

## Exchange and Computation History (I'm the guy on the right in the screenshots)

### RSA

I'm Alice



```
if __name__ == "__main__":  
    # print(fast_expo_modulo(55116,50237,80687))  
    print(fast_expo_modulo(base: 123, power: 575869439, modulus: 1864046491))
```

Run

fast\_expo\_modulo ×

/usr/local/bin/python3.11 /Users/Mark\_1/Downloads/Cryptography/code\_assignments/code/fast\_expo\_  
1348505685

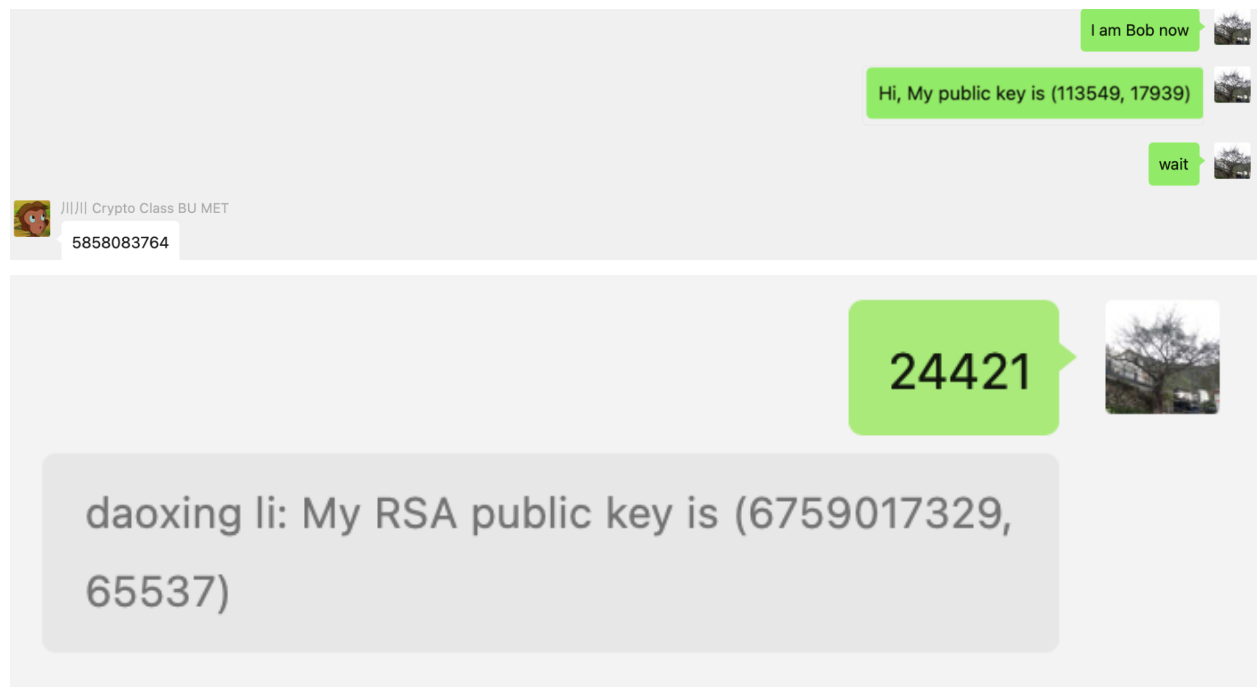
I'm Bob

Public key generation

```
nts [cryptography] sources root, ~/Downloads/Cr 13
16
22
23
24
25
26
27
28
29
30
31
32
33
34
class MyTestCase(unittest.TestCase):
    def test1(self):
        self: <break_rsa_test.MyTestCase testMethod=test1>
        n = p * q    n: 113549
        phi_of_n = (p - 1)*(q - 1)    phi_of_n: 112860
        e = 2    e: 17939
        while phi_of_n % e == 0: # that is, gcd(e, phi_of_n) = 1
            e = sympy.randprime(a=2, phi_of_n - 1)
        public_key = (n, e)    public_key: (113549, 17939)
        # receiver generates private key
        private_key = rsa_private_key_generator(public_key, p, q)    private_key: (113549, 30299)
        # sender generates the message
        message = random.randint(a=2, n - 1) # 1<x<n
        while euclidean(message, n) != 1:
            message = random.randint(a=2, n - 1)
        # sender encrypt the message

Python tests in break_rsa_test.py x
Debugger Console
Evaluate expression (⌘) or add a watch (⌘⌘)
test1, break_rsa    e = (int) 17939
                    n = (int) 113549
                    p = (int) 419
                    phi_of_n = (int) 112860
                    private_key = (tuple: 2) (113549, 30299)
                    public_key = (tuple: 2) (113549, 17939)
                    q = (int) 271
                    self = (MyTestCase) <break_rsa_test.MyTestCase testMethod=test1>
```

Message Decryption:



```
if __name__ == "__main__":
    # print(rsa_encrypt())
    print(rsa_decrypt(rsa_private_key_generator( public_key: (113549, 17939), prime_p: 419, prime_q: 271), ciphertext: 5858083764))
```

