# RSA-KEM-OAEP

In [1]:
```
def rprime(l):
        return random_prime(2**l-1,True,2**(l-1))
```

In [2]:
```
l = 1024
q = rprime(l)
p = rprime(l+1)


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


G = IntegerModRing(phi)
R = IntegerModRing(N)


def generateKeys():
    e = G(rprime(512)) #public exponent
    s = 1/e #private exponent
    return (e,s)

e,s = generateKeys()
```

In [3]:
```
def OAEP(pk,m): ##OAEP encrypt
    a = R(m)
    cm = a**pk
    return cm


def OAEPinv(sk,cm):
    b=R(cm)
    dm = b**sk
    return dm
```

In [4]:
```
def generateRandomString(size):
    i = 0
    stream = ""
    while(i<size):
        j = randint(0,1)
        stream = stream + str(j)
        i+=1
    return stream

def generateZeroString(size):
    i = 0
    stream = ""
    while(i<size):
        stream = stream + str(0)
        i+=1
    return stream

xor = lambda x, y: x.__xor__(y)

def concat(i,j):
    return i +j
```

In [5]:
```
class KEM:
    def encrypt(self,pk,x,n):
        return power_mod(x,ZZ(pk),n)

    def decrypt(self,sk,enc,n):
        return power_mod(enc,ZZ(sk),n)

    def encapsulation(self,pk): #enc
        x = randint(1,N-1)
        #enc = self.encrypt(pk,x,N)
        enc = OAEP(pk,x)
        k = hash(x)
        return (k,enc)

    def reveal(self,sk,enc):
        #x = self.decrypt(sk,enc,N)
        x = OAEPinv(sk,enc)
        k = hash(x)
        return k

    def enc(self,pk):
        a = generateRandomString(l)
        zero = generateZeroString(l)
        (k,enc) = self.encapsulation1(pk,concat(a,zero))
        return (k,enc)

    def encapsulation1(self,pk,a0):
        enc = OAEP(pk,int(a0))
        print('encapsulation1- enc: ' + str(enc))
        k = hash(enc)
        print('encapsulation1- k: ' + str(k))
        return (k,enc)
```

In [6]:
```
kem = KEM()
(k,enc) = kem.encapsulation(e)
print('k: ' + str(k))
print('enc: ' + str(enc))

k1 = kem.reveal(s,enc)
print('k1: ' + str(k1))
```

```
k: 139641236788874034
enc: 17325246539289664483127633599705708342190544556721954590158946178977459380595557318566898755198796393251965400478763186374758103696351011098509175831413395939592462936976479673501490891547616282847843105984955773119794673495510260927869637482486562185005902377414048516164956797176559955722020984258488391781412201036446241048260888669496253121314671809571151206445058819532547552731721201207154350126905643339281311486946058371436050304343154030054172081946037157928365833307260325683678932591734425139045437479559890034563698450990841451513626910189184255202916207805778324879586203988210330275680842821215781102194
k1: 139641236788874034
```

In [7]:
```
class PKE:
    def __init__(self):
        self.kem = KEM()

    def encrypt(self,pk,m):
        (k,enc) = self.kem.encapsulation(pk)
        c = xor(m,k)
        return (enc,c)

    def decrypt(self,sk,c):
        (enc,m1) = c
        k = self.kem.reveal(sk,enc)
        m = xor(m1,k)
        return m
```

In [8]:
```
class FOT:
    def __init__(self):
        self.kem = KEM()

    def encrypt(self,pk,m):
        a = generateRandomString(l)
        (enc,k) = self.encrypt1(pk,m,a)
        return (enc,k)

    def encrypt1(self,pk,a,m):
        (enc,k) = kem.encapsulation1(pk,concat(a,hash(m)))
        #print('encrypt1 - enc: ' + str(enc))
        #print('encrypt1 - k: ' + str(k))
        aux1 = concat(str(a),str(m))
        #print('encrypt1 - aux1: ' + str(aux1))
        aux2 = xor(int(aux1),int(k))
        #print('encrypt1 - aux2: ' + str(aux2))
        return (enc,aux2)

    def decrypt(self,sk,c):
        (enc,m1) = c
        k = kem.reveal(sk,enc)
        print('decrypt- k: ' + str(k))
        am = xor(m1,k)
        a = am[0:l]
        m = am[l:]
        if(c == encrypt1(pk,a,m)):
            return m
        else:
            return false
```

In [ ]:
```
pke = PKE()

fot = FOT()
pk = e
sk = s

m = 123
c = pke.encrypt(pk,m)
(enc,t) = c
print('t: ' + str(t))
m1 = pke.decrypt(sk,c)
print('m1: ' + str(m1))

c = fot.encrypt(pk,m)
enc,k = c
print('k: '+ str(k))
c2 = fot.decrypt(sk,c)
print(c2)
```