Dhaval Patel
M.Sc. (Cyber Security) Sem-3
Er no: 032200300002034

ECDSA Implementation

Source code:-

```
p = pow(2, 255) - 19

base =
15112221349535400772501151409588531511454012693041857206046113283949847762
202,
46316835694926478169428394003475163141307993866256225615783033603165251855
960

# Function for finding positive modulus
# of the number
def findPositiveModulus(a, p):
if a < 0:
a = (a + p * int(abs(a)/p) + p) % p
return a

# Function for typecasting from
# string to int
def textToInt(text):
encoded_text = text.encode('utf-8')
hex_text = encoded_text.hex()
int_text = int(hex_text, 16)
return int_text

# Function to find greatest
# common divisor(gcd) of a and b
def gcd(a, b):
while a != 0:
a, b = b % a, a
return b

# Function to find the modular inverse
# of a mod m
def findModInverse(a, m):

if a < 0:
a = (a + m * int(abs(a)/m) + m) % m

# no mod inverse if a & m aren't
```

```
# relatively prime
if gcd(a, m) != 1:
return None

# Calculate using the Extended
# Euclidean Algorithm:
u1, u2, u3 = 1, 0, a
v1, v2, v3 = 0, 1, m
while v3 != 0:

# // is the integer division operator
q = u3 // v3
v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q * v2), (u3 - q * v3), v1, v2, v3
return u1 % m

def applyDoubleAndAddMethod(P, k, a, d, mod):

additionPoint = (P[0], P[1])

# 0b1111111001
kAsBinary = bin(k)

# 1111111001
kAsBinary = kAsBinary[2:len(kAsBinary)]
# print(kAsBinary)

for i in range(1, len(kAsBinary)):
currentBit = kAsBinary[i: i+1]

# always apply doubling
additionPoint = pointAddition(additionPoint, additionPoint, a, d, mod)

if currentBit == '1':

# add base point
additionPoint = pointAddition(additionPoint, P, a, d, mod)

return additionPoint

# Function to calculate the point addition
def pointAddition(P, Q, a, d, mod):
x1 = P[0]; y1 = P[1]
x2 = Q[0]; y2 = Q[1]

x3 = (((x1*y2 + y1*x2) % mod) * findModInverse(1+d*x1*x2*y1*y2, mod)) % mod
y3 = (((y1*y2 - a*x1*x2) % mod) * findModInverse(1- d*x1*x2*y1*y2, mod)) % mod

return x3, y3

# ax^2 + y^2 = 1 + dx^2y^2
```

```
# ed25519
a = -1; d = findPositiveModulus(-121665 * findModInverse(121666, p), p)

# print("curve: ",a,"x^2 + y^2 = 1 + ",d,"x^2 y^2")
x0 = base[0]; y0 = base[1]

print("----------------------")
print("Key Generation: ")

# privateKey = 47379675103498394144858916095175689
# 779086087640336534911165206022228115974270 #32 byte secret key
import random
privateKey = random.getrandbits(256) #32 byte secret key

# print("private key: ",privateKey)
publicKey = applyDoubleAndAddMethod(base, privateKey, a, d, p)
print("public key: ", publicKey)

message = textToInt("Hello, world!")

# Function for hashing the message
def hashing(message):
import hashlib
return int(hashlib.sha512(str(message).encode("utf-8")).hexdigest(), 16)

# ----------------------------------------
# sign
r = hashing(hashing(message) + message) % p
R = applyDoubleAndAddMethod(base, r, a, d, p)

h = hashing(R[0] + publicKey[0] + message) % p

# % p
s = (r + h * privateKey)

print("----------------------")
print("Signing:")
print("message: ",message)
print("Signature (R, s)")
print("R: ",R)
print("s: ",s)


# ------------------------------------
# verify
h = hashing(R[0] + publicKey[0] + message) % p

P1 = applyDoubleAndAddMethod(base, s, a, d, p)

P2 = pointAddition(R, applyDoubleAndAddMethod(publicKey, h, a, d, p), a, d, p)
```

```
print("--------------------")
print("Verification:")
print("P1: ",P1)
print("P2: ",P2)
print("--------------------")
print("result")
if P1[0] == P2[0] and P1[1] == P2[1]:
print("The Signature is valid")
else:
print("The Signature violation detected!")
# --------------------------------
```

Output:

```
Windows PowerShell
PS C:\Users\babbu\OneDrive\Desktop> python ecdsa.py
----------------------
Key Generation:
public key:  (25551851118968282033404536051616419678018911547843369430826500574009915131328, 29587546356156344005872748224416128533035851296979615498975310461615277011220)
----------------------
Signing:
message:  5735816763073854953388147237921
Signature (R, s)
R:  (17398582206386920242697161388728121547970811789911961114394435433884805588241, 4477075765930951269674344356261240319783403544001405833534737085436262538822)
s:  678339085240980469204231768890240906990054281208589814529072838162498785349301342769760132916602637353059729651124256858822282927309263526221198991194611
----------------------
Verification:
P1:  (22271374672250125746505957296811514988937184259857410479062134509364611410517, 55952881145206732356571826634519630448830265572978866080518303798280392111780)
P2:  (22271374672250125746505957296811514988937184259857410479062134509364611410517, 55952881145206732356571826634519630448830265572978866080518303798280392111780)
----------------------
result
The Signature is valid
PS C:\Users\babbu\OneDrive\Desktop> |
```