Assignment -Basic Crypto

Problem 1 Mind your Ps and Qs

We download the values and got
Decrypt my super sick RSA:
c:623247839491341191594088165133349125343435173008801376916627808954099927602
62021
n:128067841582221405786452479845329781918191062157394547754475817105596824511
6423923
e: 65537
After that i used integer factorization calculator which give me the prime factor of n and to get n's factor it took 9 min and got the value and those are p and q

```
p=189910798652748353534451711394853132833 1
q=674357869540600933870145899564746495319033
```

I can specify the p and q and got the private key using the code and got the plain text of the cipher text
After that i convert into hexadecimal after that i took hex value and convert into byte array and again convert into string and got the flag

```
def egcd(a,b):
    if a==0:
        return (b,0,1)
    else:
        g,y,x=egcd(b%a,a)
        return (g,x-(b//a)*y,y)
def modinv(a,m):
    g,x,y=egcd(a,m)
    if g!=1:
        raise Exception('modular inverse does not exist')
    else:
        return x%m

p=189910798652748353534451711394853132833 1
q=674357869540600933870145899564746495319033
n=p*q
phi=(p-1)*(q-1)

e=65537
d=modinv(e,phi)
c=623247839491341191594088165133349125343435173008801376916627808954099927
60262021
```

```
plain=pow(c,d,n)
print(plain)
print(hex(plain))
print(bytearray.fromhex(hex(plain)[2:]).decode())
```

My flag is **picoCTF{sma11_N_n0_g0od_05012767}**

Problem 2-Easy Peasy

When we do `nc mercury.picoctf.net 64260`
It give me encrypted flag
51466d4e5f575538195551416e4f5300413f1b5008684d5504384157046e4959 and ask what data want to encrypt, and when i open otp.py i see there is key length is 50000 and there is loop means if i enter more than 50000 then wrap around, means again same value and when i do the len(encrypted flag) it give 64 hex digit means 32 characters so to get key what i did i run
Python3 -c “print (‘A’*49968) ; print(‘a’*32)” | nc mercury.picoctf.net 64260
After this got the key which is
03463d190702003d195004133d190356433d1902503d1950563d1900513d1959
What i did i added 32 byte plain text like pa=’A’*32
After this ‘{:x}’.format(flag^key^pa) this give me hexadecimal is
336131363934346461643433323731376363633339343564336439363432316 1
And now convert this to ascii and got flag
Flag is –**picoCTF{3a16944dad432717ccc3945d3d96421a}**

Problem 3 - New Caesar
Here i write the code just the reverse of given code to get the flag

```
import string

LOWERCASE_OFFSET = ord("a")
ALPHABET = string.ascii_lowercase[:16]

cipher_text =
"mlnklfnknljflfmhjimkmhjhmljhjomhmmjkjpmmjmjkjpjojgjmjpjojojnjojmmkmlmijimhjmmj"

def b16_decode(solve):
    dec = ""
    for idx in range(0, len(solve), 2):
```

```
        # Get the current and next letters so we can create the one letter that
        # was split in half by `b16_encode`.
        c1 = solve[idx]
        c2 = solve[idx + 1]
        # Determine the location of these letters in the alphabet. The coresponds to
        # the code inside the square brackets `ALPHABET[int(binary[:4], 2)]` from
        # `b16_encode`.
        c1 = ALPHABET.index(c1)
        c2 = ALPHABET.index(c2)
        # Convert the numbers to binary
        binary1 = "{0:04b}".format(c1)
        binary2 = "{0:04b}".format(c2)
        # Add the two binary strings together to recover the encoded character
        binary = int(binary1 + binary2, 2)

        dec += chr(binary)
    return dec

def unshift(c, k):
    # Offset the letters in the opposite direction.
    t1 = ord(c) + LOWERCASE_OFFSET
    t2 = ord(k) + LOWERCASE_OFFSET
    # Instead of moving forward through the alphabet, move backwards.
    return ALPHABET[(t1 - t2) % len(ALPHABET)]

def is_ascii(s):
    return len(s) == len(s.encode())

# Try every possible offset to see what key produces the flag.
for letter in ALPHABET:
    # `letter` is the key
    dec = ""
    # Apply the opposite of the `shift` function on each letter.
    for i, c in enumerate(cipher_text):
        dec += unshift(c, letter)
    # Reverse the `b16_encode` function.
    dec = b16_decode(dec)
    # Only show potential flags that are ascii.
    if is_ascii(dec) and " " not in dec:
```

```
    print("Flag: picoCTF{%s}" % dec)
```

