# Exercise 3

## Question 7

In [33]:
```
a0 = np.array([1, 1, 0, 1, 0, 1])
b0 = np.array([1, 1, 0, 0, 1, 1])
```

In [34]:
```
def rand_key(p):
    key = []
    for i in range(p):
        key.append(random.randint(0,1))
    return key
```

In [35]:
```
def random_vector(s, t):
    global a0, b0
    key = 0
    while(True):
        key = np.array(rand_key(6))
        if(getB(a0, key) == s and getB(b0, key) == t):
            break
    return key
```

In [36]:
```
'''Function to check if the generated vectors fulfil the independen
cy requirements'''
#input two lists a & b, each contains 4 vectors
#Return "True" if any 3 pairs of vectors from (a1,b1),(a2,b2),(a3,b
3),(a4,b4) are linearly independent
def check_dependency(a,b):
    #
    for v1 in range(1,3): #1st vector from 1 to 2
        for v2 in range(v1+1,4):
            for v3 in range(v2+1,5):
                squareMatrix = np.vstack((a[v1], b[v1], a[v2], b[v2
],a[v3], b[v3]))
                determinant = np.linalg.det(squareMatrix)
                if determinant == 0: #if determinant is 0, the vect
ors are not linearly dependent
                    return False

    # check if a0,b0 and any two random selected pairs of vectors a
re linearly independent
    for v1 in range(1,4): #1st vector from 1 to 3
        for v2 in range(v1+1,5):
            squareMatrix = np.vstack((a[0], b[0], a[v1], b[v1], a[v
2], b[v2]))
            determinant = np.linalg.det(squareMatrix)
            if determinant == 0: #if determinant is 0, the vectors
are not linearly dependent
                return False
    return True
```

In [37]:
```
## IGNORE THIS
def generateDependent():
    global a0, b0
    aT = []
    bT = []
    while True:
        aT = [0]
        bT = [0]
        for i in range(4):
            aT.append(rand_key(6))
            bT.append(rand_key(6))
        print(aT)
        print(bT)
        if(not check_dependency(aT, bT)):
            continue
        aT = [a0.tolist()] + aT
        bT = [b0.tolist()] + bT

        #Generate combination of 4
        t2 = []
        for i in range(2**4):
            w = [0] * 4
            t3 = bin(i)[2:]
            while(len(t3)<4):
                t3 = "0" + t3
            for j in range(4):
                if(t3[j] == "1"):
                    w[j] = 1
                else:
                    w[j] = 0
            t2.append(w)

        t3 = []
        for i in t2:
            if(sum(i) == 3):
                t3.append([1]+i)
        print(t3)
        for i in range(len(t3)):
            aT1 = []
            bT1 = []
            for j in range(len(t3[i])):
                if(t3[i][j] == 1):
                    aT1.append(aT[j])
                    bT1.append(bT[j])
            print(aT1)
            print(bT1)
            if(not check_dependency(aT1, bT1)):
                cond = False
                break

        if(not cond):
            continue
        return aT, bT
```

In [38]:
```
def generateDependent1():
    global a0, b0
    aT = []
    bT = []
    while True:
        aT = [a0]
        bT = [b0]
        for i in range(4):
            aT.append(rand_key(6))
            bT.append(rand_key(6))
        if(not check_dependency(aT, bT)):
            continue
        return aT, bT
```

In [39]:
```
at, bt = generateDependent1()
```

In [40]:
```
a1 = at[1]
b1 = bt[1]

a2 = at[2]
b2 = bt[2]

a3 = at[3]
b3 = bt[3]

a4 = at[4]
b4 = bt[4]
```

In [41]:
```
print(at)
```

```
[array([1, 1, 0, 1, 0, 1]), [1, 0, 0, 1, 1, 1], [0, 0, 1, 0, 1, 1]
, [0, 1, 0, 0, 1, 0], [0, 0, 0, 1, 1, 1]]
```

