Madanapalle Institute Of Technology And Science

**(UGC-AUTONOMOUS)**

## (Affiliated to J.N.T.U.A, Anantapuramu and Approved by AICTE New Delhi) An ISO 9001:2008 Certified Institution

**Department of Computer Science and Engineering (Data Science)**

**CYPTOGRAPHY ALGORITHM**

**LABORAT0RY**



**Name of the Course Instructor : Mrs. M. NANDHINI**

**Subject & Code : Cryptography Algorithms(20CDS2609)**

**Academic Year/ Semester : 2023 – 24 / I Semester Class / Section : CDS**

**Year & Semester : IV& I**

**Vision**

The Department is committed to inculcate discipline, offering best Technical Education and Research Opportunities with inculcated discipline and ethically strong to meet the global challenges, who in turn shall contribute to the improvement and well-being of the society.

## Mission

* M1: Train the students in the most recent technologies of AI&DS.
* M2: Stimulating students to solve societal issues through AI techniques by inculcating values and ethics.
* M3: Enrich employability and entrepreneurial skills in the field of AI & DS through experiential and self-directed learning.

**Program Educational Objectives (PEOs)**

|  |  |
| --- | --- |
| PEO1 | Gain Successful Professional career in IT industry as an efficient software engineer. |
| PEO2 | Succeed in Master/Research programmes to gain knowledge on emerging technologies in Computer Science and Engineering. |
| PEO3 | Grow as a responsible computing professional in their own area of interest with intellectual skills and ethics through lifelong learning approach to meet societal needs. |

**Program Outcomes (POs)**

|  |  |
| --- | --- |
| PO1 | Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and engineering specialization to the solution of complex engineering problems. |
| PO2 | Problem Analysis: Identify, formulate, research literature, and analyze engineering problems to arrive at substantiated conclusions using first principles of  mathematics, natural, and engineering sciences. |
| PO3 | Design/development of solutions: Design solutions for complex engineering problems and design system components, processes to meet the specifications with consideration for the public health and safety, and the cultural, societal, and environmental considerations. |
| PO4 | Conduct investigations of complex Problems: Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |
| PO5 | Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |

|  |  |
| --- | --- |
| PO6 | The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| PO7 | Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| PO8 | Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| PO9 | Individual and team work: Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings. |
| PO10 | Communication: Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports  documentation. Make effective presentations, and give and receive clear instructions |
| PO11 | Project management and finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one’s own work, as a member and leader in a team. Manage projects in multidisciplinary environments. |
| PO12 | Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. |

**Program Specific Outcomes (PSOs)**

|  |  |
| --- | --- |
| PSO1 | Apply mathematical foundations, algorithmic principles and computing techniques in the modelling and design of computer-based systems |
| PSO2 | Design and develop software in the areas of relevance under realistic constraints |
| PSO3 | Analyze real world problems and develop computing solutions by applying concepts of Computer Science |

**Course Description:**

This course covers the principles and practices of cryptography. The fundamental topics of attacks, attack model, and few classical techniques are introduced. Symmetric and Asymmetric ciphers are illustrated. Message authentication and Hash functions are exemplified. Also, practical exposure on encryption algorithms, authentication techniques.

## Course Objectives:

This course enables students to

1. Understand the basic categories of threats to computers and networks

2. Learn the Symmetric cryptographic algorithms.

3. Learn the Asymmetric cryptographic algorithms

4. Understand cryptographic Hash Functions

5. Understand message authentication and Digital Signature.

## List of Programs:

1. Implementation of Caesar cipher.
2. Implementation of playfair cipher.
3. Implementation of Column transposition.
4. Implementation of AES.
5. Implementation of RSA Algorithm
6. Implementation of Diffie-Hellman Algorithm .
7. Implementation of RC4 Algorithm
8. Implementation of SHA-3 Algorithm
9. Implementation of HMAC
10. Implementation of DSS

## Course Outcomes:

1. Upon successful completion of the course, students will be able to 1. Implement classical Encryption Techniques 2. Apply symmetric key cryptographic algorithms 3. Experiment with various asymmetric key cryptographic algorithms 4. Execute stream cipher algorithms and hash algorithms 5. Make use of Authentication functions

**Text Books:**

## 1. “Cryptography and Network Security - Principles and Practice” by William Stallings, Pearson Education, 7th Edition, 2017.