My flag is **picoCTF{et_tu?_a2da1e18af49f649806988786deb2a6c}**

Problem 4 Dachshund Attacks

When i connect to server i got

e=69976593353808213280651886883869388292409943750165282934157392
18336726066235486817740322976584346089638904090600037858917582 43
36945911851479580887086413883636727145469164114395805476827633484
59654347676658200132696584588793977956000598690504452878761638 14
53521174622991675550784306238914546319014786996253031

n=10258227637718601809686725352528860093431613057951465642278583
88816973216705981636503852775429499442650278262284235322377799 84
22227679243563744815836918715160122394286682981024635253078179 20
20075459098791948474755746607070615856025775507477518863069191 82
61891453275240266541614059294261278293684440378420892 19

c=53378813555780450062408881600891375443311648755268537514666147
87284909821699097865071011655148387984218336232672705649826883 59
94796640176165144738944919894572734516991079765866878591392813 03
33788131338024666846386954190011321756160596956300659811678778058
19333334304768133427707336102484646820456275061309 07

This is actually wiener attack so i look for this attack and got the script

```
from typing import Tuple, Iterator, Iterable, Optional


def isqrt(n: int) -> int:
    """
    ref: https://en.wikipedia.org/wiki/Integer_square_root

    >>> isqrt(289)
    17
    >>> isqrt(2)
    1
    >>> isqrt(1000000 ** 2)
    1000000
    """
    if n == 0:
```

```
        return 0

    # ref:
https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Rough_estimation
    x = 2 ** ((n.bit_length() + 1) // 2)
    while True:
        y = (x + n // x) // 2
        if y >= x:
            return x
        x = y


def is_perfect_square(n: int) -> bool:
    """
    ref: https://hnw.hatenablog.com/entry/20140503

    >>> is_perfect_square(100)
    True

    >>> is_perfect_square(2000000000000000000000000000 ** 2)
    True

    >>> is_perfect_square(2000000000000000000000000000 ** 2 + 1)
    False
    """
    sq_mod256 =
(1,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,
0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,
0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,
0)
    if sq_mod256[n & 0xff] == 0:
        return False

    mt = (
        (9, (1,1,0,0,1,0,0,1,0)),
        (5, (1,1,0,0,1)),
        (7, (1,1,1,0,1,0,0)),
```

```
        (13, (1,1,0,1,1,0,0,0,0,1,1,0,1)),
        (17, (1,1,1,0,1,0,0,0,1,1,0,0,0,1,0,1,1))
    )
    a = n % (9 * 5 * 7 * 13 * 17)
    if any(t[a % m] == 0 for m, t in mt):
        return False

    return isqrt(n) ** 2 == n


def rational_to_contfrac(x: int, y: int) -> Iterator[int]:
    """
    ref: https://en.wikipedia.org/wiki/Euclidean_algorithm#Continued_fractions

    >>> list(rational_to_contfrac(4, 11))
    [0, 2, 1, 3]
    """
    while y:
        a = x // y
        yield a
        x, y = y, x - a * y


def contfrac_to_rational_iter(contfrac: Iterable[int]) -> Iterator[Tuple[int, int]]:
    """
    ref:
https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/krypto2ss08/shortsecre
texponents.pdf (6)
    """
    n0, d0 = 0, 1
    n1, d1 = 1, 0
    for q in contfrac:
        n = q * n1 + n0
        d = q * d1 + d0
        yield n, d
        n0, d0 = n1, d1
        n1, d1 = n, d
```

```
def convergents_from_contfrac(contfrac: Iterable[int]) -> Iterator[Tuple[int, int]]:
    """
    ref:
https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/krypto2ss08/shortsecre
texponents.pdf Section.3
    """
    n_, d_ = 1, 0
    for i, (n, d) in enumerate(contfrac_to_rational_iter(contfrac)):
        if i % 2 == 0:
            yield n + n_, d + d_
        else:
            yield n, d
        n_, d_ = n, d


def attack(e: int, n: int) -> Optional[int]:
    """
    ref:
https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/krypto2ss08/shortsecre
