A Practical Report Submitted in fulfillment of the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

Year 2022-2023

By

MR. ABHISHEK RAMAKRISHNAN

(Application ID: 72475)

Under guidance of

PROF. TRUPTI RONGARE

Institute of Distance and Open Learning,

Vidya Nagar, Kalina, Santacruz East- 400098.

University of Mumbai

INSTITUTE OF DISTANCE AND OPEN LEARNING

Vidya Nagari, Kalina, Santacruz East – 400098

CERTIFICATE

This is to certify that,

Mr. Abhishek Ramakrishnan, Application ID: **72475,**

Student of Master of Science in Computer Science has Satisfactorily

Completed the Practical in

Cyber and Information Security (Network Security).

Name	**Application ID**
Mr. Abhishek Ramakrishnan	72475

_____ _____

Subject In-charge **Examiner**

INDEX

Sr No.	Date	List of Practical	Signature
1.		Write a program to store username and password in an encrypted form in a database to implement integrity lock.	
2.		Write SQL query to retrieve sensitive information from less sensitive queries	
3.		Write SQL query to create a view to implement concept of views and commutative filter in distributed databases.	
4.		Write a program to implement SSL.	
5.		Write a program to send an encrypted email.	
6.		Write a program to digitally sign MIME to create an ‘opaque’ signature.	
7.		Write a program to generate DSA SSH key.	
8.		Write a program to implement multilevel security.	

PRACTICAL 1:
WRITE A PROGRAM TO STORE USERNAME AND PASSWORD IN AN ENCRYPTED FORM IN A DATABASE TO IMPLEMENT INTEGRITY LOCK.

Python Program:

```
#PRACTICAL 1:

import mysql.connector as ms

Mydb = ms.connect(
  host="localhost",
  user="root",
  password="student",
)

mycursor = mydb.cursor()
def sql_execute(statement):
    mycursor.execute(statement)
try:
    try:
        sql_execute("DROP DATABASE AR;")
    except:
        pass
    sql_execute("CREATE DATABASE AR;")
    mydb = ms.connect(
    host="localhost",
    user="root",
    password="student",
    database = 'ar'
    )

mycursor = mydb.cursor()
    sql_execute("CREATE TABLE USER (username varchar(255), password
varchar(255));")
    sql_execute("INSERT INTO USER (username,password) values
('Abhishek',ENCODE('GoodPassword@123',''));")
    sql_execute("INSERT INTO USER (username,password) values
('Mandar',ENCODE('worstpassword',''));")
except Exception as e:
    print(e)

mydb.commit()
sql_execute("SELECT * FROM USER;")

results = mycursor.fetchall()
print("User Table created successfully here are the values inside usertable:")
for x in results:
    print(x)
```

Output:

```
User Table created successfully here are the values inside usertable:
('Abhishek', '\x18¬§ClLÙ\xa0\x15ãð¹é\x02\x19›')
('Mandar', '\x1aæ~îÉq†½†2\x18Šw')
```

PRACTICAL 2:
WRITE SQL QUERY TO RETRIEVE SENSITIVE INFORMATION FROM LESS SENSITIVE QUERIES

SQL Query:

```
CREATE TABLE LOGIN(username varchar(255), password varchar(255));
INSERT INTO LOGIN(username,password) values
('Abhishek',aes_encrypt('MypasswordIsBest@123','secret'));
INSERT INTO LOGIN(username,password) values
('Mandar',aes_encrypt('worstpassword','secret'));
SELECT username as 'USER NAME',password as 'ENCRYPTED PASSWORD'FROM LOGIN;
SELECT username as 'USER NAME',aes_decrypt(password,'secret') as 'DECRYPTED
PASSWORD'FROM LOGIN;
```

Output:

```
mysql> CREATE TABLE LOGIN(username varchar(255), password varchar(255));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO LOGIN(username,password) values ('Abhishek',aes_encrypt('MypasswordIsBest@123','secret'));
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO LOGIN(username,password) values ('Mandar',aes_encrypt('worstpassword','secret'));
Query OK, 1 row affected (0.00 sec)

mysql> SELECT username as 'USER NAME',password as 'ENCRYPTED PASSWORD'FROM LOGIN;
+-----------+----------------------------------+
| USER NAME | ENCRYPTED PASSWORD               |
+-----------+----------------------------------+
| Abhishek  | y¥ⁿ¥OOðIfƒí HQ◄>▬♣Åⱼ≈> ╢▒ü        ┌+↕╛r |
| Mandar    |îπp:±¢@#DòR«=çN                 |
+-----------+----------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT username as 'USER NAME',aes_decrypt(password,'secret') as 'DECRYPTED PASSWORD'FROM LOGIN;
+-----------+----------------------+
| USER NAME | DECRYPTED PASSWORD   |
+-----------+----------------------+
| Abhishek  | MypasswordIsBest@123 |
| Mandar    | worstpassword        |
+-----------+----------------------+
2 rows in set (0.00 sec)
```

PRACTICAL 3:
WRITE SQL QUERY TO CREATE A VIEW TO IMPLEMENT CONCEPT OF VIEWS AND COMMUTATIVE FILTER IN DISTRIBUTED DATABASES.