## Reference Books:

1. “Cryptography and Network Security” by C K Shyamala, N Harini and Dr T R Padmanabhan, Wiley, 1st Edition, 2011.
2. “Cryptography and Network Security” by Forouzan and Mukhopadhyay, McGraw Hill, 2nd Edition, 2010.
3. “Cryptography and Network Security” by Atul Kahate, McGraw Hill, 3 rd Edition, 2017.

**Mode of Evaluation**: Continuous Internal Evaluation and End Semester Examination

# INDEX

|  |  |  |
| --- | --- | --- |
| **Week** | **List of Experiments** | **Page No** |
| **1** | Implementation of Caesar cipher. | **7-9** |
| **2** | Implementation of playfair cipher. | **10-12** |
| **3** | Implementation of Column transposition. | **13-14** |
| **4** | Implementation of AES. | **15-17** |
| **5** | Implementation of RSA Algorithm | **18-19** |
| **6** | Implementation of Diffie-Hellman Algorithm . | **20-22** |
| **7** | Implementation of RC4 Algorithm | **23-27** |
| **8** | Implementation of SHA-3 Algorithm | **28-29** |
| **9** | Implementation of HMAC | **30-32** |
| **10** | Implementation of DSS | **33-34** |

**Implementation of Caesar cipher**

**AIM:**

To implement a Caesar cipher substitution technique inJava.

**ALGORITHM:**

1. Assign the 26 letters in alphabet to the variable named ALPHABET.

2. Convert the plaintext letters into lowercase.

3. To encrypt a plaintext letter, the first set of plaintext letters and slides it to LEFT by the number of positions of the secret shift.

4. The plaintext letter is then encrypted to the ciphertext letter on the sliding ruler underneath.

5. On receiving the ciphertext, the receiver who also knows the secret shift, positions his sliding ruler underneath the ciphertext alphabet and slides it to RIGHT by the agreed shift number, 3 in this case.

6. Then replaces the ciphertext letter by the plaintext letter on the sliding ruler underneath.

**PROGRAM**

#Ceaser Cipher

def encrypt(text,s):

  result=""

  for i in range(len(text)):

    char=text[i]

    if char.isupper():

      result+=chr((ord(char)+s-65)%26+65)

    else:

      result+=chr((ord(char)+s-97)%26+97)

  return result

def decrypt(text,s):

  result=""

  for i in range(len(text)):

    char=text[i]

    if char.isupper():

      result+=chr((ord(char)-s-65)%26+65)

    else:

      result+=chr((ord(char)-s-97)%26+97)

  return result

text='ATTACKONCE'

s=4

print("Text:"+text)

print("Shift:"+str(s))

print("Cipher:"+encrypt(text,s))

print("original text:"+decrypt(encrypt(text,s),s))

OUTPUT:

Text:ATTACKONCE

Shift:4

Cipher:EXXEGOSRGI

original text:ATTACKONCE

**RESULT:**

**Thus the Ceaser cipher substitution technique was implemented and executed**

**Successfully.**

**PLAYFAIR CIPHER**

**AIM:** To implement a Playfair cipher substitution technique in Java.

**ALGORITHM:**

1. Read the keyword.

2. Then create the key table of 5x5 grid of alphabets.

3. Read the word to encrypt.

4. Ifthe input word should be even and then process it.

5. Then the plaintext message is split into pairs of two letters (digraphs).

6. Ifboth the letters are in the same column, take the letter below each one.

7. Ifboth letters are in the same row, take the letter to the right of each one.

8. If neither of the preceding two rules are true, form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

**PROGRAM:**

#Playfair Cipher

def toLowerCase(text):

  return text.lower()

def removeSpaces(text):

  newText = ""

  for i in text:

    if i == " ":

      continue

    else:

      newText = newText + i

  return newText

def Diagraph(text):

  Diagraph = []

  group = 0

  for i in range(2, len(text), 2):