texponents.pdf Section.4

    >>> attack(2621, 8927)
    5
    >>> attack(6792605526025, 9449868410449)
    569
    >>>
attack(307496863058020618163345911672840307344780314277514955279223880
993819211726205693109454180074673064541600145978283907097708615774793
297939481034084894940252728344735558548350441533749785544144163050122
676439578389986486511007054468759795736757676053873337338765375283532
370766260945533679771340792925937464168756068767357179058922806645383
460009503436716552570463640672214698071382328204460157698824721605518
400529219303579883343066591202531147906384964800923619515365764272957
894291974835978596579778323689125347611002690655093513450507589436746
510534199825610944322581036148304483829497654599396989518244478184975
99,
109966163992903243770643456296093759130737510333736483352345488643432
614201030629970207047930115652268531222079508230987041869779760776072
105738457123387124961036111210544028669181361694095594938869077306417
```

```
325203381820822917059651429857093388618818437282624857927551285811542
685269229705594166370426152128895901914709902037365652575730201897361
139518816164746228733410283595236405985958414491372301878718635708605
256444921222945267625853091126691358833453283744166617463257821375566
155675868452032401961727814314481343467702299949407935602389342183536
222842556906657001984320973035314726867840698884052182976760066141)

422190901650907812920180123687944676069788522092850669615064693823744
099274668340988114145183193919060974344767652532554396336235392398907
6199470515758399
    """
    f_ = rational_to_contfrac(e, n)
    for k, dg in convergents_from_contfrac(f_):
        edg = e * dg
        phi = edg // k

        x = n - phi + 1
        if x % 2 == 0 and is_perfect_square((x // 2) ** 2 - n):
            g = edg - phi * k
            return dg // g
    return None

from Crypto.Util.number import long_to_bytes

# Wiener's attack function


# RSA parameters
e=699765933538082132806518868838693882924099437501652829341573921833 6726066235486817740322976584346089638904090600037858917582433694591185 1479580887086413883636727145469164114395805476827633484596543476766582001326965845887939779560005986905044528787616381453521174622991675550784306238914546319014786996253031
n=102582276377186018096867253525288600934316130579514656422785838881697321670598163650385277542949944265027826228423532237779984222276792435637448158369187151601223942866829810246352530781792020075459098791948474755746607070615856025775507477518863069191826189145327524026654161405929426127829368444037842089219
```

```
c=
53378813555780450062408881600891375443311648755268537514666147872849098216990978650710116551483879842183362326727056498268835994796640176165144738944919894572734516991079765866878591392813033378813133802466684638695419001132175616059695630065981167877805819333334304768133427707336102484646820456275061309 07
# Perform the attack
d = attack(e, n)

if d is None:
   print("Failed to recover private key.")
else:
   print("Hacked d={}".format(d))
#Hacked
d=19817564165913710368327440954104869084649126680724834800880820654952432948783
# >>> pow(c,d,n)
#
19861423537367410378888830698564358719410804547767404982836679721978935 4877
# >>> from Crypto.Util.number import long_to_bytes
# >>>
long_to_bytes(19861423537367410378888830698564358719410804547767404982836679721978935 4877)
```

My flag is —**picoCTF{proving_wiener_2635457}**

Problem 5– credstuff

First we got password and username file where each username just have one password equivalent to its index .\In challenge it says to look for cultris so i searches this in username and got its password in encrypted form then i go to caesar cipher decoder where i got the decryption of that password which is our flag