In [42]:
```
'''Function to converting String to binary array'''
def str2bits(s):
    res = ''.join(format(ord(i), 'b') for i in s)
    bitsArray = []
    for i in res:
        bitsArray.append(int(i))
    return bitsArray
```

In [43]:
```
passBit = np.array(str2bits("Potter"))
print("Password Bits:", passBit)
```

```
Password Bits: [1 0 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 1 1 1 0
1 0 0 1 1 0 0 1 0 1 1 1
 1 0 0 1 0]
```

In [44]:
```
n = passBit.shape[0]//2
passBT = passBit.reshape((2,n))
```

In [45]:
```
uN = []
for i in range(n):
    uN.append(random_vector(passBT[0][i], passBT[1][i]))
```

In [46]:
```
betaI = []
gammaI = []

for i in range(n):
    betaTemp = []

    for j in range(1,len(at)):
        betaTemp.append(getB(uN[i], at[j]))

    gammaTemp = []

    for j in range(1, len(bt)):
        gammaTemp.append(getB(uN[i], bt[j]))

    betaI.append(betaTemp)
    gammaI.append(gammaTemp)

betaI = np.array(betaI)
gammaI = np.array(gammaI)
```

In [47]:
```
def findSet(comb):
    # comb is vector which have 1 at pos i to indicate taking vecto
r i
    global at, bt
    global betaI, gammaI, n

    X = []

    Y =[]

    for i in range(len(comb)):
        if(comb[i] == 1):
            X.append(at[i+1])
            X.append(bt[i+1])

            Y.append(betaI[:,i])
            Y.append(gammaI[:,i])

    X = np.array(X)
    Y = np.array(Y)

    uN = []
    for i in range(n):
        w = [0 for j in range(6)]
        for j in range(2**6):
            #Get All possible weights
            t1 = bin(j)[2:]
            while(len(t1)<6):
                t1 = "0" + t1
            for k in range(6):
                if(t1[k] == "1"):
                    w[k] = 1
                else:
                    w[k] = 0
            if(np.all(np.equal(np.mod(np.matmul(X,w),2), Y[:,i]))):
                break
        else:
            print("Not found")
            return -1

        uN.append(w)

    return uN
```

In [48]:
```
def bits2str(b):
    NumOfChar = len(b)//7
    string = ''
    for i in range(NumOfChar):
        bitsChar = ''.join(str(j) for j in b[7*i:7*i+7]) # 7 digits
represents 1 char
        decimalChar = int(bitsChar,2) #convert binary to decimal
        string = string + chr(decimalChar) #convert decimal to stri
ng
    return string
```

In [49]:
```
combs = []
for i in range(2**4):
    w = [0] * 4
    t3 = bin(i)[2:]
    while(len(t3)<4):
        t3 = "0" + t3
    for j in range(4):
        if(t3[j] == "1"):
            w[j] = 1
        else:
            w[j] = 0
    if(sum(w) == 3):
        combs.append(w)
```

In [50]:
```
combs
```

Out[50]:
```
[[0, 1, 1, 1], [1, 0, 1, 1], [1, 1, 0, 1], [1, 1, 1, 0]]
```

In [51]:
```
len(combs)
```

Out[51]:
```
4
```

In [52]:
```
for i in range(len(combs)):
    u = findSet(combs[i])

    s = []
    t = []
    for j in range(len(u)):
        s.append(np.dot(a0,u[j])%2)
        t.append(np.dot(b0,u[j])%2)

    passBitString = s + t
    print(bits2str(passBitString))
```

```
Potter
Potter
Potter
Potter
```

Single Case

In [53]:
```
u1 = findSet([1,1,1,0])
```

In [54]:
```
s1 = []
t1 = []
for i in range(len(u1)):
    s1.append(np.dot(a0,u1[i])%2)
    t1.append(np.dot(b0,u1[i])%2)

passBitString = s1 + t1
```

In [55]:
```
bits2str(passBitString)
```

Out[55]:
```
'Potter'
```