    Diagraph.append(text[group:i])

    group = i

  Diagraph.append(text[group:])

  return Diagraph

def FillerLetter(text):

  k = len(text)

  if k % 2 == 0:

    for i in range(0, k, 2):

      if text[i] == text[i+1]:

        new\_word = text[0:i+1] + str('x') + text[i+1:]

        new\_word = FillerLetter(new\_word)

        break

      else:

        new\_word = text

  else:

    for i in range(0, k-1, 2):

      if text[i] == text[i+1]:

        new\_word = text[0:i+1] + str('x') + text[i+1:]

        new\_word = FillerLetter(new\_word)

        break

      else:

        new\_word = text

  return new\_word

list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',

    'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

def generateKeyTable(word, list1):

  key\_letters = []

  for i in word:

    if i not in key\_letters:

      key\_letters.append(i)

  compElements = []

  for i in key\_letters:

    if i not in compElements:

      compElements.append(i)

  for i in list1:

    if i not in compElements:

      compElements.append(i)

  matrix = []

  while compElements != []:

    matrix.append(compElements[:5])

    compElements = compElements[5:]

  return matrix

def search(mat, element):

  for i in range(5):

    for j in range(5):

      if(mat[i][j] == element):

        return i, j

def encrypt\_RowRule(matr, e1r, e1c, e2r, e2c):

  char1 = ''

  if e1c == 4:

    char1 = matr[e1r][0]

  else:

    char1 = matr[e1r][e1c+1]

  char2 = ''

  if e2c == 4:

    char2 = matr[e2r][0]

  else:

    char2 = matr[e2r][e2c+1]

  return char1, char2

def encrypt\_ColumnRule(matr, e1r, e1c, e2r, e2c):

  char1 = ''

  if e1r == 4:

    char1 = matr[0][e1c]

  else:

    char1 = matr[e1r+1][e1c]

  char2 = ''

  if e2r == 4:

    char2 = matr[0][e2c]

  else:

    char2 = matr[e2r+1][e2c]

  return char1, char2

def encrypt\_RectangleRule(matr, e1r, e1c, e2r, e2c):

  char1 = ''

  char1 = matr[e1r][e2c]

  char2 = ''

  char2 = matr[e2r][e1c]

  return char1, char2

def encryptByPlayfairCipher(Matrix, plainList):

  CipherText = []

  for i in range(0, len(plainList)):

    c1 = 0

    c2 = 0

    ele1\_x, ele1\_y = search(Matrix, plainList[i][0])

    ele2\_x, ele2\_y = search(Matrix, plainList[i][1])

    if ele1\_x == ele2\_x:

      c1, c2 = encrypt\_RowRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

    elif ele1\_y == ele2\_y:

      c1, c2 = encrypt\_ColumnRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

    else:

      c1, c2 = encrypt\_RectangleRule(

        Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

    cipher = c1 + c2

    CipherText.append(cipher)

  return CipherText

text\_Plain = 'instruments'

text\_Plain = removeSpaces(toLowerCase(text\_Plain))

PlainTextList = Diagraph(FillerLetter(text\_Plain))

if len(PlainTextList[-1]) != 2:

  PlainTextList[-1] = PlainTextList[-1]+'z'

key = "Monarchy"

print("Key text:", key)

key = toLowerCase(key)

Matrix = generateKeyTable(key, list1)

print("Plain Text:", text\_Plain)

CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)

CipherText = ""

for i in CipherList:

  CipherText += i

print("CipherText:", CipherText)

**OUTPUT:**

Key text: Monarchy

Plain Text: instruments

CipherText: gatlmzclrqtx

## RESULT:

## Thus the Playfair cipher substitution technique was implemented and executed

## successfully

**Implementation of Column transposition.**

**AIM:** To implement a Playfair cipher substitution technique inJava.

**ALGORITHM:**

1. Consider the plain text hello world, and let us apply the simple columnar transposition

technique as shown below

h e l l

o w o r

l d

2. The plain text characters are placed horizontally and the cipher text is created with

vertical format as: holewdlo lr.