My flag is **picoCTF{C7r1F_54V35_71M3}**

Problem 6–rail-fence

We have message is Ta _7N6D49hlg:W3D_H3C31N__A97ef sHR053F38N43D7B i33___N6
We go to rail fence decoder and put msg and change the rails number to get the flag and at rails 4 i got my flag
The flag is: WH3R3_D035_7H3_F3NC3_8361N_4ND_3ND_4A76B997

Problem 7–spelling-quiz

Download public.zip and got the flag.txt file in encrypted form and in encrypted.py file i found that it just shuffle the alphabets so i used quipqiup tool where i put the encrypted flag and solve and got the value where i try with all the letter like a ,b ,c and at letter e and solve(patristocrat) then got some text which look like the flag and i put that text and it is passed so for that my flag is
picoCTF{perhaps_the_dog_jumped_over_was_just_tired}

Problem 8-Vigenere
For this i used vigenere cipher decoder where i put my message and key which is mention in challenge itself CYLAB

Message is rgnoDVD{O0NU_WQ3_G1G3O3T3_A1AH3S_2951c89f}
My flag is picoCTF{D0NT_US3_V1G3N3R3_C1PH3R_2951a89h}

Problem 9 - Double DES

We can perform like meet in middle attack we can encrypt strings after double des encryption
I write the code `#!/usr/bin/python3 -u`

```
1. from Crypto.Cipher import DES
2. from Crypto.Util.number import *
3. from itertools import product
4. from socklib import parse_nc_cmd
5. import binascii
6. import itertools
7. import random
8. import string
```

```
9.
10.
11.KEY_LENGTH = 6
12.
13.
14.def pad(msg):
15.    block_len = 8
16.    over = len(msg) % block_len
17.    pad = block_len - over
18.    return (msg + " " * pad).encode()
19.
20.def generate_key():
21.    return pad("".join(random.choice(string.digits) for _ in
   range(KEY_LENGTH)))
22.
23.
24.def get_input():
25.    try:
26.        res = binascii.unhexlify(input("What data would you like to
   encrypt? ").rstrip()).decode()
27.    except:
28.        res = None
29.    return res
30.
31.def double_encrypt(sock, m):
32.    sock.recvuntil("What data would you like to encrypt? ")
33.    sock.sendline(m)
34.    enc = sock.recvline().rstrip()
35.    return binascii.unhexlify(enc)
36.
37.with parse_nc_cmd("nc mercury.picoctf.net 33425") as s:
38.    s.recvline()
39.    flag = binascii.unhexlify(s.recvline().rstrip())
40.
41.    enc_A = double_encrypt(s, "41" * 8)
42.    enc_A = enc_A[:8]
43.    print("[+] enc_A: {}".format(enc_A))
44.
45.    # https://en.wikipedia.org/wiki/Meet-in-the-middle_attack
46.    subcipher = {}
47.
48.    for i, key in enumerate(product(string.digits, repeat=KEY_LENGTH)):
49.        if (i + 1) % 50000 == 0:
50.            print(f"[+] subkey 1: {((i+1)/10**6)*100}%")
51.        tmp_key = pad("".join(key))
52.        cipher = DES.new(tmp_key, DES.MODE_ECB)
53.        enc = cipher.encrypt(b"AAAAAAAA")
54.        subcipher[enc] = tmp_key
55.
```

```
    print("[+] subkey 1 complete!")
    key_pair = set()

    for i, key in enumerate(product(string.digits, repeat=KEY_LENGTH)):
        if (i + 1) % 50000 == 0:
            print(f"[+] subkey 2: {((i+1)/10**6)*100}%")
        tmp_key = pad("".join(key))
        cipher = DES.new(tmp_key, DES.MODE_ECB)
        dec = cipher.decrypt(enc_A)
        sub = subcipher.get(dec)
        if sub:
            key_pair.add((sub, tmp_key))

    print("[+] subkey 2 complete!")

    for k1, k2 in key_pair:
        cipher1 = DES.new(k1, DES.MODE_ECB)
        cipher2 = DES.new(k2, DES.MODE_ECB)
        dec_2 = cipher2.decrypt(flag)
        dec_1 = cipher1.decrypt(dec_2)
        print(dec_1.rstrip())
```

