Being Eve

Author: Lily Li
Conferred With: Jeff Ondich

1. Diffie Hellman

a. Shared Secret Key: **36**

b. Process *(with Python Code)*:

i. Find Alex's X and Bob's Y with ***brute force***
By trying every single possible X to satisfy A = g^X mod p, and similar for Y.
(**This would have failed if the integers involved were much larger**, because it may take till the end of the universe to figure that out. In classical computing, with numbers sufficiently large, there are currently no efficient algorithms to integer factorization)

```
# Brute force find Alice's X
print("Diff Begins:")
for x in range(p):
    if g**x % p == A:
        print("Alice X:",x)
        break

# Brute force find Bob's Y
for y in range(p):
    if g**y % p == B:
        print("Bob Y:",y)
        break
```

ii. Calculate the shared secret key with B^X mod p and A^Y mod p, and double check that Alex and Bob got the same secret key.

```
Alice_key = B**x % p
Bob_key = A**y % p

if Alice_key == Bob_key:
    print("Shared Key Established = ", Bob_key)
else:
    print("Shared key not established.")
    exit()
```

iii. Code Results:

```
Diffie Begins:
Alice X: 36
Bob Y: 15
Shared Key Established =  36
```

2. RSA

a. The 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

b. Process *(with Python Code)*:

i. Find q_Bob and p_Bob with ***brute force***.

We can do that because we know n_Bob is the product of two distinct prime numbers q_Bob and p_Bob, which means that n_Bob has only 4 factors: 1, itself, q_Bob, and p_Bob. Once we find one of q_Bob and p_Bob, we can easily find the other one. There is no difference in calculation no matter which one is which. (**This would have failed if the n_Bob involved were much larger**, because. Just as in Diffie, in classical computing, with numbers sufficiently large, there are currently no efficient algorithms for integer factorization.)

```
print("RSA Begins:")
n_Bob = 5561
e_Bob = 13
# Brute force to find p_Bob, and q_Bob
for s in range(1,n_Bob):
    if n_Bob % s == 0 and s!=1 and s!= n_Bob:
        print("p_Bob is",s)
        p_Bob = s
        break

q_Bob = n_Bob//p_Bob
print("q_Bob is",q_Bob)
```

ii. Find λ(n_Bob)

We've found p_Bob and q_Bob.
We know λ(n_Bob) = lcm(p_Bob - 1, q_Bob - 1).
Also, lcm(a, b)= (a × b)//gcd(a, b). So we can use gcd to find lcm.
Gcd can be efficiently computed with the Euclidean Algorithm, so it's okay even if our number is large.

```
lambda_nBob = (q_Bob-1)*(p_Bob-1)//math.gcd(q_Bob-1, p_Bob-1)
print("λ_nBob is: ",lambda_nBob)
```

iii. Find d_Bob **(Private Key of Bob)**

We know e_Bob and λ(n_Bob),
And we know d_Bob mod λ(n_Bob) = 1
We can use the python function to compute the modular inverse
We can use the Extended Euclidean Algorithm to find it, whose runtime is logarithmic, therefore pretty efficient even for larger numbers.

```
# Brute Force Find d_Bob
d_Bob = pow(e_Bob, -1, lambda_nBob)
```

iv. Decrypted the message

Now that we have the private key (d_Bob) ***1249***
We can decrypt the message. For each number x in the encrypted sequence, calculate y = x^d_Bob mod n_Bob, and then transform that number y, into ASCII characters to get the final message.

```
Encrypted_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 = ''
for num in Encrypted_message:
    Decrypted += (chr(num**d_Bob % n_Bob))
print(Decrypted)
```

v. Code Results

```
RSA Begins:
p_Bob is 67
q_Bob is 83
λ_nBob is:  2706
d_Bob is 1249
Hey Bob. It's even worse than we thought! Your pal, Ali
ce. https://www.schneier.com/blog/archives/2022/04/airt
ags-are-used-for-stalking-far-more-than-previously-repo
rted.html
```

c. Explanation of how the message is *encoded:*

i. First of all, Alice starts with the message she wants to send

```
Hey Bob. It's even worse than we thought! Your pal, Ali
ce. https://www.schneier.com/blog/archives/2022/04/airt
ags-are-used-for-stalking-far-more-than-previously-repo
rted.html
```

ii. And then she transforms each character into its corresponding ASCII number, and put them into a list in order, which should be like this:

```
[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, 1
01, 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, 11
2, 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, 11
4, 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]
```

iii. And then Alice encrypts each of the numbers in the sequence, with the public key (e_Bob, n_Bob) she received from Bob. She replaces each number x in this sequence with x^e_Bob mod n_Bob.
For instance, for the first number 72, she would replace it with: 72^13 mod 5561= 1516. And then we'd end up with the original sequence of numbers which Eve intercepts.

d. Why the message encoding is insecure with larger integers:

This is because this **message encrypts one character at a time!**
Eve already obtained n_Bob, it is public, Eve also knows the mechanism of encryption, given the public key.
Suppose n_Bob is huge and it's close to impossible to factor it. All Eve needs to do is encrypt all common characters (there are only 26 letters in English), and match them to the encrypted message.
This way, Eve doesn't need the private key at all to decrypt the message.
Maybe grouping 2,3 or more letters together as one number would help a tiny bit, but with modern computational powers it'd also be easily broken.
The more secure way is to encrypt the whole message together.