3. Now, the receiver has to use the same table to decrypt the cipher text to plain text.

***PYTHON***

"""

Created on Wed Nov 15 10:54:53 2023

@author: Nandhini

"""

def columnar(plaintext,key):

values={}

seqlist=[]

nextvalue=1

indices=rand(len(key))

for letter in plaintext:

for i in indices:

if letter==key[i]:

values[i]=nextvalue

nextvalue=nextvalue+1

for i in indices:

seqlist.append(values[i])

return seqlist

def encode(txt,key):

sz = len(key) # how big are the columns

cols = list(map("".join,zip(\*zip(\*[iter(txt)]\*sz)))) # list partitioned into columns

return "".join([cols[key.index(str(c))] for c in range(1,sz+1)])

encoded = encode("IHAVETWOCATS","3124")

print (encoded)

def split\_len(seq, length):

return [seq[i:i + length] for i in range(0, len(seq), length)]

def encode(key, plaintext):

order = {

int(val): num for num, val in enumerate(key)

}

ciphertext = ''

for index in sorted(order.keys()):

for part in split\_len(plaintext, len(key)):

try:

ciphertext += part[order[index]]

except IndexError:

continue

return ciphertext

print(encode('3214', 'IHAVETWOCATS'))

#>>> HTAAWTIECVOS

**OUTPUT:**

runfile('H:/python practise/untitled3.py', wdir='H:/python practise')

HTAAWTIECVOS

AWTHTAIECVOS

RESULT:

Thus the Column Transposition technique was implemented and executed

successfully

**Implementation of AES.**

**AIM:**

To apply Advanced Encryption Standard (AES) Algorithm for a practical application

like URL Encryption.

**ALGORITHM:**

1. AES is based on a design principle known as a substitution–permutation.

2. AES does not use a Feistel network like DES, it uses variant of Rijndael.

3. It has a fixed block size of128 bits, and a key size of 128, 192, or 256 bits.

4. AES operates on a 4 × 4 column-major order array of bytes, termed the state

**PROGRAM:**

pip install pycryptodome

Collecting pycryptodome

Downloading pycryptodome-3.19.0-cp35-abi3-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (2.1 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 25.3 MB/s eta 0:00:00

Installing collected packages: pycryptodome

Successfully installed pycryptodome-3.19.0

PROGRAM:

#Advanced Encryption Standard

#Using ECB(Electronic CodeBook) mode

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad,unpad

Block\_Size=32

def aes\_encrypt(plaintext,key):

  cipher=AES.new(key,AES.MODE\_ECB)

  ciphertext=cipher.encrypt(pad(plaintext.encode(),Block\_Size))

  return ciphertext

def aes\_decrypt(ciphertext,key):

  cipher=AES.new(key,AES.MODE\_ECB)

  decrypttext=cipher.decrypt(ciphertext)

  return decrypttext.decode()

plaintext="HELLO123"

key = b'AESKey1234567890'

ciphertext = aes\_encrypt(plaintext, key)

print("Ciphertext:", ciphertext.hex())

decrypted\_text = aes\_decrypt(ciphertext, key)

print("Decrypted Text:", decrypted\_text)

OR

#AES using CBC mode which is more secure than ECB mode

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad,unpad

Block\_Size=16

def aes\_encrypt(plaintext,key,iv):

  cipher=AES.new(key,AES.MODE\_CBC,iv)

  ciphertext=cipher.encrypt(pad(plaintext.encode(),Block\_Size))

  return ciphertext

def aes\_decrypt(ciphertext,key,iv):

  cipher=AES.new(key,AES.MODE\_CBC,iv)

  decrypttext=unpad(cipher.decrypt(ciphertext),Block\_Size)

  return decrypttext.decode()

plaintext="HELLO123"

key = b'AESKey1234567890'

iv=get\_random\_bytes(Block\_Size)

ciphertext = aes\_encrypt(plaintext, key,iv)

print("Ciphertext:", ciphertext.hex())

decrypted\_text = aes\_decrypt(ciphertext, key,iv)

print("Decrypted Text:", decrypted\_text)

OUTPUT:

Ciphertext: 448639e5802a1ec13dcfa7189c0b31c6

Decrypted Text: HELLO123

**OUTPUT:**

Ciphertext: b48dcf62e2c913c89228fc98a4cab2f4219ed6ff0ff8bcd543f1edf670ec7cc2

Decrypted Text: HELLO123

**RESULT:**

Thus the java program for applying Advanced EncryptionStandard (AES) Algorithm

for a practical application of URL encryption is written and executed successfully

**Implementation of RSA Algorithm**

**AIM:**

To implement a RSA algorithm using HTML and Javascript.