It connects to a server at `mercury.picoctf.net` on port `33425`
`And receive the flag in hexadecimal form`
It performs a double encryption of the string `"41" * 8`. This encrypted result is stored in `enc_A`

Meet-in-the-Middle Attack:
Subcipher Generation: It generates a dictionary subcipher where the keys are the encrypted versions of "AAAAAAAA" using all possible 6-digit digit keys, and the values are the corresponding keys.
Subkey 1 Complete: Prints a message indicating that the generation of the first set of subkeys is complete.
Subkey 2 Generation: It generates another set of subkeys and tries to decrypt enc_A using these subkeys. If the decryption result matches any of the entries in subcipher, it adds the corresponding key pairs to key_pair.
Subkey 2 Complete: Prints a message indicating that the generation of the second set of subkeys is complete.
Decryption: It iterates over the key pairs found and performs Double DES decryption on the flag using these key pairs. The decrypted flag is then printed.

My flag is `af5fa5d565081bac320f42feaf69b405`

Problem 10 -Scrambled RSA

When i go to this `mercury.picoctf.net 47987`. This gives me a flag and it ask me to enter the text it gives its encrypted value so when i enter a it gives me its encrypted value and everytime when the a enter the encrypted value is different from which i got it perform the randomization on the encrypted text so to get rid of that i write a python code which first connect to the server and got flag and n and e and after that perform brute force.

It initializes the `known_flag` variable with the prefix `"picoCTF{"` which is known to be part of the flag. It also initializes an empty list `known_list` to store the encrypted versions of known parts of the flag.
it iterates over each character of the known flag prefix and sends it to the server for encryption.
It stores the encrypted responses in `known_list`
It enters a loop that continues until the entire flag is decrypted.
For each character in `CAND_CHARS`, it appends the character to the `known_flag` and sends the updated payload to the server for encryption.

It then compares the encrypted response with the flag received from the server.

If a match is found, it updates the `known_flag` and adds the encrypted response to `known_list`.

The loop continues until the entire flag is decrypted.

```
1. from socklib import parse_nc_cmd
2. from collections import defaultdict
3. import string
4.
5. CAND_CHARS = "_}" + string.ascii_letters + string.digits
6.
7. def encrypt(sock, msg):
8.     sock.recvuntil("I will encrypt whatever you give me: ")
9.     sock.sendline(msg)
10.    sock.recvuntil("Here you go: ")
11.    return sock.recvline().rstrip()
12.
13.with parse_nc_cmd("nc mercury.picoctf.net 47987") as s:
14.    s.recvuntil("flag: ")
15.    flag = s.recvline().rstrip()
16.
17.    s.recvuntil("n: ")
```

```
    n = s.recvline().rstrip()

    s.recvuntil("e: ")
    e = s.recvline().rstrip()

    known_flag = "picoCTF{"
    known_list = []

    print("[+] gather data")

    for i in range(1, len(known_flag) + 1):
        tmp = known_flag[:i]
        enc = encrypt(s, tmp)
        if i != 1:
            for kl in known_list:
                enc = enc.replace(kl, b"")
        known_list.append(enc)

    print("[+] ok!")

    print("[+] start bruteforce 💪")
    while True:
        for prog, cand in enumerate(CAND_CHARS):
            payload = known_flag + cand
            enc = encrypt(s, payload)
            for kl in known_list:
                enc = enc.replace(kl, b"")
            if flag.find(enc) != -1:
                known_flag = payload
                known_list.append(enc)
                break
        if cand == "}":
            break
        else:
            print(f"[+] found char: {cand}")
            print(f"[+] current flag: {known_flag}")
    print(f"[+] finish bruteforce 👍")
    print(f"[+] flag: {known_flag}")
```

My flag is **picoCTF{bad_1d3a5_7571572}**