Code:

DH:

```
P = 23
# A primitive root for P, G is taken
G = 9
 print('The Value of P is :%d'%(P))
 print('The Value of G is :%d'%(G))
 # Alice will choose the private key a
   a = 4
   print('The Private Key a for Alice is :%d'%(a))
  # gets the generated key
   x = int(pow(G,a,P))
  # Bob will choose the private key b
   b = 3
   print('The Private Key b for Bob is :%d'%(b))
   # gets the generated key
   y = int(pow(G,b,P))
   # Secret key for Alice
   ka = int(pow(y,a,P))
   # Secret key for Bob
   kb = int(pow(x,b,P))
```

```
    print('Secret key for the Alice is : %d'%(ka))
    print('Secret Key for the Bob is : %d'%(kb))
```

RSA:

```
# Write Python3 code here
from decimal import Decimal


def gcd(a,b):
        if b==0:
                return a
        else:
                return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)


for e in range(2,t):
        if gcd(e,t)== 1:
                break
```

```
for i in range(1,10):
        x = 1 + i*t
        if x % e == 0:
                d = int(x/e)
                break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n

dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text = '+str(dt))
```

RSA:(hash value verification)

```
def euclid(m, n):

    if n == 0:
        return m
    else:
```

```
        r = m % n
        return euclid(n, r)



# Program to find
# Multiplicative inverse
def exteuclid(a, b):

    r1 = a
    r2 = b
    s1 = int(1)
    s2 = int(0)
    t1 = int(0)
    t2 = int(1)

    while r2 > 0:

        q = r1//r2
        r = r1-q * r2
        r1 = r2
        r2 = r
        s = s1-q * s2
```

```
        s1 = s2

        s2 = s

        t = t1-q * t2

        t1 = t2

        t2 = t


    if t1 < 0:

        t1 = t1 % a


    return (r1, t1)


# Enter two large prime

# numbers p and q

p = 823

q = 953

n = p * q

Pn = (p-1)*(q-1)


# Generate encryption key

# in range 1<e<Pn

key = []
```

```
for i in range(2, Pn):

    gcd = euclid(Pn, i)

    if gcd == 1:
        key.append(i)


# Select an encryption key
# from the above list
e = int(313)

# Obtain inverse of
# encryption key in Z_Pn
r, d = exteuclid(Pn, e)
if r == 1:
    d = int(d)
    print("decryption key is: ", d)

else:
    print("Multiplicative inverse for\
    the given encryption key does not \
```

```
    exist. Choose a different encrytion key ")



# Enter the message to be sent
M = 19070

# Signature is created by Alice
S = (M**d) % n

# Alice sends M and S both to Bob
# Bob generates message M1 using the
# signature S, Alice's public key e
# and product n.
M1 = (S**e) % n

# If M = M1 only then Bob accepts
# the message sent by Alice.

if M == M1:
    print("As M = M1, Accept the\
    message sent by Alice")
else:
```

```
    print("As M not equal to M1,\
    Do not accept the message\
    sent by Alice ")
```

Elgamal:

```
# Python program to illustrate ElGamal encryption
import random
from math import pow
a = random.randint(2, 10)
def gcd(a, b):
	if a < b:
		return gcd(b, a)
	elif a % b == 0:
		return b;
	else:
		return gcd(b, a % b)
# Generating large random numbers
def gen_key(q):
	key = random.randint(pow(10, 20), q)
	while gcd(q, key) != 1:
		key = random.randint(pow(10, 20), q)
	return key
```

```
# Modular exponentiation

def power(a, b, c):

        x = 1

        y = a

        while b > 0:

                if b % 2 == 0:

                        x = (x * y) % c;

                y = (y * y) % c

                b = int(b / 2)

        return x % c

# Asymmetric encryption

def encrypt(msg, q, h, g):

en_msg = []

        k = gen_key(q)# Private key for sender

        s = power(h, k, q)

        p = power(g, k, q)



        for i in range(0, len(msg)):

                en_msg.append(msg[i])



        print("g^k used : ", p)

        print("g^ak used : ", s)
```

```
    for i in range(0, len(en_msg)):
        en_msg[i] = s * ord(en_msg[i])

    return en_msg, p

def decrypt(en_msg, p, key, q):

    dr_msg = []
    h = power(p, key, q)
    for i in range(0, len(en_msg)):
        dr_msg.append(chr(int(en_msg[i]/h)))

    return dr_msg

# Driver code
def main():

    msg = 'encryption'
    print("Original Message :", msg)

    q = random.randint(pow(10, 20), pow(10, 50))
    g = random.randint(2, q)
```

```
key = gen_key(q)# Private key for receiver
h = power(g, key, q)
print("g used : ", g)
print("g^a used : ", h)

en_msg, p = encrypt(msg, q, h, g)
dr_msg = decrypt(en_msg, p, key, q)
dmsg = ''.join(dr_msg)
print("Decrypted Message :", dmsg);
```

Hashing :

```
int_val = 37
str_val = 'LandTemperatureUncertainity'
flt_val = 24.56
print ("The integer hash value is : " + str(hash(int_val)))
print ("The string hash value is : " + str(hash(str_val)))
print ("The float hash value is : " + str(hash(flt_val)))
```