**ALGORITHM:**

1. Choose two prime number p and q.

2. Compute the value of n and t.

3. Find the value of public key e.

4. Compute the value of private key d.

5. Do the encryption and decryption

a. Encryption is given as,

c = t

e mod n

b. Decryption is given as,

t = c

d mod n

**PROGRAM**

#Rivest-Shamir-Adelman

import math

p=3

q=7

n=p\*q

e=2

phi=(p-1)\*(q-1)

while(e<phi):

  if math.gcd(e,phi)==1:

    break

  else:

    e+=1

k=2

d=(1+(k\*phi))/e

msg=12.0

print("Message data:",msg)

c=pow(msg,e)

c=math.fmod(c,n)

print("Encrypted data:",c)

m=pow(c,d)

m=math.fmod(m,n)

print("Original message sent:",m)

OUTPUT:

Message data: 12.0

Encrypted data: 3.0

Original message sent: 12.0

**Implementation of Diffie-Hellman Algorithm .**

**AIM:**

To implement a Diffie-Hellman Key Exchange algorithm.

**ALGORITHM:**

1. Sender and receiver publicly agree to use a modulus p and base g which is a primitive

root modulo p.

2. Sender chooses a secret integer x then sends Bob R1 = g

x mod p

3. Receiver chooses a secret integer y, then sends Alice R2 = g

y mod p

4. Sender computes k1 = B

x mod p

5. Receiver computes k2 = A

y mod p

6. Sender and Receiver now share a secret key.

**PROGRAM**

#Diffie-Hellman

import random

p=23

q=5

def compute\_public\_key(private\_key):

  return (q\*\*private\_key)%p

private\_key\_a=random.randint(1,p-1)

private\_key\_b=random.randint(1,p-1)

public\_key\_a=compute\_public\_key(private\_key\_a)

public\_key\_b=compute\_public\_key(private\_key\_b)

shared\_secret\_a=(public\_key\_b\*\*private\_key\_a)%p

shared\_secret\_b=(public\_key\_a\*\*private\_key\_b)%p

print(shared\_secret\_a)

print(shared\_secret\_b)

**OUTPUT:**

12

12

**Implementation of RC4 Algorithm**

**AIM:**

To implement a RC4 algorithm.

**ALGORITHM:**

**PROGRAM:**

#RC-4

def key\_scheduling(key):

    sched = [i for i in range(0, 256)]

    i = 0

    for j in range(0, 256):

        i = (i + sched[j] + key[j % len(key)]) % 256

        tmp = sched[j]

        sched[j] = sched[i]

        sched[i] = tmp

    return sched

def stream\_generation(sched):

    stream = []

    i = 0

    j = 0

    while True:

        i = (1 + i) % 256

        j = (sched[i] + j) % 256

        tmp = sched[j]

        sched[j] = sched[i]

        sched[i] = tmp

        yield sched[(sched[i] + sched[j]) % 256]

def encrypt(text, key):

    text = [ord(char) for char in text]

    key = [ord(char) for char in key]

    sched = key\_scheduling(key)

    key\_stream = stream\_generation(sched)

    ciphertext = ''

    for char in text:

        enc = str(hex(char ^ next(key\_stream))).upper()

        ciphertext += (enc)

    return ciphertext

def decrypt(ciphertext, key):

    ciphertext = ciphertext.split('0X')[1:]

    ciphertext = [int('0x' + c.lower(), 0) for c in ciphertext]

    key = [ord(char) for char in key]

    sched = key\_scheduling(key)

    key\_stream = stream\_generation(sched)

    plaintext = ''

    for char in ciphertext:

        dec = str(chr(char ^ next(key\_stream)))

        plaintext += dec

    return plaintext

if \_\_name\_\_ == '\_\_main\_\_':

    ed = input('Enter E for Encrypt, or D for Decrypt: ').upper()

    if ed == 'E':

        plaintext = input('Enter your plaintext: ')

        key = input('Enter your secret key: ')

        result = encrypt(plaintext, key)

        print('Result: ')

        print(result)

    elif ed == 'D':

        ciphertext = input('Enter your ciphertext: ')

        key = input('Enter your secret key: ')

        result = decrypt(ciphertext, key)

        print('Result: ')

        print(result)

    else:

        print('Error in input - try again.')

