Nicholas Vlahos-Sten

Jeff Ondich

Computer Security

27 September 2025

# Being Eve

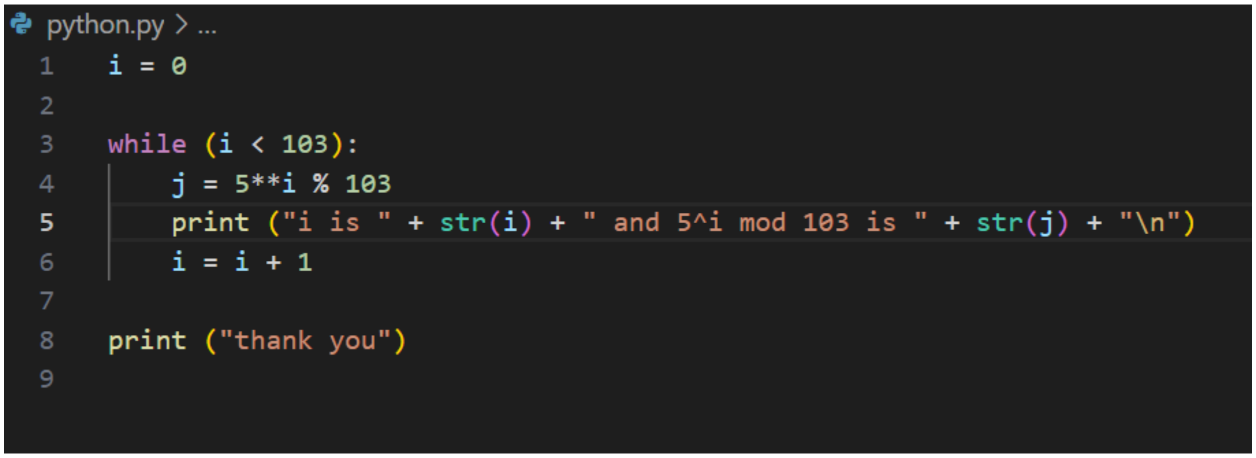
COLLABORATION NOTE: Jane helped me figure out how the plain text in Alice's RSA message was encoded, and how to decode it. I did everything up until that point by myself, including decrypting RSA.

## Diffie Hellman

The shared key K is 31.

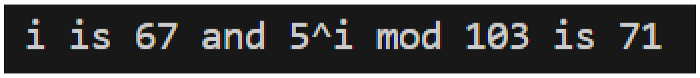
This was a funny problem, as I spent like 40 minutes trying to figure out how to "cleverly" solve this, noting stuff like "wow all exponents of 5 end with 25, maybe I could use that" and trying all sorts of nonsense that didn't work, until I said, "screw it, I could brute force this is 5 minutes." I was wrong. It took me 8 minutes to brute force it.

By "brute force" I mean I wrote a python program that calculated every possible value of g^x mod 103, where x < 103. Then from looking at the list of outcomes, I could see what values of x got me the results 71 and 10, which would tell me Bob and Alice's secret numbers, respectively.

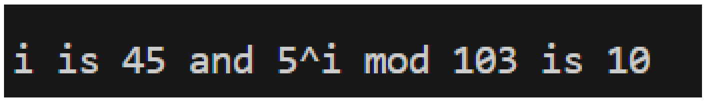


```
python.py > ...
i = 0

while (i < 103):
    j = 5**i % 103
    print ("i is " + str(i) + " and 5^i mod 103 is " + str(j) + "\n")
    i = i + 1

print ("thank you")
```

Woah I found them

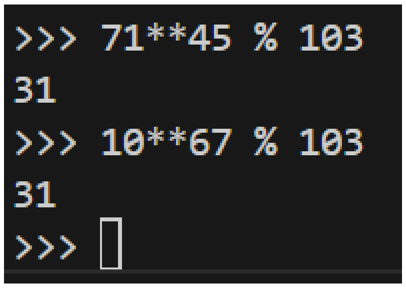


```
i is 67 and 5^i mod 103 is 71
```



```
i is 45 and 5^i mod 103 is 10
```

Now it’s just a value of plugging them both into the formula to find the secret key, and confirm that yes, it is the same secret key.



```
>>> 71**45 % 103
31
>>> 10**67 % 103
31
>>>
```

