David Toledo

9/29/24

Diffie Hellman

- The shared secret between Alice and Bob is 65.

  First, I wrote out what I knew. G was 7 and P = 97, and using what we saw from the lab, I set up the equations needed to find out the integers picked by each one. We know that $53 = 7^X \% 97$, and $82 = 7^Y \% 97$. I wrote this code to solve this.

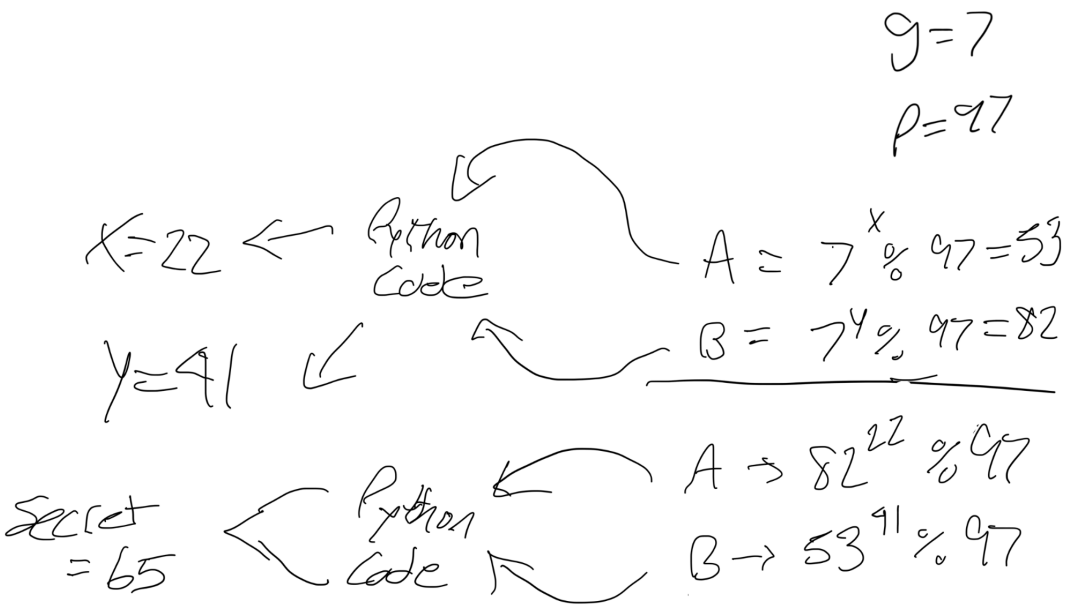
```
for x in range(1,98):
    if (7**x)%97 == 53:
        print("x is ", x)

for y in range(1,98):
    if (7**y)%97 == 82:
        print("y is ", y)
```

  This gave me the numbers needed to get the A and B Alice and Bob sent each other, X = 22 and Y = 41. From here, we can solve for what the secrets where.

```
a = (82**22)%97
print("a is ", a)

b = (53**41)%97
print("b is ", b)
```

- Show precisely where in your process you would have failed if the integers involved were much larger.

It gets hard to pass this when the numbers become really big, as it becomes hard to look for a number efficiently. They get too large enough to check one by one for modding, making it take forever to find the answer.



RSA

Alices Message:
Dear Bob, check this out. https://www.surveillancewatch.io/ See ya, Alice.

Work: First I looked up the factors of 162991. From there I saw that there were 4 factors, 1, 389, 419, and 162991. This meant that p_B and q_B were 389 and 419 respectively. To compute $\lambda(n_B)$, I looked up the least common multiple calculator and imputed 388 and 418, which resulted in 81092. From here, we were told from the public key that e was 13. From there I looked a greatest common factor calculator and confirmed that the GCF was 1. From here, I wrote a Python script to calculate $13*d_B$ % 81092 = 1

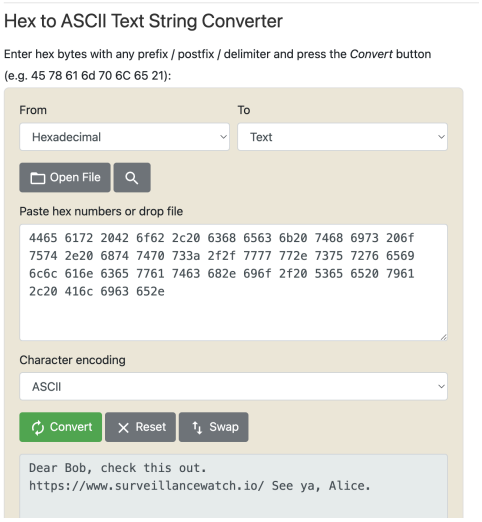
```
for dB in range(1,100000):
    if 13*dB%81092 == 1:
        print("dB is ",dB)
```

