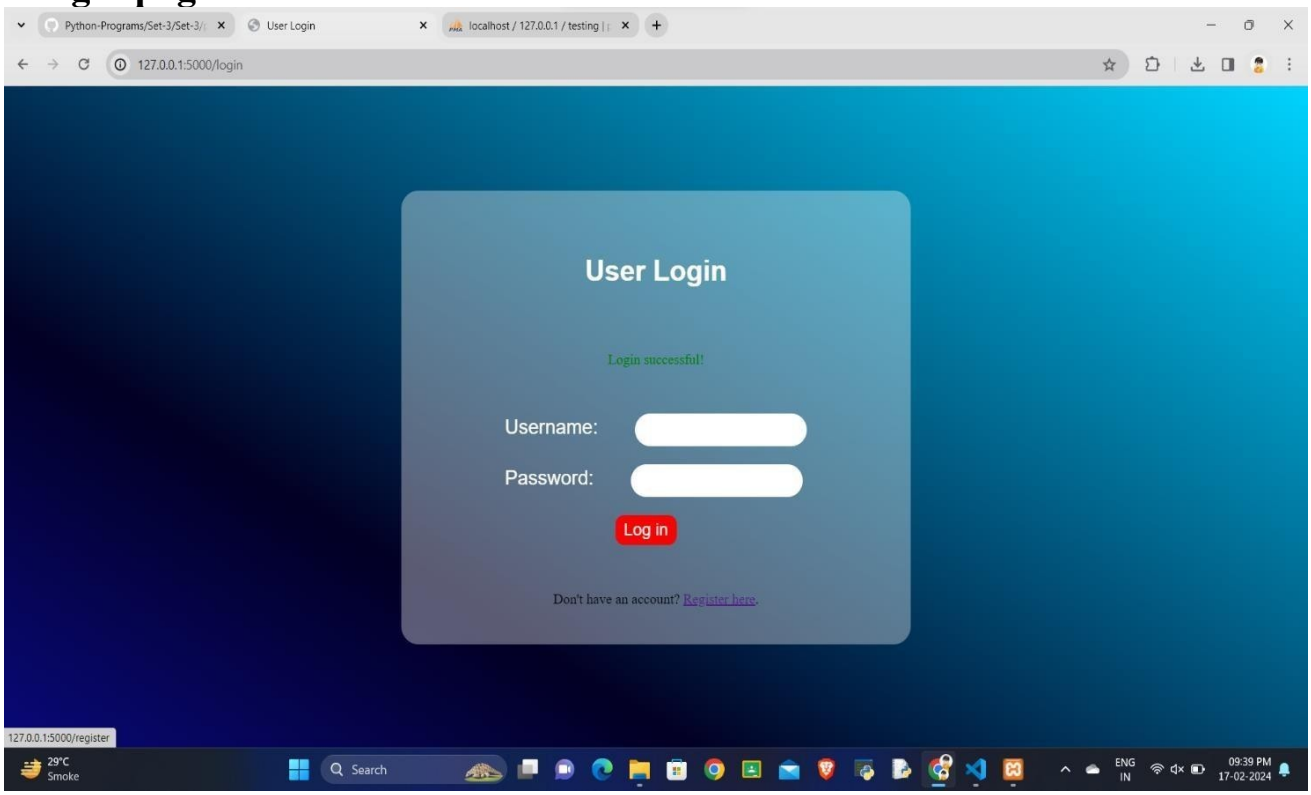
#static html page:



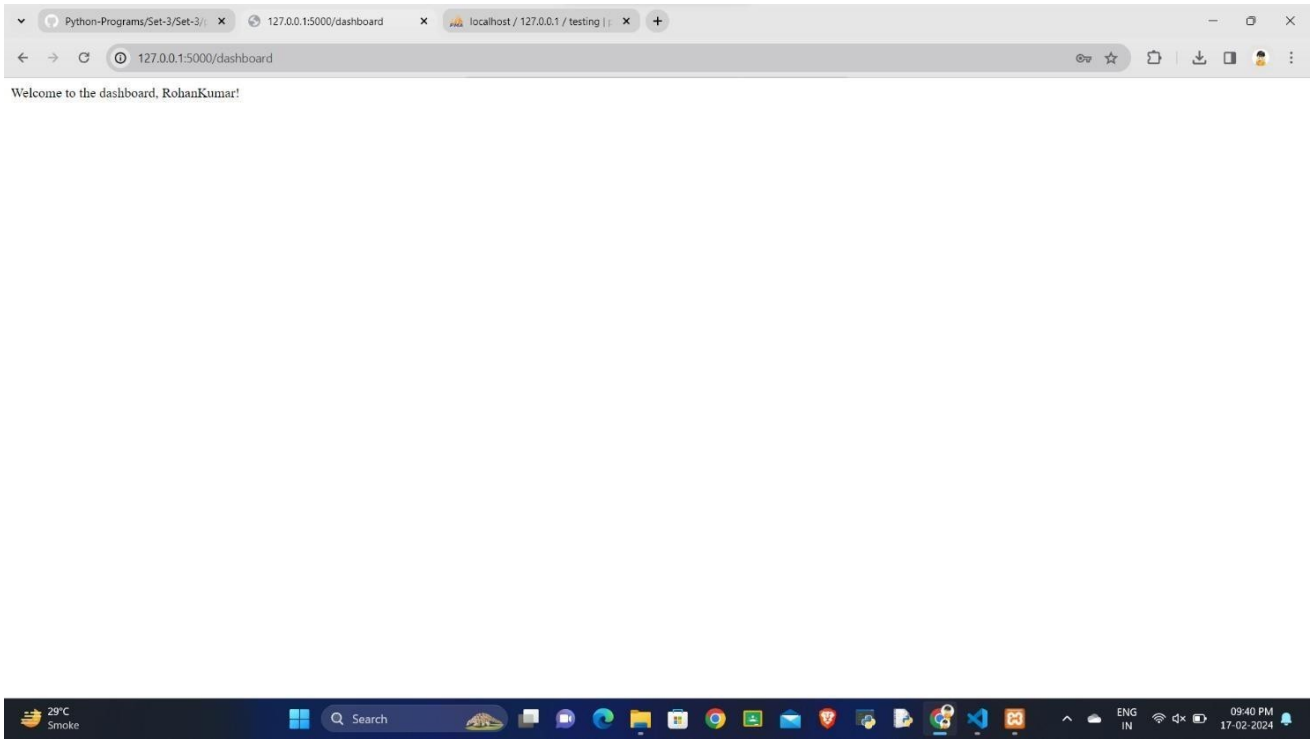
#Registation page



#Login page



#Dashboard



Conclusion:

Here We have successfully creates a web application that allows users to register and login.

Program No:3

Problem statement:

A program that creates a web application that allows users to upload and download files.

Aim:

To solve this problem statement.

Problem Description:

Develop a Flask-based web application featuring:

1. File Upload:

• Users can upload files through a web form.
• Prevent form submission without selecting a file.
• Save uploaded files to a server directory (e.g., uploads).

2. File Display:

• Display a dynamic list of uploaded files on the main page.
• Each file entry includes a "Download" button.

3. File Download:

• Enable users to download files by clicking the corresponding "Download" button.

4. User Interface:

• Design a clean and user-friendly interface using HTML templates.
• Separate application code (app.py) and HTML templates.