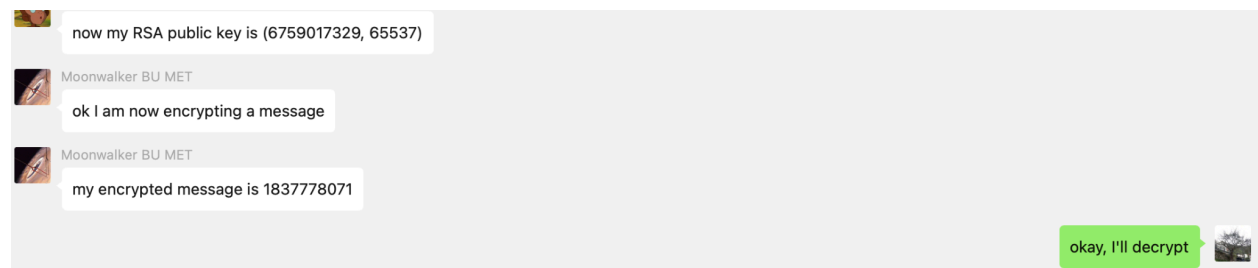
Run

Run: rsa\_without\_padding ×

/usr/local/bin/python3.11 /Users/Mark\_1/Downloads/Cryptography/code\_assignments/code/rsa\_without\_padding.py

24421

I'm Eve



```
63
64
65 print(break_rsa_with_pollard_rho( public_key: (6759017329, 65537), ciphertext: 1837778071))
66
```

Run

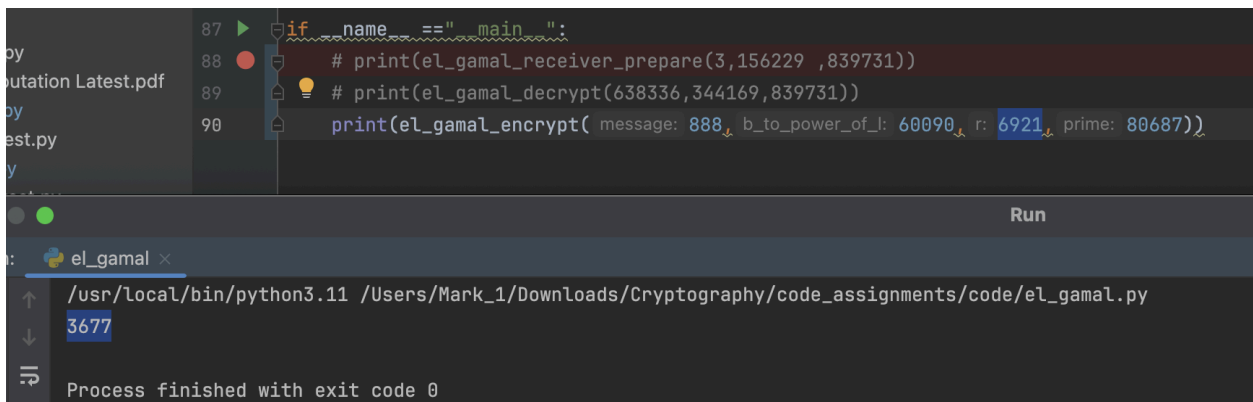
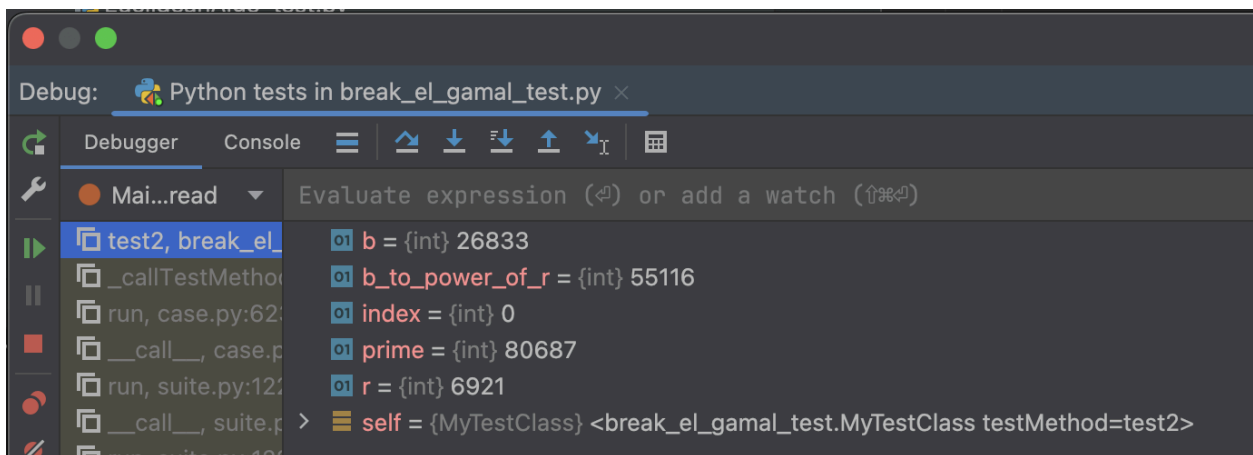
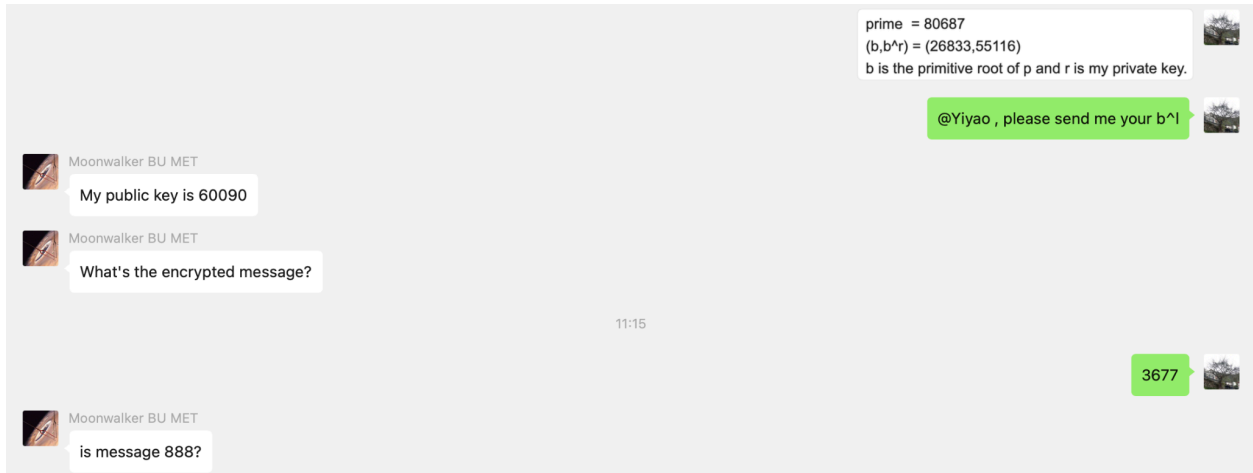
Run: break\_rsa ×

