

Infonet Security HW3

Harris Ransom

10/30/2024

```
In [110... import numpy as np
import random
import sympy
import math
```

Problem 1

```
In [111... def exponentiation(m,e,n):
    return((m**e)%n)

print(exponentiation(2000,2020,2023))
```

72

```
In [112... def extendedGCD(a, b):
    x0 = 1
    x1 = 0
    y0 = 0
    y1 = 1

    while b != 0:
        q = a // b
        a, b = b, a % b
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1

    return a, x0

def inverseFinder(a,n):
    gcd, x = extendedGCD(a, n)
    if gcd == 1:
        return x % n

print(inverseFinder(2000,2023))
```

1935

Problem 2

In [113...

```
def rsaKey(e):
    while True:
        while True:
            p = random.getrandbits(512)
            if sympy.isprime(p):
                break

        while True:
            q = random.getrandbits(512)
            if sympy.isprime(q) is True:
                if q != p:
                    break

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

        if math.gcd(e, phi) == 1 and e < phi:
            publicKey = [e,n]
            privateKey = [inverseFinder(e,phi),n]
            return(publicKey,privateKey)

publicKeys = []
privateKeys = []
for i in range(10):
    publicKey, privateKey = rsaKey(3)
    publicKeys.append(publicKey)
    privateKeys.append(privateKey)

for i in range(10):
    print(f"Public Key {i}: {publicKeys[i]}")
    print(f"Private Key {i}: {privateKeys[i]}")
    print(f"\n")
```

Public Key 0: [3, 10544538975581158699755595701918428380780111749278593142720021899261803168746385050866566290742799756869748156544248846774331570932066396391097035185133641566000921878688388238134373017065599696977762403949245284672913055580438280697939538365732817503327129168374147245167930214371058496970682525392062979541]

Private Key 0: [7029692650387439133170397134612285587186741166185728761813347932841202112497590033911044193828533171246498771029499231182887713954710930927398023456755756487360249396132149739770395274649482580071053949856106897935537877424157981472979663803373401390810430608433901948775650451147976266574441142707046172715, 10544538975581158699755595701918428380780111749278593142720021899261803168746385050866566290742799756869748156544248846774331570932066396391097035185133641566000921878688388238134373017065599696977762403949245284672913055580438280697939538365732817503327129168374147245167930214371058496970682525392062979541]

Public Key 1: [3, 8693260286873712430860605438884516552423247658208050764428954045096250279069522923124081197379272782478987235749622313285103871857344491693574768471428927101608252363819886749952282765868470759193415882819650258451885016181218862488539732089447544237872086894369060241836616463612038517567827907943507774933]

Private Key 1: [5795506857915808287240403625923011034948831772138700509619302696730833519379681948749387464919515188319324823833081542190069247904896327795716512314285947015295403148766534333800482726344762412926428296684598483768447017748910767902777591171992754110720630988218130619192629777050858681752715607396186495979, 8693260286873712430860605438884516552423247658208050764428954045096250279069522923124081197379272782478987235749622313285103871857344491693574768471428927101608252363819886749952282765868470759193415882819650258451885016181218862488539732089447544237872086894369060241836616463612038517567827907943507774933]

Public Key 2: [3, 33262310315681573756505093157489625898744525171193671270253363442583535270500705165374853785529125145856488453674204904399317308809340536617635996921043087645619766209430029180582103612300105280335585342482640752463361954846047791721807299578395115755080594921317948118730550570516327501434043119696335457117]

Private Key 2: [22174873543787715837670062104993083932496350114129114180168908961722356847000470110249902523686083430570992302449469936266211539206227024411757331280695382433644390742577398868663427193764119638594148443713571667290294239592440892861767971358445902950009533601088388265471261237934553877809555836270922065491, 33262310315681573756505093157489625898744525171193671270253363442583535270500705165374853785529125145856488453674204904399317308809340536617635996921043087645619766209430029180582103612300105280335585342482640752463361954846047791721807299578395115755080594921317948118730550570516327501434043119696335457117]

Public Key 3: [3, 7845451752226646737762227588665274897895686323754560458419046787580253599690003236116121508007501654038878974222537799215744230743993346948100214622479981032710943544976474297923277862022671300992207015303286838292892262216775404033371988254784019056868349891447575857867507871034764043831155093791847555031]

Private Key 3: [523030116815109782517481839244351659859712421583637363894603119172016906646000215741074767200500110269258598281502519947716282049599556463206680974831998221117251201042502581198337847199886399887532027555852931161103058714730662245008415974786922221452069982123383881840101074914036486589080848112592242043, 7845451752226646737762227588665274897895686323754560458419046787580253599690003236116121508007501654038878974222537799215744230743993346948100214622479981032710943544976474297923277862022671300992207015303286838292892262216775404033371988254784019056868349891447575857867507871034764043831155093791847555031]