Algorithm:

Step 1: start
Step 2: create a Project Structure
Step 3: Write a HTML file and save as a index.htmlfile **Step 4:** Write a Flask file and save as a app.py file **Step 5:** Run that Flask file.
Step 6: Stop

HTML FILE

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>File Upload and Download</title>
</head>
<body>
   <h1>File Upload and Download</h1>
   <form action="/upload" method="post" enctype="multipart/form-data">
      <label for="file">Choose a file:</label>
      <input type="file" name="file" id="file" required>
      <br>
      <input type="submit" value="Upload">
   </form>

   <h2>Uploaded Files</h2>
   {% for filename in filenames %}
      <div>
         <span>{{ filename }}</span>
         <a href="{{ url_for('download_file', filename=filename) }}" download>
            <button>Download</button>
         </a>
      </div>
   {% endfor %}
</body>
</html>
```

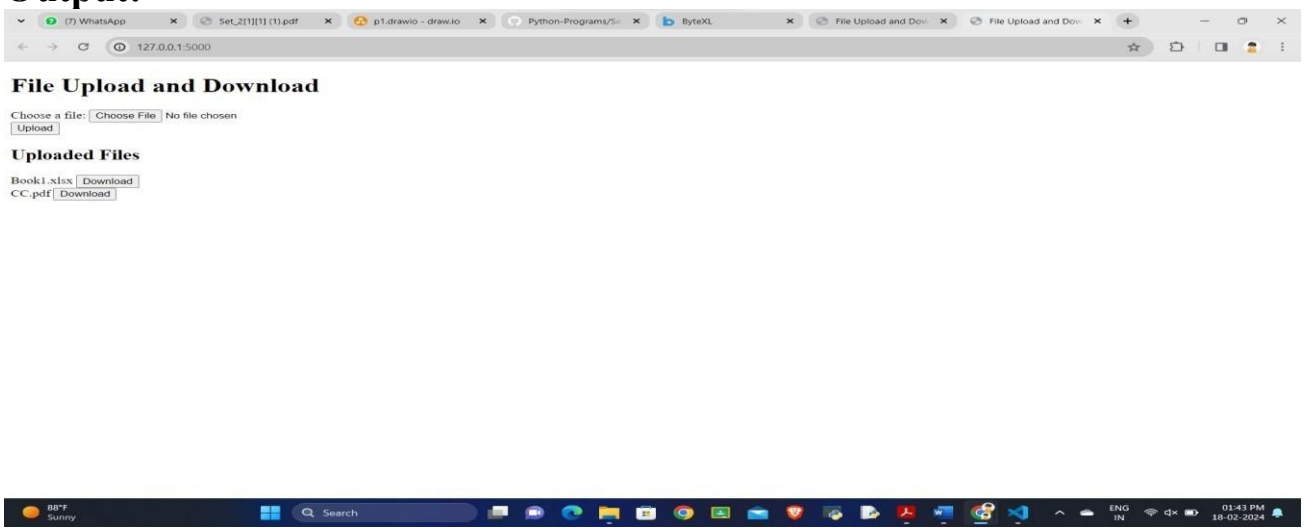
FLSK FILE

app.py

```
from flask import Flask, render_template, request, send_from_directory,
redirect, url_for
import os
app = Flask(__name__)
```

```
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
@app.route('/')
def index():
    filenames = os.listdir(app.config['UPLOAD_FOLDER'])
    return render_template('index.html', filenames=filenames)
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return "No file part"
file = request.files['file']
    if file.filename == '':
        return "No selected file"
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], file.filename))
    return redirect(url_for('index'))
@app.route('/download/<filename>')
def download_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)

if__name__== '__main
    __': app.run(debug=True)
```

Output:



Conclusion:

Here We have successfully creates a web application that allows users to upload and download files.

Program No:4

Problem statement:

A program that creates a web application that displays data from a database in a tabular format.

Aim:

To solve this problem statement.

Problem Description:

The program utilizes Flask, SQLAlchemy, and Pandas to create a web application. It defines a simple SQLAlchemy model (Person) to represent data with attributes like name and age. Sample data is inserted into an SQLite database. The main route (/) queries the database, converts the data to a Pandas DataFrame, and renders it as an HTML table using a Flask template (index.html).

Algorithm:

Step 1: start

Step 2: create a Project Structure

Step 3: Write a HTML file and save as a index.htmlfile **Step 4:** Write a Flask file and save as a app.py file **Step 5:** Run that Flask file.

Step 6: Stop

HTML FILE

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Data Display</title>
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
</head>
<body>
   <div class="container mt-5">
```