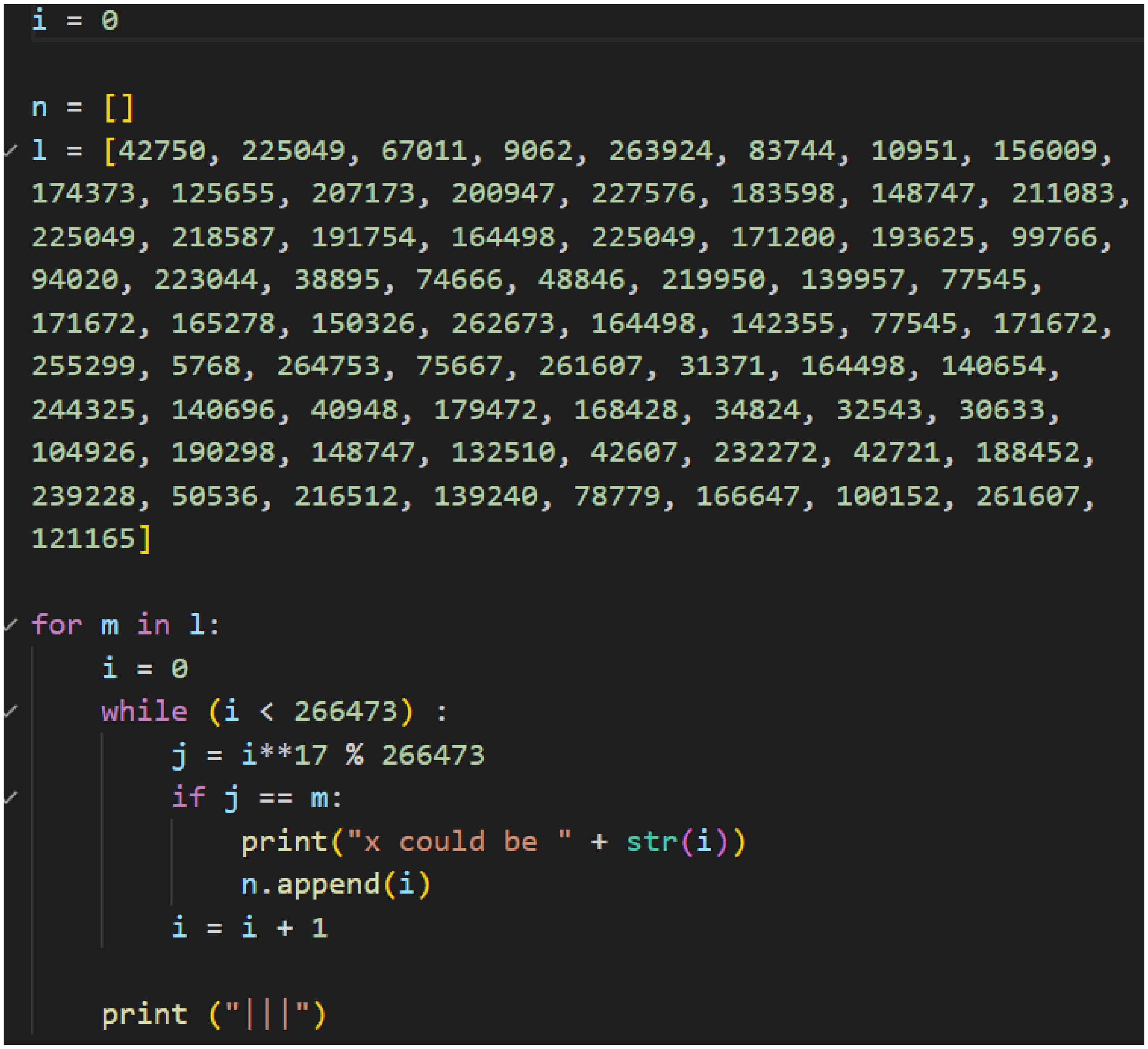
So yeah. K = 31.

My brute force algorithm would obviously not work on a very large p, because it runs in O(p) (it has to do 1 calculation for every number less than p because it has to check all of the possibilities for Alice and Bob’s secret numbers) and p getting very large would make this program take forever to run.

## RSA

The message is “Hey Bob, here's some cryptography history for you (https://en.wikipedia.org/wiki/The_Magic_Words_are_Squeamish_Ossifrage). Happy factoring, Alice.”

I solved it by doing the same brute force approach I did for Diffie Hellman, because that worked really well last time. I know that, for each encrypted integer i in the intercepted message, the corresponding integer x in the plain text message would fit the following equation: x**17 mod 266473 = i. So, after realizing with some experiments that there was only one possible x for each i, I just wrote a program that tried every possible value of x for each corresponding i, by plugging every number less than 266473 into the equation to see if it equaled i.



```
i = 0

n = []
l = [42750, 225049, 67011, 9062, 263924, 83744, 10951, 156009,
174373, 125655, 207173, 200947, 227576, 183598, 148747, 211083,
225049, 218587, 191754, 164498, 225049, 171200, 193625, 99766,
94020, 223044, 38895, 74666, 48846, 219950, 139957, 77545,
171672, 165278, 150326, 262673, 164498, 142355, 77545, 171672,
255299, 5768, 264753, 75667, 261607, 31371, 164498, 140654,
244325, 140696, 40948, 179472, 168428, 34824, 32543, 30633,
104926, 190298, 148747, 132510, 42607, 232272, 42721, 188452,
239228, 50536, 216512, 139240, 78779, 166647, 100152, 261607,
121165]

for m in l:
    i = 0
    while (i < 266473) :
        j = i**17 % 266473
        if j == m:
            print("x could be " + str(i))
            n.append(i)
        i = i + 1

    print ("|||")
```

This is where the approach would have fallen apart had the numbers in the public key been much larger. Doing 266473 different numbers to the 17th and doing it one time for each integer in the encrypted message (this is disgustingly O(n*p) where n is the number of integers in the encrypted message and p is the second value in Bob’s private key) is ridiculous. Even this program took a good 10-15 seconds to run every time which is a really really bad sign. It’s just not possible to brute force calculate bigger numbers efficiently.