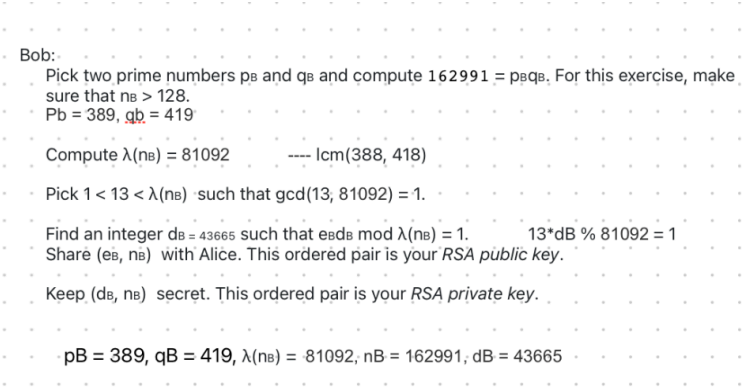
From here, I was told that d_B is 43665. Now I have all that I need to decrypt the message. I created this to apply my secret key to all of the numbers.

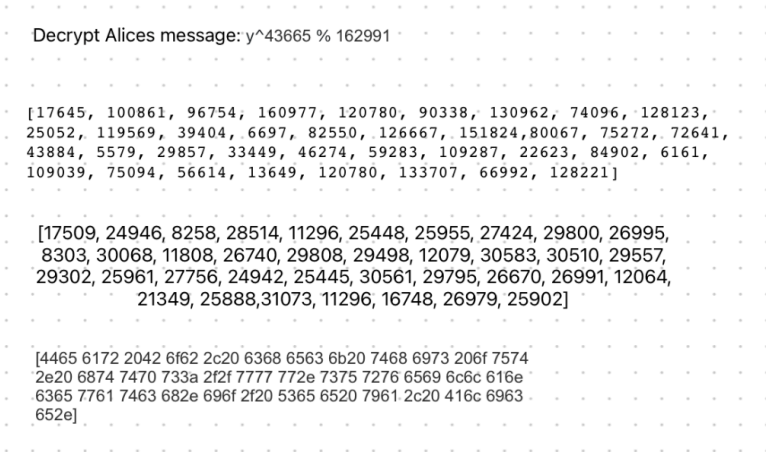
```
#Decrypting message

message = [17645, 100861, 96754, 160977, 120780, 90338, 130962, 74096, 128123, 25052,
119569, 39404, 6697, 82550, 126667, 151824,80067, 75272, 72641, 43884, 5579, 29857,
33449, 46274, 59283, 109287, 22623, 84902, 6161, 109039, 75094, 56614, 13649, 120780,
133707, 66992, 128221]

chars = []
for num in message:
    new = (num**43665) % 162991
    chars.append(new)


hexChar = []
for num in chars:
    new = hex(num)
    hexChar.append(new)
print(hexChar)
```

This gave me a new list, but the numbers were too big to be ASCII. So I did what I was told in class, converted them to hex, and then put those hex numbers into the Hex to ASCII converter. This displayed the message I was looking for.

The way Alice encoded her message was by first turning the text into ASCII, and then taking those ASCII values and turning them into hexadecimal. From that, every 4 hexadecimal number was converted back into decimal. From there, Alice applied hey key to the numbers created, and sent this as a message.







This process would've failed in the same aspect because the numbers would become too inefficient to check individually. Looking for d_b, I had to take up the range until 100000, as the numbers were getting really big. If they got bigger, the numbers would get bigger and bigger, and it would take longer and longer to check for this number.

Alice's message would be insecure even if Bob's keys involved larger integers because there are still possible attacks that would be intercepted, such as the man-in-the-middle attacks where the eavesdropper still would know enough information to meddle with the conversation. Also, with enough time people could still probably break the code by checking the numbers individually if they were still a little bigger. Her encoding is very easy to break with modern tools, and so this encoding adds more of an inconvenience to an eavesdropper more than a security layer. Her message still is vulnerable regardless of the size of the key.