krypto4

November 15, 2023

1 Cryptography handout 4

1.1 Problem 1

[85]:
```
import numpy as np

n = 275621053.0

a = int(np.ceil(np.sqrt(n)))
b = np.sqrt(a*a - n)

while True:
    if(b % np.floor(b) == 0):
        print(f"p = {a-b} : q = {a+b}")
        break
    elif a > b:
        a = a + 1
        b = np.sqrt(a*a - n)
    else:
        print("n is prime")
        break
```

```
p = 16193.0 : q = 17021.0
```

1.2 Problem 2

1.2.1 A)

p = 1283

d = 3

key = (n,d) with n = pq

potential products n:

q = 1307, n = 1676881

q = 1879, n = 2410757

q = 2003, n = 2569849

q = 2027, n = 2600641

First of all its generally good to have q and p to not be close together. This makes factorization with methods such as fermats factorization harder. We can use fermats factorization to analyze the possible q values in regards to steps needed to compute the factorization:

We reuse the code from task 1:

1. For the first q-value (1307), we only need one step of fermats factorization. This makes brute forcing the the encryption quite simple.
2. For the second value 29 steps is needed, so relatively heavier.
3. Third value requries 40 steps.
4. Fourth value requires 43 steps, making it the best choice in regards to fermat attacking and in general the best option for q.

1.2.2 B)

For RSA we have de 1 mod (p-1)(q-1)

we call (p-1)(q-1) = n

that gives us de 1 mod n

de - 1 = kn

de + (-kn) = 1

We can use extended euclidean algorithm to solve for e.

[86]:
```
def extended_gcd(a, b):
    s_0, s_1 = 1, 0
    t_0, t_1 = 0, 1
    r_0, r_1 = a, b

    while r_1 != 0:
        q = r_0 // r_1
        r_0, r_1 = r_1, r_0 - q * r_1
        s_0, s_1 = s_1, s_0 - q * s_1
        t_0, t_1 = t_1, t_0 - q * t_1

    return s_0, t_0

a = 3
p = 1283
q = 2027
n = -(p - 1) * (q - 1)

e, k = extended_gcd(a, n)

print(f"e = {e}")
print(f"k = {k}")
```

```
e = 865777
k = 1
```

From the python code we can see that the e value is 865777

1.2.3 C)

[87]:
```
a = 111
n = p*q

def mod_exp(b, e, mod):
    if e == 0:
        return 1

    if e == 1:
        return b % mod
    t = mod_exp(b, e // 2, mod)
    t = (t * t) % mod

    # if exponent is even
    if e % 2 == 0:
        return t

    # if exponent is odd
    else:
        return (b % mod * t) % mod

print(f"The encrypted message is: {mod_exp(a, e, n)} : {bin(mod_exp(a, e, n))}")
```

```
The encrypted message is: 326052 : 0b1001111100110100100
```

1.3 Problem 3

1.3.1 A)

Let n = 1829 and B = 5. Find a prime factor of n by using Pollard (p − 1) attack

[88]:
```
import math

def gcd(a, b):
    if(b == 0):
        return abs(a)
    else:
        return gcd(b, a % b)

def pollard(n, B):
    a = 2
    A = mod_exp(a, math.factorial(B), n)
    F = gcd(A-1, n)
    if F > 1:
        print(f"prime factor: {F}")
```

```
    else:
        print("No prime factor found")

n = 1829
B = 5
pollard(n, B)
```

```
prime factor: 31
```

1.3.2 B)

n = 18779

If we factorize the prime number p-1 we get a set of Q values. Choosing B as the last Q value in the sequence makes B! divisible by p-1. So the smallest B value depends on the possible prime factorization of p or q. Its impossible to predict the exact minimum B value without factorizing p or q. But the theoretical minimum for B that will produce a successful is the smallest of the last Q values in the sequence of prime factors in either p or q.