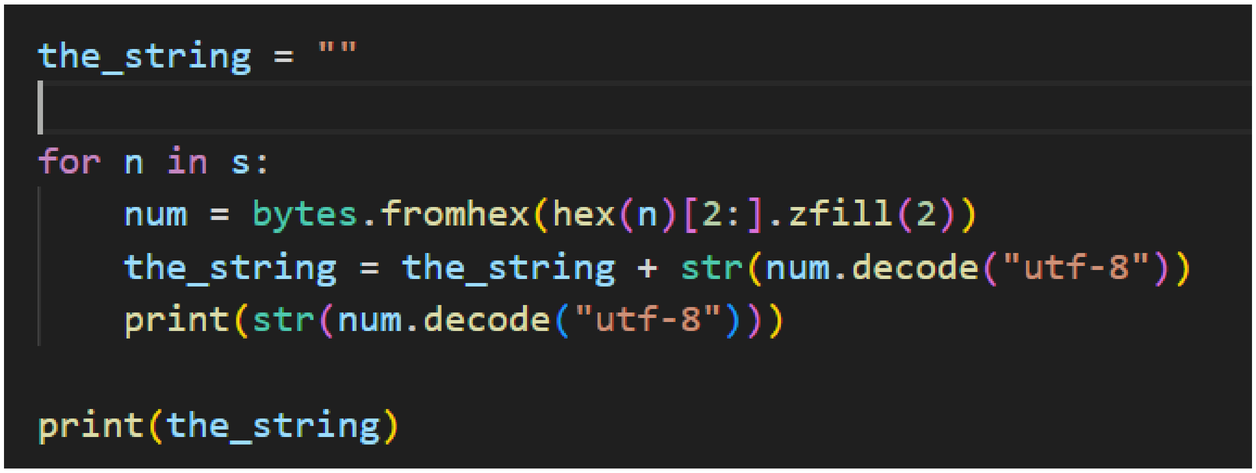
Anyway, from here I got the following decrypted message:

[18533, 31008, 17007, 25132, 8296, 25970, 25895, 29472, 29551, 28005, 8291, 29305, 28788, 28519, 29281, 28776, 31008, 26729, 29556, 28530, 31008, 26223, 29216, 31087, 29984, 10344, 29812, 28787, 14895, 12133, 28206, 30569, 27497, 28773, 25705, 24878, 28530, 26415, 30569, 27497, 12116, 26725, 24397, 24935, 26979, 24407, 28530, 25715, 24417, 29285, 24403, 29045, 25953, 28009, 29544, 24399, 29555, 26982, 29281, 26469, 10542, 8264, 24944, 28793, 8294, 24931, 29807, 29289, 28263, 11296, 16748, 26979, 25902]

And then Jane helped me decode the decrypted message and told me to use this website to write a decoding function:

https://www.geeksforgeeks.org/python/how-to-convert-int-to-bytes-in-python/#using-bytesfromhex

Anyway, Alice encoded her message by putting **2** (!) UTF8 characters in each integer. What that meant was that she shoved the binary of one UTF8 character right after the binary of the one before that, turned the combined binaries into one singular decimal integer, and encrypted each decimal integer with RSA. I wrote a program to split these integers back into 2 hexadecimal numbers and convert those to UTF-8.



```
the_string = ""

for n in s:
    num = bytes.fromhex(hex(n)[2:].zfill(2))
    the_string = the_string + str(num.decode("utf-8"))
    print(str(num.decode("utf-8")))

print(the_string)
```

So the message again is: “Hey Bob, here's some cryptography history for you (https://en.wikipedia.org/wiki/The_Magic_Words_are_Squeamish_Ossifrage). Happy factoring, Alice.”

Anyway, this method of encoding is still insecure even if we use larger numbers in the public key. Because we are encoding and encrypting each set of two characters in the exact same way, we know that if a set of two characters (“ee” for example) appears twice, each will translate to the same integer in the encrypted message. And this actually happens in this example (notice that 17162 appears twice in the encrypted message). So not only does this repetition of oddly specific numbers tell me that each integer is encoded and encrypted in the same way, which is super useful to Eve, pattern recognition can then be used to potentially decrypt and decode the message.