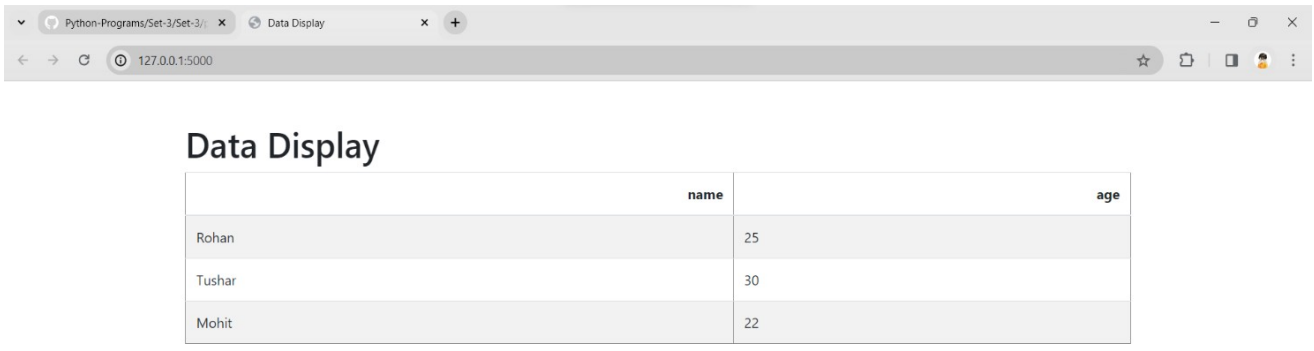
```
        <h1>Data Display</h1>
        <!-- Render the HTML table -->
        {{ table_html | safe }}
    </div>
</body>
</html>
```

FLSK FILE

app.py

```
from flask import Flask,
render_template import pandas as pd

app = Flask(__name__)
sample_data = {'name': ['John', 'Alice', 'Bob'],
            'age': [25, 30, 22]}
df = pd.DataFrame(sample_data)
@app.route('/')
def display_data():
    table_html = df.to_html(classes='table table-striped', index=False)
    return render_template('index.html', table_html=table_html)

if__name___== '__main
    __': app.run(debug=True)
```

Output:



Conclusion:

Here We have successfully creates a web application that displays data from a database in a tabular format.

Program No:5

Problem statement:

A program that creates a web application that accepts user input and sends it to a server-side script for processing.

Aim:

To solve this problem statement.

Problem Description:

You are tasked with creating a web application using Flask that enables users to input data on the main page through a form. The entered data should be sent to the server, processed by a server-side script, and the result displayed on the same page. The provided code includes a basic structure for achieving this, where user input is obtained from a form, and a simple processing logic is applied.

Algorithm:

Step 1: start
Step 2: create a Project Structure
Step 3: Write a HTML file and save as a index.htmlfile **Step 4:** Write a Flask file and save as a app.py file **Step 5:** Run that Flask file.
Step 6: Stop

HTML FILE
index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>User Input</title>
  </head>
  <style>
    * {
      margin: 0;
```

```
  padding: 0;
  box-sizing: border-box;
}
body {
  height: 100vh;
  width: 100%;
  background: #a2d2ff;
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction:
  column;
}
.container {
  display: flex;
  align-items: center;
  justify-content: space-
  evenly; flex-direction:
  column; width: 500px;
  height: 600px;
  border-radius:
  20px;
  background: #ffffff5a;
  backdrop-filter:
  blur(20px); & h1{
    font-family: Arial, Helvetica, sans-serif;
    color: #3a86ff;
    font-size: 2rem;
  }
  & label{
    color: #3a86ff;
    font-family: Arial, Helvetica, sans-
    serif; font-size: 1.2rem;
    padding: 10px;
    margin: 10px 20px;
  }
  & .enter{
    padding: 10px
    20px; border: none;
    outline: none;
```