We use this knowledge to see if we can find the lower B value:

[89]:
```
n = 18779.0

a = int(np.ceil(np.sqrt(n)))
b = np.sqrt(a*a - n)

while True:
    if(b % np.floor(b) == 0):
        print(f"p = {a-b} : q = {a+b}")
        break
    elif a > b:
        a = a + 1
        b = np.sqrt(a*a - n)
    else:
        print("n is prime")
        break

p,q = a-b,a+b
```

```
p = 89.0 : q = 211.0
```

[90]:
```
def primeFactors(n):
    while n % 2 == 0:
        print(2)
        n = n // 2
    for i in range(3,int(math.sqrt(n))+1,2):
        while n % i== 0:
            print(i)
            n = n // i
```

```
    if n > 2:
        print(n)

print("Prime factors of p-1")
primeFactors(p-1)
print("Prime factors of q-1")
primeFactors(q-1)
```

```
Prime factors of p-1
2
2
2
11.0
Prime factors of q-1
2
3
5
7
```

Smallest B is given by the smallest of the final Q values, in this case 7. We can test and see if the pollard attack works:

[91]:
```
pollard(n, 7)
```

```
prime factor: 211.0
```

1.4 Problem 4

1.4.1 A)

We will prove following property of RSA:

$$(e_k(x1) * e_k(x2)) mod n = (x1 * x2) mod n$$

The definition of encryption with RSA goes as follows:

$$e_k(x) \equiv x^k mod n$$

Following modular arithmetic we can write the congruence expression as:

$$e_k(x) = kn + x$$

We use the this in the expression from the problem.

$$(e_k(x1) * e_k(x2)) mod n$$

$$((kn + x1)(xn + x2)) mod n$$

$$((kn)^2 + kn * x2 + kn * x1 + x1 * x2) mod n$$

The sum of all the expression with kn will be a multiple of n, meaning the modulo operation will result in 0. Leaving us with the expression:

$$(x1 * x2) mod n$$

1.4.2 B)

Show how RSA is vulnerable to chosen cipher text attack: For ciphertext y, then Eva can choose some r 1 mod n, and construct y = y · r^e. If she then knows the decryption x = dK(y), show how she can calculate x = dK(y). (Hint: She can also calculate r^−1 mod n)

Given a public key e, and a private key d. Eva knows a r which is not congruent to n. She knows a ciphertext y, and calculates a new ciphertext

$$y' = y * r^e$$

$$y' \equiv y * r^e modn$$

We can then express the decrypted x like this:

$$x' \equiv (y')^d modn$$

From definition of y' we can reexpress this as:

$$x' \equiv (y * r^e)^d modn$$

We know from following definition:

$$de \equiv 1(mod(p-1)(q-1)$$

That we have the following for r^ed

$$r^{ed} \equiv rmodn$$

We therefore rexpress r^ed as r giving

$$x' \equiv (y^d * r)modn$$

We know that Eva can calculate the inverse of r, and can therefore calculate the cleartext for x with:

$$x \equiv x' * (r^{-1}(modn))(modn)$$

1.5 Problem 5

1.5.1 A)

[92]:
```
for i in range(100):
    a = 3**i
    print(f"{a}")


for i in range(100):
    a = 5**i
    print(f"{a}")
```

```
1
3
9
27
```

81
243
729
2187
6561
19683
59049
177147
531441
1594323
4782969
14348907
43046721
129140163
387420489
1162261467
3486784401
10460353203
31381059609
94143178827
282429536481
847288609443
2541865828329
7625597484987
22876792454961
68630377364883
205891132094649
617673396283947
1853020188851841
5559060566555523
16677181699666569
50031545098999707
150094635296999121
450283905890997363
1350851717672992089
4052555153018976267
12157665459056928801
36472996377170786403
109418989131512359209
328256967394537077627
984770902183611232881
2954312706550833698643
8862938119652501095929
26588814358957503287787
79766443076872509863361
239299329230617529590083
717897987691852588770249
2153693963075557766310747

6461081889226673298932241
19383245667680019896796723
58149737003040059690390169
174449211009120179071170507
523347633027360537213511521
1570042899082081611640534563
4710128697246244834921603689
14130386091738734504764811067
42391158275216203514294433201
127173474825648610542883299603
381520424476945831628649898809
1144561273430837494885949696427
3433683820292512484657849089281
10301051460877537453973547267843
30903154382632612361920641803529
92709463147897837085761925410587
278128389443693511257285776231761
834385168331080533771857328695283
2503155504993241601315571986085849
7509466514979724803946715958257547
22528399544939174411840147874772641
67585198634817523235520443624317923
202755595904452569706561330872953769
608266787713357709119683992618861307
1824800363140073127359051977856583921
5474401089420219382077155933569751763
16423203268260658146231467800709255289
49269609804781974438694403402127765867
147808829414345923316083210206383297601
443426488243037769948249630619149892803
1330279464729113309844748891857449678409
3990838394187339929534246675572349035227
11972515182562019788602740026717047105681
35917545547686059365808220080151141317043
107752636643058178097424660240453423951129
323257909929174534292273980721360271853387
969773729787523602876821942164080815560161
2909321189362570808630465826492242446680483
8727963568087712425891397479476727340041449
26183890704263137277674192438430182020124347
78551672112789411833022577315290546060373041
235655016338368235499067731945871638181119123
706965049015104706497203195837614914543357369
2120895147045314119491609587512844743630072107
6362685441135942358474828762538534230890216321
19088056323407827075424486287615602692670648963
57264168970223481226273458862846808078011946889
171792506910670443678820376588540424234035840667

