Being Eve

Josiah Misplon (and no one else)

For both parts of the assignment, I coded my approach in Python. Largely, I followed the methods described in the lectures/videos.

Diffie Helman:

Code:

```
#### Diffie Helman

powers_list=[]
multiples_list=[]
g=17
p=61
## A=46 and B=5
def find_exponent(g,p,dif):
    for power in range(1,10000):
        if (g**power - dif) % p == 0:
            print(str(power) + ' is the exponent you seek for ' + str(dif))
            return power

a = find_exponent(g,p,46)
b = find_exponent(g,p,5)
key_k = (g**(a*b)%p)
print("The shared secret is: " + str(key_k))
```

The output from the code:

```
14 is the exponent you seek for 46
26 is the exponent you seek for 5
The shared secret is: 12
```

The shared secret, as the output shows, is 12.

As the code also shows, to find the values for “a” and “b” that Alice and Bob, respectively, generated, I employed a brute force search to find the correct exponents. This approach would fail if the integers g and p were significantly larger. For the small values of g and p we were given, simply looping over potential exponents is relatively quick, but for large values this method is quite slow (I tried a few values for g and p in the low millions, and it ran for several minutes without getting to an answer).

RSA:
Code:

```
#### RSA section
e = 31
n = 4661
prime_factors=[59,79]
d=0
message_encrypted = [2677, 4254, 1152, 4645, 4227, 1583, 2252, 426, 3492, \
    4227, 3889, 1789, 4254, 1704, 1301, 4227, 1420, 1789, 1821, 1466, 4227, \
    2252, 3303, 1420, 2234, 4227, 4227, 1789, 1420, 1420, 4402, 1466, 4070, 3278,\
    3278, 414, 414, 414, 2234, 1466, 1704, 1789, 2955, 4254, 1821, 4254, 4645, \
    2234, 1704, 2252, 3282, 3278, 426, 2991, 2252, 1604, 3278, 1152, 4645, 1704, \
    1789, 1821, 4484, 4254, 1466, 3278, 1512, 3602, 1221, 1872, 3278, 1221, 1512, \
    3278, 4254, 1435, 3282, 1152, 1821, 2991, 1945, 1420, 4645, 1152, 1704, 1301, \
    1821, 2955, 1604, 1945, 1221, 2234, 1789, 1420, 3282, 2991, 4227, 4410, 1821, \
    1301, 4254, 1466, 3454, 4227, 4410, 2252, 3303, 4645, 4227, 3815, 4645, 1821, \
    4254, 2955, 2566, 3492, 4227, 3563, 2991, 1821, 1704, 4254]
for d_temp in range(1,1000000):
    if (e * d_temp) % ((58) * (78)) == 1:
        d = d_temp
        break

message_decrypted=""

for number in message_encrypted:
    decrypted_number = ((number ** d) % n)
    message_decrypted+= chr(decrypted_number)

print(str(d)+" is the value for d")
print(message_decrypted)
```

The output from the code:

```
2335 is the value for d
Dear Bob, Check this out.  https://www.schneier.com/blog/archives/2017/12/e-mail_tracking_1.html Yikes! Your friend, Alice
```

So, the secret message is (a little larger for seeing easily):

Dear Bob, Check this out. https://www.schneier.com/blog/archives/2017/12/e-mail_tracking_1.html Yikes! Your friend, Alice

As the code shows, my approach for finding the correct d was a brute force search to find the first appropriate value. While this could be slow (though in the case of multiplication and not exponents, there seem to be identifiable patterns which might lead to smarter approaches to finding d), the actual main point where my approach would break is factoring of 4661 into 59

and 79. For that, I used Wolfram Alpha (the coding approach that first jumps out to me is using the sieve of Eratosthenes – which we implemented in Programming languages). For much larger values of n, factoring n into its prime factors can take tremendous amounts of time.