```
border-radius:
20px;
```

```
      }
      & .submit{
        padding: 10px 20px;
        color: #fff;
        background: #2a9d8f;
        outline: none;
        border: none;
        border-radius:
        10px; transition:
        .3s;
        transform: translateX(150px);
        margin: 30px;
        &:hover{ colo
            r: #000;
            cursor: pointer;
            background: #fff;
        }
      }
      & h2{
        font-family: Arial, Helvetica, sans-serif;
        color: #3a86ff;
        font-size: 2rem;
      }
    }
  </style>
  <body>
    <div class="container">
      <h1>User Input Form</h1>
      <form method="post" action="/">
        <label for="user_input">Enter something:</label>
        <input type="text" class="enter" name="user_input" id="user_input"
required />
        <br />
        <input class="submit" type="submit" value="Submit" />
      </form>

      {% if result %}
      <div>
        <h2>Result:</h2>
        <p>{{ result }}</p>
```

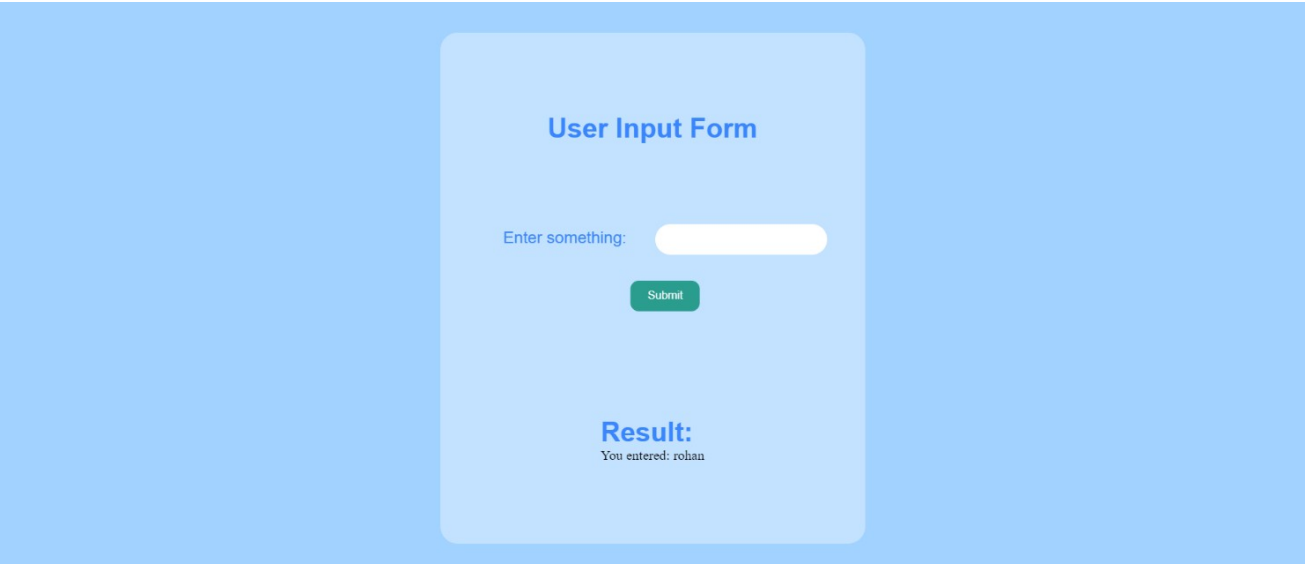
```
    </div>
    {% endif %}
  </div>
 </body>
</html>
```

FLSK FILE

app.py

```
from flask import Flask, render_template, request

app = Flask(__name__)
@app.route('/', methods=['GET', 'POST'])
def index():
    result = None
    if request.method == 'POST':
        # Get user input from the form
        user_input = request.form.get('user_input')
        result = f"You entered: {user_input}"
    return render_template('index.html', result=result)

if__name__== '__main
    __': app.run(debug=True)
```

Output:



Conclusion:

Here We have successfully creates a web application that accepts user input and sends it to a server-side script for processing.