1
5
25
125
625
3125
15625
78125
390625
1953125
9765625
48828125
244140625
1220703125
6103515625
30517578125
152587890625
762939453125
3814697265625
19073486328125
95367431640625
476837158203125
2384185791015625
11920928955078125
59604644775390625
298023223876953125
1490116119384765625
7450580596923828125
37252902984619140625
186264514923095703125
931322574615478515625
4656612873077392578125
23283064365386962890625
116415321826934814453125
582076609134674072265625
2910383045673370361328125
14551915228366851806640625
72759576141834259033203125
363797880709171295166015625
1818989403545856475830078125
9094947017729282379150390625
45474735088646411895751953125
227373675443232059478759765625
1136868377216160297393798828125
5684341886080801486968994140625
28421709430404007434844970703125
142108547152020037174224853515625
710542735760100185871124267578125

35527136788005009293556213378906 25
17763568394002504646778106689453125
888178419700125232338905334472656 25
4440892098500626161694526672363281 25
22204460492503130808472633361816406 25
111022302462515654042363166809082031 25
555111512312578270211815834045410156 25
2775557561562891351059079170227050781 25
13877787807814456755295395851135253906 25
69388939039072283776476979255676269531 25
346944695195361418882384896278381347656 25
1734723475976807094411924481391906738281 25
8673617379884035472059622406959533691406 25
43368086899420177360298112034797668457031 25
216840434497100886801490560173988342285156 25
1084202172485504434007452800869941711425781 25
5421010862427522170037264004349708557128906 25
27105054312137610850186320021748542785644531 25
135525271560688054250931600108742713928222656 25
677626357803440271254658000543713569641113281 25
3388131789017201356273290002718567848205566406 25
16940658945086006781366450013592839241027832031 25
84703294725430033906832250067964196205139160156 25
423516473627150169534161250339820981025695800781 25
2117582368135750847670806251699104905128479003906 25
10587911840678754238354031258495524525642395019531 25
52939559203393771191770156292477622628211975097656 25
264697796016968855958850781462388113141059875488281 25
1323488980084844279794253907311940565705299377441406 25
6617444900424221398971269536559702828526496887207031 25
33087224502121106994856347682798514142632484436035156 25
165436122510605534974281738413992570713162422180175781 25
827180612553027674871408692069962853565812110900878906 25
4135903062765138374357043460349814267829060554504394531 25
20679515313825691871785217301749071339145302772521972656 25
103397576569128459358926086508745356695726513862609863281 25
516987882845642296794630432543726783478632569313049316406 25
2584939414228211483973152162718633917393162846565246582031 25
12924697071141057419865760813593169586965814232826232910156 25
64623485355705287099328804067965847934829071164131164550781 25
323117426778526435496644020339829239674145355820655822753906 25
1615587133892632177483220101699146198370726779103279113769531 25
8077935669463160887416100508495730991853633895516395568847656 25
40389678347315804437080502542478654959268169477581977844238281 25
201948391736579022185402512712393274796340847387909889221191406 25
1009741958682895110927012563561966373981704236939549446105957031 25
5048709793414475554635062817809831869908521184697747230529785156 25
25243548967072377773175314089049159349542605923488736152648925781 25

```
12621774483536188886587657044524579674771302961744368076324462890625
63108872417680944432938285222622898373856514808721840381622314453125
315544362088404722164691426113114491869282574043609201908111572265625
1577721810442023610823457130565572459346412870218046009540557861328125
```

The most notable difference is in the sizes of the sequences. The power of 5 sequence grows faster and maxes out with a lot more size.

Find the common key when the prime is 101, n = 3, Alice has secret 33 and bob 65. key = n^(a*b)

```python
[93]: n = 3
p = 101
a = 33
b = 65
key = (n**(33*65)) % p
print(key)
```

```
32
```

Their shared key is 32

```python
[ ]:
```

```python
[ ]:
```