Viktor Chekhovoi is Eve in…

BEING EVE

1. Diffie-Hellman

   Alice and Bob agreed on g = 11 and p = 59. Since Alice sent Bob 57, she picked some number a so that $11^a \equiv 57\ mod(59)$. Since the numbers are small, I can write a short Python script that picks the a:

```
g = 11
p = 59
A = 57
a = 0
while g**a % p != A:
    a += 1

print(a)
```

   Almost instantly, the script printed the number 36, which I confirmed to be correct. I used a similar script to pick Bob's secret number b:

```
g = 11
p = 59
B = 44
b = 0
while g**b % p != B:
    b += 1

print(b)
```

   Again, the script only took seconds to run and it printed 15, which also fits the requirement for b. For both *a* and *b*, I was only able to brute-force the numbers because they were so small. According to https://justcryptography.com/diffie-hellman-key-exchange/, the recommended size for a Diffie-Hellman public key is 2048 bits, so a public key with the appropriate size would have taken an extremely long time to decode.

   Since I now have both *a* and *b*, I can compute the shared secret:

$$K \equiv g^{ab}\ mod\ p$$
$$K \equiv 11^{15*36}\ mod\ 59$$
$$K = 36$$

2. RSA

Since *n* is relatively small, I can easily factor it into *p* and *q*:

```
24  e_bob = 13
25  n_bob = 5561
26
27  p = 2
28
29  while n_bob % p != 0:
30      p += 1
31
32  q = n_bob // p
33
34  print(p, q)
```

The output is 67 and 83, so I know the values of *p* and *q*. I also know *e* because it's a part of the public key, so now I can pick *d*. I know that $d * e \equiv 1\ mod(\lambda(n))$. According to my notes and Wikipedia, $\lambda(n) = lcm(p-1, q-1) = lcm(66, 82)$. Using an online calculator, I found that $\lambda(n) = lcm(66, 82) = 2706$.

```
38  d = 1
39  lambda_n = 2706
40
41  while e_bob*d % lambda_n != 1:
42      d += 1
43
44  print(d)
```

The output was that $d = 1249$.

In this case, although *p* and *q* were small compared to the size of RSA keys used for serious encryption, it took much longer to pick the secret key than with the Diffie-Hellman example. This goes to show that if *p* and *q* were even bigger, it would be impossible to brute-force *d* in a reasonable amount of time.

Knowing *d*, I can decrypt the message. First, however, I need to figure out how it was encrypted by Alice. The encrypted message is split into separate elements, so it's reasonable to assume Alice somehow split up the message into blocks before encrypting. I'll decipher each number using the following script and determine the course of action depending on the result.

```
1  > ciphertext = [1516, 3860, 2891, 570, 3483, 4022, 3437, 299, ···
23  plaintext = []
24  d = 1249
25  n = 5561
26
27  for character in ciphertext:
28      plainAscii = character**d % n
29      plainChar = chr(plainAscii)
30      plaintext.append(plainChar)
31
32  print(''.join(plaintext))
33
```

The output is: “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.” The message is coherent, so it’s clear that Alice encrypted each character separately, maybe using a script like this (most of the plaintext message is cut off by the screenshot):

```
39  (e_Bob, n_Bob) = (13, 5561)
40  plaintext = "Hey Bob. It's even worse than we thought!
41  ciphertext = []
42  for char in plaintext:
43      cipher = ord(char)**e_Bob % n_Bob
44      ciphertext.append(cipher)
45
46  print(ciphertext)
47
```

However, Alice’s message could have been decrypted without even picking the keys, because she encrypted the message character-by-character. This means that I can analyze her message and determine which ASCII character corresponds to which number:

```
1   from collections import defaultdict
2
3 > ciphertext = [1516, 3860, 2891, 570, 3483, 4022, 3437, 299, ···
25
26  frequency = defaultdict(lambda: 0)
27
28  for character in ciphertext:
29      frequency[character] += 1
30
31  freq_list = list(frequency.items())
32  freq_list.sort(key=lambda x:x[1], reverse=True)
33
34  print(freq_list)
35
36
```

The output is a list of the numbers that appear in the message, sorted by frequency: [(3860, 15), (3075, 12), (570, 11), (3433, 11), (4022, 10), (653, 9), (2442, 9), (5077, 9), (1511, 8), (4951, 7), (2458, 6), (3455, 6), (299, 5), (4851, 5), (2187, 5), (3411, 4), (5139, 4), (3000, 4), (1106, 4), (482, 3), (3194, 3), (2875, 3), (2891, 2), (3437, 2), (3668, 2), (1165, 2), (2713, 2), (1516, 1), (3483, 1), (843, 1), (5450, 1), (4180, 1), (4169, 1), (4759, 1), (2863, 1), (3269, 1), (4063, 1), (3409, 1)]

Using a chart from https://www.englishlanguagefaqs.com/2013/04/what-are-most-used-letters-in-english.html, I can partially decipher the message. For example, the most common letter in English is ‘e’, and the most common number in the message is 3860. I can make a reasonable assumption that 3860 corresponds to ‘e’, and that’s correct. This is more challenging for characters that appear less frequently, but it still makes it possible to decrypt the message without getting *d*.

P. S. Alice, are you really his pal if you’re using encryption that’s this weak? That’s not a very friendly thing to do.