/usr/local/bin/python3.11 /Users/Mark\_1/Downloads/Cryptography/code\_assignments/code/break\_rsa.py

178


## EL GAMAL



I'm Alice




I'm Bob


Private Key Calculation:

 Jlljll Crypto Class BU MET  
 prime = 839731  
 b = 3 b^r = 156229

got it, one sec >   
 i need to recalculate b^l > 

10:27

 Moonwalker BU MET  
 what's your public key for this?@Liangmi Zhang


b^l = 517945 > 

```

def el_gamal_receiver_prepare(b, b_to_power_of_r, prime):    b: 3    b_to_power_of_r: 156229    prime: 839731
    """
    for receiver to do precomputations
    :param b: integer, which is the primitive root of the group
    :param b_to_power_of_r: which is b^r mod p, shared by the sender.
    :param prime: a prime number
    :return:
    brl, an integer, which is (b^r)^l mod prime ;
    b_to_power_of_l, an integer, which is (b^l) mod prime ;
    brl_inverse, and integer, which is the inverse of brl in group U(prime)
    """
    # check the primality of the argument prime
    if not is_prime(prime):
        raise Exception(f"{prime} is not prime. Argument should be prime.")

    # l = random_number_generator(0, prime - 2)
    l = 839    l: 839
    # generate a private key
    b_to_power_of_l = fast_expo_modulo(b, l, prime)    b_to_power_of_l: 517945~
    # computes (b^r)^l
    brl = fast_expo_modulo(b_to_power_of_r, l, prime)    brl: 730095
    # computes inverse of (b^r)^l in G
    brl_inverse = extended_euclidean_2(brl, prime)[0]    brl_inverse: 344169
    return brl, b_to_power_of_l, brl_inverse
  
```

Decryption:

plaintext is 178 > 


daoxing li: the encrypted message is 638336

```
87
88 if __name__ == "__main__":
89     # print(el_gamal_receiver_prepare(3,156229 ,839731))
90     print(el_gamal_decrypt( cipher_text: 638336, brl_inverse: 344169, prime: 839731))
91     # print(el_gamal_encrypt(888,41404,55116,80687))


Run

el_gamal x
/usr/local/bin/python3.11 /Users/Mark_1/Downloads/Cryptography/code_assignments/code/el_gamal.py
178
Process finished with exit code 0
```


I'm Eve

 Moonwalker BU MET


Public key (p, b, x): (205823, 94379, 27883)

 川川 Crypto Class BU MET

$b^x = 27883$

 Moonwalker BU MET

Encrypted message: 25774002

 Moonwalker BU MET

That's right

Liangmi Zhang: is plaintext 951?

Yiyao: Encrypted message: 25774002

is plaintext 951?

```
ation Latest.pdf 22
23
24 if __name__ == "__main__":
25     print(break_el_gamal( cipher_text: 25774002, p: 205823, b: 94379, b_to_power_of_r: 27883, b_to_power_of_l: 27883))
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Run

break\_el\_gamal x

/usr/local/bin/python3.11 /Users/Mark\_1/Downloads/Cryptography/code\_assignments/code/break\_el\_gamal.py

951

Process finished with exit code 0