SQL Query:

```
CREATE TABLE USERS(username varchar(255), lastname varchar(255));
INSERT INTO USERS(username,lastname) values ('Abhishek','Ramakrishnan');
INSERT INTO USERS(username,lastname) values ('Mandar','Bodane');
CREATE VIEW USERTABLEVIEW AS SELECT * FROM USERS WHERE USERNAME = 'Abhishek';
SELECT * FROM USERTABLEVIEW;
```

Output:

```
mysql> CREATE TABLE USERS(username varchar(255), lastname varchar(255));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO USERS(username,lastname) values ('Abhishek','Ramakrishnan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO USERS(username,lastname) values ('Mandar','Bodane');
Query OK, 1 row affected (0.00 sec)

mysql> CREATE VIEW USERTABLEVIEW AS SELECT * FROM USERS WHERE USERNAME = 'Abhishek';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM USERTABLEVIEW;
+----------+-------------+
| username | lastname    |
+----------+-------------+
| Abhishek | Ramakrishnan |
+----------+-------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM USERS;
+----------+-------------+
| username | lastname    |
+----------+-------------+
| Abhishek | Ramakrishnan |
| Mandar   | Bodane      |
+----------+-------------+
2 rows in set (0.00 sec)
```

PRACTICAL 4:
WRITE A PROGRAM TO IMPLEMENT SSL.

Python Program:

```
#Practical 4:
import socket
import ssl

hostname = input("Enter website to securely connect:")
context = ssl.create_default_context()

print("Website securely connected here are the details of the connection:")

with socket.create_connection((hostname, 443)) as sock:
    with context.wrap_socket(sock, server_hostname=hostname) as ssock:
        print(ssock)
        print(ssock.version())
```

Output:

```
Website securely connected here are the details of the connection:
<ssl.SSLSocket fd=1504, family=2, type=1, proto=0, laddr=('192.168.1.100', 1309), raddr=('142.250.182.196', 443)>
TLSv1.3
```

PRACTICAL 5:
WRITE A PROGRAM TO SEND AN ENCRYPTED EMAIL.

Python Program:

```
#PRACTICAL 5:
import smtplib, ssl
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

sender_email = "put_sender_email@xyz.com"
receiver_email = "put_reciever_email@xyz.com"
password = "put_password_here"
message = MIMEMultipart("alternative")
message["Subject"] = "Test Mail"
message["From"] = sender_email
message["To"] = receiver_email

# Create the plain-text and HTML version of your message
text = "This is a digitally encrypted mail"
# Turn these into plain/html MIMEText objects
part1 = MIMEText(text, "plain")

message.attach(part1)

# Create secure connection with server and send email
context = ssl.create_default_context()

with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
    server.login(sender_email, password)
    server.sendmail(
        sender_email, receiver_email, message.as_string()
    )
    print("Mail Sent!")
```

Output:

```
Mail sent!
```

PRACTICAL 6:
Write a program to digitally sign MIME to create an 'opaque' signature.

Python Program:

```
import os
import email
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.serialization import load_pem_private_key

def create_mime_message(sender, recipient, subject, body):
    msg = MIMEMultipart()
    msg["From"] = sender
    msg["To"] = recipient
    msg["Subject"] = subject
    msg.attach(MIMEText(body, "plain"))
    return msg

def sign_mime_message(mime_message, private_key_path):
    with open(private_key_path, "rb") as key_file:
        private_key = load_pem_private_key(key_file.read(), password=None,
backend=default_backend())

    signature = private_key.sign(
        mime_message.as_bytes(),
        hashes.SHA256()
    )

    return signature

if __name__ == "__main__":
    sender = "sender@example.com"
    recipient = "recipient@example.com"
    subject = "Test Email"
    body = "This is a digitally signed email."

    mime_message = create_mime_message(sender, recipient, subject, body)
    private_key_path = "privatekey.pem"

    signature = sign_mime_message(mime_message, private_key_path)

    mime_message["X-Signature"] = signature.hex()  # Add the signature as a
header to the MIME message
```

```
    # Send or save the mime_message with the added signature
    print(mime_message.as_string())
```

Output:

```
Content-Type: multipart/mixed; boundary="===============0103498092230740333=="
MIME-Version: 1.0
From: sender@example.com
To: recipient@example.com
Subject: Test Email
X-Signature:
302c02145217fa68a6cf6f1d437cfc020e1b31820a30a6ba021407228577e5f08ddd311a31ef7e
8d7aaa11c5ac55

--===============0103498092230740333==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

This is a digitally signed email.
--===============0103498092230740333==--
```

PRACTICAL 7:
WRITE A PROGRAM TO GENERATE DSA SSH KEY.

