```python
# Jack Huffman
# Diffie-Hellman/RSA Assignment
# CS338 w/ Jeff Ondich

g = 7
p = 61

# Alice sent Bob the number 30.
A = 30

# Bob sent Alice the number 17.
B = 17


# Try different values of a to find Alice's private key
a = 1
while True:
    if (g ** a) % p == A:
        break
    a += 1
# 41

b = 1
while True:
    if (g ** b) % p == B:
        break
    b += 1
# 23

print("Alice's private key (a):", a)
print("Bob's private key (b):", b)

aliceKey = (B ** a) % p
bobKey = (A ** b) % p

print("Alice computed value (k):", aliceKey)
print("Bob computed value (k):", bobKey)

# SUMMARY/ANSWERS TO QUESTIONS FOR DIFFIE-HELLMAN
```

```
# Figure out the shared secret agreed upon by Alice and Bob. This will be an integer.
# The shared secret is k = 6

# Show your work. Exactly how did you figure out the shared secret?
# To figure out the shared secret I used the following code to do a couple calculations. The first two while loops calculate potential secret numbers that alice and bob could have held
# which given the values for g = 7 and p = 61 could lead to the calculation of A = 30 and B = 17, those two numbers being a = 41 and b = 23. To check if these numbers work I raised,
# both of the sent values (A and B) to the computed secret values (a and b) then modded their results by p and checked if they gave the same result. They both gave 6 meaning that I have
# the correct secret value for both an the correct secret key (k = 6)

# Show precisely where in your process you would have failed if the integers involved were much larger.
# This process would be much more difficult in the case that we are dealing with much larger numbers as the loops I run would have to run for considerably longer. Additionally, in this case
# the first two values that were found that satisfied the clause ended up being the correct values, but its possible that the first value which meets that criteria is not the correct secret key.
# Given that the values we used were low this process of checking values was relatively quick but as the integer size increase so does the number of potential checks we need to make.











e_Bob = 13
```

```python
n_Bob = 5561

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]

def factorize(n):
    factors = []
    divisor = 2
    while n > 1:
        while n % divisor == 0:
            factors.append(divisor)
            n //= divisor
        divisor += 1
    return factors

factors = factorize(n_Bob)
if len(factors) == 2:
    p_Bob, q_Bob = factors
    print("p_Bob:", p_Bob)
    print("q_Bob:", q_Bob)
```

```
else:
    print("Factorization failed")


def modinv(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise Exception('The modular inverse does not exist')
    else:
        return x % m

def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = extended_gcd(b % a, a)
        return (g, y - (b // a) * x, x)


# Calculate Bob's private key (d_Bob)
phi_n_Bob = (p_Bob - 1) * (q_Bob - 1)  # Calculate phi(n_Bob) using Bob's
prime factors (p and q)
d_Bob = modinv(e_Bob, phi_n_Bob)  # Calculate the modular multiplicative
inverse

# Decryption
decrypted_message = []
for y in message:
    decrypted_value = pow(y, d_Bob, n_Bob)
    decrypted_message.append(decrypted_value)


# Convert the integers back to ASCII characters
plaintext_message = ''.join([chr(val) for val in decrypted_message])

# Print the decrypted message
print("Decrypted Message:", plaintext_message)

# SUMMARY/ANSWERS TO QUESTIONS FOR RSA
```

```
# The decrypted message reads as such "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"
# They way this was obtained is by using some python code to calculate bobs private key from his publicly available information and then use said private key to decrypt the information.
# This is done by using a factorization loop to generate a list of potential factors for n_bob (in this case n_bob = 5561). If there are exactly two factors then we know that these are the two prime factors for n
# and we can use those values to be able to calculate phi. In this case the two prime factors were 67 and 83. By doing the calculation (p_Bob - 1) * (q_Bob - 1) we get phi which we can then
# use in conjuction with e_Bob and a modular multiplicative inverse function to determine the value of the secret key for bob (called d_Bob). Now that we have the secret key decrypting the message is done simply
# by iterating over every number in the message and then taking that value and raising it to the power of d_Bob and finding modding it with n_Bob. Which then gives us our decrypted message as an integer which corresponds
# to an ascii value. Then it's just a matter of converting from ASCII to plaintext so that it is readable.

# Given that for this problem the numbers we were dealing iwth were realitvely small, the calculating of potential prime factors for the numbers was a relatively quick process which
# wasn't a computational issue. However, when the integeres get much bigger it becomes much more time and resource expensive to be able to calculate these factors as there are simply so many numbers
# to account for.

# This would still be considered an insecure implementation because even if large integers were used because it is encrypted simply by doing the RSA encryption on each ascii character which means that every ascii character
# of the same type would be encrypted the same way. This means that someone could possibly gain information as to the contents of the message by simply looking for patterns in the encrypted
# version of the text.
```