**Output :**

Enter E for Encrypt, or D for Decrypt: E

Enter your plaintext: 001010010010

Enter your secret key: 101001000001

Result:

0X640XD10X150X1A0X800XE80X220X6F0XB80XF80X200XC0

**Result:**

**Implemmentation of HMAC**

**AIM:**

**ALGORITHM:**

**PROGRAM:**

#HMAC

import hmac

import hashlib

def generate(key,message):

  hash\_function=hashlib.sha256

  key=bytes(key,'utf-8')

  hmac\_res=hmac.new(key,bytes(message,'utf-8'),digestmod=hash\_function)

  hmac\_hex=hmac\_res.hexdigest()

  return hmac\_hex

secret\_key='my\_secret\_key'

data\_to\_hash="Hello, HMAC!"

res=generate(secret\_key,data\_to\_hash)

print("Input Message:",data\_to\_hash)

print("Secret Key:",secret\_key)

print("HMAC Output:",res)

**OUTPUT:**

Input Message: Hello, HMAC!

Secret Key: my\_secret\_key

HMAC Output: 2135134b3d2d6ac11a8eb3f6340dac59282a4edbb21ba861772ab2cb3ca56942

**Implementation of SHA-3 Algorithm**

**PROGRAM:**

#SHA-3

import sys

import hashlib

if sys.version\_info<(3,6):

  import sha3

s=hashlib.sha3\_224()

print(s.name)

print(s.digest\_size)

s.update(b"Geeks")

print(s.hexdigest())

**OUTPUT:**

sha3\_224

28

a32d875e701e4b6b0c582d39562fa989e1f4b6dfc17e45ad22fee234

**Implementation of DSS(DIGITAL SIGNATURE SCHEME)**

AIM:

To implement the signature scheme - Digital Signature Standard.

ALGORITHM:

1. Declare the class and required variables.

2. Create the object for the class in the main program.

3. Access the member functions using the objects.

4. Implement the SIGNATURE SCHEME - Digital Signature Standard.

5. It uses a hash function.

6. The hash code is provided as input to a signature function along with a random

number K generated for the particular signature.

7. The signature function also depends on the sender„s private key.

8. The signature consists of two components.

9. The hash code of the incoming message is generated.

10.The hash code and signature are given as input to a verification function.

## PROGRAM

#DSA-Digital Signature Standard

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.asymmetric import dsa

from cryptography.hazmat.backends import default\_backend

def generate\_dsa\_key():

  private\_key=dsa.generate\_private\_key(key\_size=1024,backend=default\_backend)

  public\_key=private\_key.public\_key()

  return private\_key,public\_key

def sign\_message(private\_key,message):

  signature=private\_key.sign(message,hashes.SHA256())

  return signature

def verify\_signature(public\_key,signature,message):

  try:

    public\_key.verify(signature,message,hashes.SHA256())

    return True

  except dsa.InvalidSignature:

    return False

private\_key,public\_key=generate\_dsa\_key()

message=b"Hello,DSA!"

signature=sign\_message(private\_key,message)

verification\_result=verify\_signature(public\_key,signature,message)

print("Message:",message.decode('utf-8'))

print("Signature:",signature)

print("Verification Result:", verification\_result)

## OUTPUT:

Message: Hello,DSA!

Signature: b'0.\x02\x15\x00\xba\xdb\xff\xf5H\xf3\x88\xb1\xafQC\xa4)R\x01I\xac\xb0\xc2K\x02\x15\x00\xf4\xfe\xd3\xb8\x9bzv\xee\xda\xf3\xb9\x86\xf2\x87\xa1\xd0\x07\x00\xc0|'

## Verification Result: True

## RESULT:

## Thus the Digital Signature Standard Signature Scheme has been implemented and

## executed successfully.