Mary Blanchard and Angela Ellis

Diffie-Hellman

- Shared secret: $K = 36$
- To find the shared secret, we first worked to find Alice and Bob's secret numbers. We used a Python script to test all of the possible integers for each person's secret number that were less than $p = 59$ (Figure 1) to see if they would result in the publicly exchanged numbers. This told us that Alice's secret number was x = 36 and Bob's secret number was y = 15 (Figure 2). We then used Alice and Bob's publicly exchanged numbers ($A = 57$, $B = 44$) to calculate the shared key for each of them. Both resulted in $K = 36$, which matched what we would expect because the secret key will always be g ** (x * y) mod p (Figure 3).
- Our process for finding the shared key would begin to fall apart while brute-force finding the possibilities of the secret numbers. For larger integers, there are many more possibilities of what Alice and Bob's secret numbers could be. Keeping g and p the same, here is what our script returned when we tested numbers up to 1000 (Figure 4).

```
1  for i in range(1,59):
2      if ((11**i)%59 == 57):
3          print("a: " , i)
4
5  for i in range(1,59):
6      if ((11**i)%59 == 44):
7          print("b: ",i)
8
```

Figure 1

```
acellis@DESKTOP-KTKT4D7:/mnt/c/CS/cs338/cryptography$ python3 dh.py
b:  15
a:  36
```

Figure 2

```
>>> (11**(15*36))%59
36
>>> (44**36)%59
36
>>> (57**15)%59
36
```

Figure 3

```
acellis@DESKTOP-KTKT4D7:/mnt/c/CS/cs338/cryptography$ python3 dh.py
b:  15
b:  73
b:  131
b:  189
b:  247
b:  305
b:  363
b:  421
b:  479
b:  537
b:  595
b:  653
b:  711
b:  769
b:  827
b:  885
b:  943
a:  36
a:  94
a:  152
a:  210
a:  268
a:  326
a:  384
a:  442
a:  500
a:  558
a:  616
a:  674
a:  732
a:  790
a:  848
a:  906
a:  964
```

Figure 4

RSA

- Encrypted message: “Hey Bob. It's even worse than we thought! Your pal, Alice. https://www.schneier.com/blog/archives/2022/04/airtags-are-used-for-stalking-far-more-than-previously-reported.html”
- Alice’s message was first encoded using ASCII, so each character was converted into an integer between 0 and 127. Alice then encrypted the message using Bob’s public key following the RSA encryption process of (ASCII value)** e_Bob mod n_Bob. In order to decrypt the message, we wrote a script that could test each ASCII value from 0 to 127. When the expression (ASCII value)** e_Bob mod n_Bob matched the encrypted number, we saved that ASCII value in our list of decrypted characters (Figure 5). After that, we used a Python function to turn each decrypted ASCII value into a character, which allowed us to print out the plaintext message (Figure 6).
- If Bob’s key numbers were higher, our process could fail at line 24 in our decryption code. The encryption process could start to produce numbers that are too big for the computer to store efficiently, and the calculations would start to take longer. For example, char ** e_Bob would fail if e_Bob got too large. Our method would also begin to break down if the encoding was done with something that uses more space or complex numbers, such as Unicode instead of ASCII.
- Even if Bob’s key numbers were higher or the original encoding used larger integers than ASCII, Alice’s form of message encoding would still be insecure because she is encoding character by character. This method of encoding is easy to reverse using letter frequencies and other strategies that don’t have anything to do with how the numbers directly correspond to the characters.

[figures on next page]

rsa.py

```python
e_Bob = 13
n_Bob = 5561
encrpyted_message = [1516, 3860, 2891, 570, 3483, 4022, 3437, 299, 570, 843,
3433, 5450, 653, 570, 3860, 482, 3860, 4851, 570, 2187, 4022, 3075, 653, 3860,
 570, 3433, 1511, 2442, 4851, 570, 2187, 3860, 570, 3433, 1511, 4022, 3411,
 5139, 1511, 3433, 4180, 570, 4169, 4022, 3411, 3075, 570, 3000, 2442, 2458,
 4759, 570, 2863, 2458, 3455, 1106, 3860, 299, 570, 1511, 3433, 3433, 3000,
 653, 3269, 4951, 4951, 2187, 2187,  2187, 299, 653, 1106, 1511, 4851, 3860,
 3455, 3860, 3075, 299, 1106, 4022, 3194, 4951, 3437, 2458, 4022, 5139, 4951,
 2442, 3075, 1106, 1511, 3455, 482, 3860, 653, 4951, 2875, 3668, 2875, 2875,
 4951, 3668, 4063, 4951, 2442, 3455, 3075, 3433, 2442, 5139, 653, 5077, 2442,
 3075, 3860, 5077, 3411, 653, 3860, 1165, 5077, 2713, 4022, 3075, 5077, 653,
 3433, 2442, 2458, 3409, 3455, 4851, 5139, 5077, 2713, 2442, 3075, 5077, 3194,
 4022, 3075, 3860, 5077, 3433, 1511, 2442, 4851, 5077, 3000, 3075, 3860, 482,
 3455, 4022, 3411, 653, 2458, 2891, 5077, 3075, 3860, 3000, 4022, 3075,
 3433, 3860, 1165, 299, 1511, 3433, 3194, 2458]

 decrypted_message = []
# for each encrypted character
for num in encrpyted_message:
    # check each possible ascii value
    for i in range(128):
        # if we find an ascii value that encrypts and matches,
        if i ** e_Bob % n_Bob == num:
            # add it to our decrypted list
            decrypted_message.append(i)
# show us the list of asciis
print(decrypted_message)
# turn the decrypted ascii values into readable chars
decrypted_string = ""
for char in decrypted_message:
    decrypted_string += chr(char)
print("\n\n", decrypted_string)

```

Figure 5

```
acellis@DESKTOP-KTKT4D7:/mnt/c/CS/cs338/cryptography$ python3 rsa.py
[72, 101, 121, 32, 66, 111, 98, 46, 32, 73, 116, 39, 115, 32, 101, 118, 101, 110, 32, 119, 111, 114, 115, 101, 32, 116,
104, 97, 110, 32, 119, 101, 32, 116, 104, 111, 117, 103, 104, 116, 33, 32, 89, 111, 117, 114, 32, 112, 97, 108, 44, 32,
65, 108, 105, 99, 101, 46, 32, 104, 116, 116, 112, 115, 58, 47, 47, 119, 119, 119, 46, 115, 99, 104, 110, 101, 105, 101,
 114, 46, 99, 111, 109, 47, 98, 108, 111, 103, 47, 97, 114, 99, 104, 105, 118, 101, 115, 47, 50, 48, 50, 50, 47, 48, 52,
 47, 97, 105, 114, 116, 97, 103, 115, 45, 97, 114, 101, 45, 117, 115, 101, 100, 45, 102, 111, 114, 45, 115, 116, 97, 108
, 107, 105, 110, 103, 45, 102, 97, 114, 45, 109, 111, 114, 101, 45, 116, 104, 97, 110, 45, 112, 114, 101, 118, 105, 111,
 117, 115, 108, 121, 45, 114, 101, 112, 111, 114, 116, 101, 100, 46, 104, 116, 109, 108]


 Hey Bob. It's even worse than we thought! Your pal, Alice. https://www.schneier.com/blog/archives/2022/04/airtags-are-u
sed-for-stalking-far-more-than-previously-reported.html
```

Figure 6