Python Program:

```
#PRACTICAL 7:

from Crypto.PublicKey import DSA

# Create a new DSA key

output = input()

key = DSA.generate(1024)
with open("{}.pri".format(output), "wb") as f:
    # Write the private key to file
    f.write(key.export_key('PEM'))
    print("Private Key written successfully!")

with open("{}.pub".format(output),"wb") as g:
    g.write(key.public_key().export_key('PEM'))
    print("Public Key written successfully!")

print(key.export_key())
print(key.public_key().exportKey())
```

Output:

```
Private Key written successfully!

Public Key written successfully!

b'-----BEGIN PRIVATE KEY-----
\nMIIBTAIBADCCASwGByqGSM44BAEwggEfAoGBAN1nceg37amfW+/JozwnDi4JFylX\nFv3s3WJEfx
zr9Eo1Y9hXMmxZhb/A/9JSD4vBZAoaMdbLxTF4MerzdVqfyrRuILH+\nzKkA+8y7rzpKez6tDnIcCY
EIWpdJQbTg2PY09dukOuitcqDL1tSPjkZIBpgmwpKv\nPU/x4eoz6bMg8KUpAhUA+n4hdhskRlu38l
bU4KqBB7eABE0CgYEAlwgjW1PyKdfZ\nASo4W/ZyeC1GXWXZ3mMll2dYCGZIirXjHC737WqkknKJZT
NBrrPXq/Mndz79rnzp\n65Cnx0/JrKwPvflGSH2vMY48pi2O9LM0x8NwVvmUW6TAoTuXrFO7Q/eTBJ
e/6RZk\nmxaG/+VhUnhGU4Gd7y2f9rOJH8Oo910EFwIVALF1+ILajBvh8FzPRUz4bBK8wkee\n----
-END PRIVATE KEY-----'


b'-----BEGIN PUBLIC KEY-----
\nMIIBtzCCASwGByqGSM44BAEwggEfAoGBAN1nceg37amfW+/JozwnDi4JFylXFv3s\n3WJEfxzr9E
o1Y9hXMmxZhb/A/9JSD4vBZAoaMdbLxTF4MerzdVqfyrRuILH+zKkA\n+8y7rzpKez6tDnIcCYEIWp
dJQbTg2PY09dukOuitcqDL1tSPjkZIBpgmwpKvPU/x\n4eoz6bMg8KUpAhUA+n4hdhskRlu38lbU4K
qBB7eABE0CgYEAlwgjW1PyKdfZASo4\nW/ZyeC1GXWXZ3mMll2dYCGZIirXjHC737WqkknKJZTNBrr
PXq/Mndz79rnzp65Cn\nx0/JrKwPvflGSH2vMY48pi2O9LM0x8NwVvmUW6TAoTuXrFO7Q/eTBJe/6R
ZkmxaG\n/+VhUnhGU4Gd7y2f9rOJH8Oo910DgYQAAoGAR5VEMhiLYhFFGeb5XbQm2ww4Rxub\n5fHO
h2bgmZNqAH24zcCfO6tugayDRcSGETMD7CCZx1uCFwGA1G/M/q1PyqsvsV3B\n0mgX2zLQEqdKCTSl
naOOzd7+ASu1Z0xzKswji7TZ2fUZLCYPKi5iy1JDqI0VzfCE\n+OFHQq6+yY8eBR0=\n-----END
PUBLIC KEY-----'
```

PRACTICAL 8:
WRITE A PROGRAM TO IMPLEMENT MULTILEVEL SECURITY.

Python Program:

```
#Practical 8

class User:
    def __init__(self, username, password, level):
        self.username = username
        self.password = password
        self.level = level

class SecuritySystem:
    def __init__(self):
        self.users = []

    def add_user(self, username, password, level):
        user = User(username, password, level)
        self.users.append(user)

    def authenticate(self, username, password):
        for user in self.users:
            if user.username == username and user.password == password:
                return user
        return None

    def has_access(self, user, required_level):
        return user.level >= required_level

if __name__ == "__main__":
    security_system = SecuritySystem()

    # Add users
    security_system.add_user("user1", "pass1", "Regular")
    security_system.add_user("user2", "pass2", "Admin")

    # User login
    username = input("Enter username: ")
    password = input("Enter password: ")

    user = security_system.authenticate(username, password)

    if user:
        print(f"Welcome, {user.username}! You are a {user.level} user.")
        if security_system.has_access(user, "Admin"):
            print("You have admin privileges.")
        else:
```

```
            print("You have regular user privileges.")
    else:
        print("Authentication failed.")
```

Output:

```
Welcome, user2! You are a/an Admin user.
You have admin privileges.
```