Ryan Son

CS338

Jeff Ondich

September 28, 2024

# Being Eve

## ***Diffie Hellman Decryption:***

Alice and Bob: (g = 7) and (p = 97)

Alice sent Bob the number 53 → A and Bob sent Alice the number 82 → B

Using these values, we can calculate for x and y respectively:

- 53 = A = g^x mod p
- 82 = B = g^y mod p

Using python to brute force calculate x and y with a for loop where:

- a = g ** x % p
- b = g ** y % p

```
for k in range (1,100):
    a = 7 ** k % 97
    if (a == 53):
        print(k)

for k in range (1,100):
    b = 7 ** k % 97
    if (b == 82):
        print(k)
```

Where k in each respective for loop is the calculated value of x and y.

The Shared Secret is then calculated by A^y mod p = B^x mod p:

- 53^41 mod 97 = 82^22 mod 97 = 65

Resulting in the Shared Secret = 65

This approach would be very difficult if not impossible with large numbers because calculating for x and y with a for loop would span a very large range of values.

***RSA Encryption:***

Bob's Public Key: (e_Bob, n_Bob) = (13, 162991)

While attempting this problem, I came across two possible approaches. The first approach is very similar to how I found the Shared Secret in the Diffie Hellman decryption exercise, involving a for loop that completed calculations that would output a value if it matched a certain criterion. The second approach involved finding Bob's Secret Key using the given Public Key.

**Approach One:**

```
encrypted_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]

orig_string = ""

for message_encrypted in encrypted_message:
    #print(message)
    for message_orig in range (1,100000):
        e_Bob = 13
        n_Bob = 162991
        encryption = (message_orig ** 13) % n_Bob
        if (encryption == message_encrypted):
            orig_string += str(message_orig)
            orig_string += " "

print(orig_string)
```

This approach matched the calculated encrypted value with the actual encrypted value found in the encrypted_message array by guessing what the original message could be. If the encryption values matched each other, the possible original message was added to the orig_string variable and printed.

Where orig_string is a string containing the decimal values of the decrypted message below:

17509 24946 8258 28514 11296 25448 25955 27424 29800 26995 8303 30068 11808 26740

29808 29498 12079 30583 30510 29557 29302 25961 27756 24942 25445 30561 29795 26670

26991 12064 21349 25888 31073 11296 16748 26979 25902

**Approach Two: Private Key (d_Bob, n_Bob)**

Find the prime numbers that equal n_Bob: 162991 = p * q → 389 * 419

Then find the least common multiple of (p-1, q-1) → 81092

Then confirm that this value is correct by confirming the greatest common divisor between g (13) and 81092 is 1.

Then next the value of d_Bob can be found using g * d mod lcm(n) = 1

- 13 * d mod 81092 = 1, where d = 43665 found using the code below:

```
for d in range (1,100000):
    answer = (13 * d) % 81092
    if answer == 1:
        print(d)
```

Then the standard RSA decryption using the Private Key was done using the code below:

```
orig_string = ""

for message_encrypted in encrypted_message:
    d_bob = 43665
    n_Bob = 162991
    decrypted = (message_encrypted ** d_bob) % n_Bob
    print(hex(decrypted))
    orig_string += str(decrypted) + " "
```

Where orig_string is a string containing the decimal values of the decrypted message below:

17509 24946 8258 28514 11296 25448 25955 27424 29800 26995 8303 30068 11808 26740

29808 29498 12079 30583 30510 29557 29302 25961 27756 24942 25445 30561 29795 26670

26991 12064 21349 25888 31073 11296 16748 26979 25902

This would not work that well with large numbers because it would be computationally difficult to try to find two prime numbers that are probably very large that multiply up to a large n_Bob value.

This method of encryption would also be insecure because pattern recognition of each character could be used to identify what encrypted values correspond to characters found in the message. Those can then be used to either construct the larger message or discover what values are required to decrypt the whole message.