Public Key 4: [3, 6826010860457115646987139980991978484981453571601266397780278919827062490462947327534459032828402126536733114835368647324178755218466878922970970189989030233746162117556720080326790495808604934947090664247581555073739626369042471324701331662086719562143776210421784178355316649419439386110547791571794725541]
Private Key 4: [4550673906971410431324759987327985656654302381067510931853519279884708326975298218356306021885601417691155409890245764882785836812311252615313980126659346267254920374874884176066484878608175345627414160014912790069539027927384666983627417665048886607133563835942608414658642027727203946571647913617022807667, 6826010860457115646987139980991978484981453571601266397780278919827062490462947327534459032828402126536733114835368647324178755218466878922970970189989030233746162117556720080326790495808604934947090664247581555073739626369042471324701331662086719562143776210421784178355316649419439386110547791571794725541]

Public Key 5: [3, 84640251257869169554796988148892793786182246753490456787402735519488590969519154125980272972553141787475333020807744394045472578169309285101533131821769573400266958989596067522318847640817774561650152422614428051062941734408885488147942883692760613698625681973505989798830238392613246205042459972803207543027]
Private Key 5: [56426834171912779703197992099261862524121497835660304524935157012992393979679436083986848648368761191650222013871829596030315052112872856734355421214513036497044358995869603764430640085797523178295547642103582355104445316583102815903639881556661285308974149554230694954573088309033683979673115767361388578827, 84640251257869169554796988148892793786182246753490456787402735519488590969519154125980272972553141787475333020807744394045472578169309285101533131821769573400266958989596067522318847640817774561650152422614428051062941734408885488147942883692760613698625681973505989798830238392613246205042459972803207543027]

Public Key 6: [3, 44090541414756306758579295441127286597515513557710749159289008084609721820720329094576596528668217007254436254754310549176367439888906604579370516764079659953711820226459878864565275276542707255676649759120789505014005073743796631939043316532375951524764394404782010996088138967957597835868740911951109981437]
Private Key 6: [29393694276504204505719530294084857731677009038473832772859338723073147880480219396384397685778811338169624169836207032784244959925937736386247011176053097102167527508747355998875919123797422280042830218073394258621899847479144251747281394850026639590538982297773416732733884371018361242797835570313003495851, 44090541414756306758579295441127286597515513557710749159289008084609721820720329094576596528668217007254436254754310549176367439888906604579370516764079659953711820226459878864565275276542707255676649759120789505014005073743796631939043316532375951524764394404782010996088138967957597835868740911951109981437]

Public Key 7: [3, 66413229193833288023261584216703529730094752680126328457826905632918742848764299709319160405275124089108863121470050911714865377851971403581889706389925271021392427973032318946624204679328497568629316205437414446849411131491149790275304487471750971306085500001201582774988693746101715242089185761969746469233]
Private Key 7: [4427548612922219201550772281113568648672983512008421897188460375527916189917619980621277360351674939273924208098003394114324358523464760238792647092661683569093760596558955547613011133593424962581571332915094370070284291413278175026667918403738733712975303062071304057473769870550472626207080005422281297227, 66413229193833288023261584216703529730094752680126328457826905632918742848764299709319160405275124089108863121470050911714865377851971403581889706389925271021392427973032318946624204679328497568629316205437414446849411131491149790275304487471750971306085500001201582774988693746101715242089185761969746469233]

Public Key 8: [3, 32053038220286200825989921493706345505720656757573324009270638346677854141355052040597334419340961581946628224819585729261067665165553974347637239520724308932686651419815997237739340448329881111164299467027568322613087156789175549645620154179077061871435639726749765453151360629915744785512655958117688816497]

Private Key 8: [21368692146857467217326614329137563670480437838382216006180425564451902760903368027064889612893974387964418816546390486174045110110369316231758159680482863213584830016686208701344825843995588986235397004216635920485715891058967172592913443819497552313020159216200642831086654945644318908019390427234205706827, 32053038220286200825989921493706345505720656757573324009270638346677854141355052040597334419340961581946628224819585729261067665165553974347637239520724308932686651419815997237739340448329881111164299467027568322613087156789175549645620154179077061871435639726749765453151360629915744785512655958117688816497]

Public Key 9: [3, 12949762013422672416671821417433514816017306332456645412161485536080305599036515370936864663469922964487501558531588859369956014681574924753409616564812396821631260087140695470433223929552942381227156204185258963348897728177071849357432591671071525784708379350288831419574902174988921123095658918477834369587]

Private Key 9: [86331746756151149444478809449556765440115375549710969414409903573868703993576769139579097756466153096583343723543925729133040097877166165022730777098749296940746681313280414571354175648061041440686223361409990762629578017617042040677320461847548089376292375077489154915043159542426271923540608546446208038891, 12949762013422672416671821417433514816017306332456645412161485536080305599036515370936864663469922964487501558531588859369956014681574924753409616564812396821631260087140695470433223929552942381227156204185258963348897728177071849357432591671071525784708379350288831419574902174988921123095658918477834369587]

In [114...

```
def rsaEncrypt(msg, pubKey):
    e = int(pubKey[0])
    n = int(pubKey[1])
    msg = int.from_bytes(msg.encode('utf-8'), 'big')

    cipherText = pow(msg, e, n)
    return(cipherText)

def rsaDecrypt(msg, privateKey):
    d = int(privateKey[0])
    n = int(privateKey[1])

    cipherText = pow(msg, d, n)

    cipherText = cipherText.to_bytes((cipherText.bit_length() + 7) // 8, 'big')
    recoveredPlaintext = cipherText.decode('utf-8')

    return(recoveredPlaintext)

pubKey, privateKey = rsaKey(65537)
plainText = "123456789"
print(f"Plain Text Message: {plainText}")
cipher = rsaEncrypt(plainText, pubKey)
recoveredText = rsaDecrypt(cipher, privateKey)
print(f"Recovered Message: {recoveredText}")
```

Plain Text Message: 123456789

Recovered Message: 123456789

```
In [115... plainText = "hello world!"
print(f"Plain Text Message: {plainText}")
cipher = rsaEncrypt(plainText, pubKey)
recoveredText = rsaDecrypt(cipher, privateKey)
print(f"Recovered Message: {recoveredText}")
```

Plain Text Message: hello world!

Recovered Message: hello world!

Problem 3

Problem 3a

```
In [116... def primitiveRoot(a,n):
    result = []
    for i in range(n-1):
        result.append((a**(i+1))%n)
    if len(result) != len(set(result)):
        print(f"{a} is not a primitive root mod {n}")
        print(result)
    if len(result) == len(set(result)):
        print(f"{a} is a primitive root mod {n}")
        print(result)
    return(result)

_ = primitiveRoot(2,13)
_ = primitiveRoot(3,13)
```

2 is a primitive root mod 13

[2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1]

3 is not a primitive root mod 13

[3, 9, 1, 3, 9, 1, 3, 9, 1, 3, 9, 1]

Problem 3b

$$2^x \equiv 12 \pmod{13}$$

$$g = 2, h = 12, p = 13, N = 13 - 1 = 12 = 2^2 - 1$$

q	e	$\frac{p-1}{q^e}$	
		g	h
2	2	8	1728
3	1	16	20736

$$x \equiv (x_0 + 2x_1) \pmod{2^2}$$

$$(8^{2^1})^{x_0}(\text{mod}13) = (1728^{2^1})(\text{mod}13) \Rightarrow 12^{x_0} = 1 \Rightarrow x_0 = 0$$

$$(8^{2^1})^{x_1}(\text{mod}13) = (1728 * 8^{-x_0})^{2^0}(\text{mod}13) \Rightarrow 12^{x_1} = 12 \Rightarrow x_1 = 1$$

$$x \equiv 0 + 2(1)(\text{mod}2^2) \Rightarrow x \equiv 2(\text{mod}2^2)$$

$$x \equiv x_0(\text{mod}3^1)$$

$$16^{x_0}(\text{mod}13) = 20736(\text{mod}13) \Rightarrow 16^{x_0} = 1 \Rightarrow x_0 = 0$$

$$x \equiv 0(\text{mod}3)$$

We can use Chinese Remainder Theorem to solve for x

$$3 * 2(\text{mod}4) \Rightarrow 2(\text{mod}4)$$

$$4 * 0(\text{mod}3) \Rightarrow 0(\text{mod}3)$$

$$x = 3 * 2 + 0 = 6$$

$$x = 6$$

Problem 3c

We can find the following using the result from 3b

$$2^x \equiv 3(\text{mod}13)$$

$$2^6 \equiv 12(\text{mod}13)$$

$$\frac{2^6}{4} \equiv \frac{12}{4}(\text{mod}13)$$

$$2^4 \equiv 3(\text{mod}13)$$

$$x = 4$$

In [117...

```
discreteLogs = primitiveRoot(2,13)
solutions = []
for i in range(len(discreteLogs)):
    if discreteLogs[i] == 12:
        solutions.append(i+1)
print(f"Solutions to 2B:")
print(solutions)

solutions = []
for i in range(len(discreteLogs)):
    if discreteLogs[i] == 3:
        solutions.append(i+1)
print(f"Solutions to 2C:")
print(solutions)
```

2 is a primitive root mod 13
 [2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1]
 Solutions to 2B:
 [6]
 Solutions to 2C:
 [4]