CS338 Spring 2022
McKenna Wirth and Antonia Ritter

BEING EVE

Diffie Hellman

- Alice and Bob agree on g = 11 and p = 59.
- Alice sent Bob the number 57.
- Bob sent Alice the number 44.

Following the instructions from the lab activity "Diffie Hellman and RSA by Hand," we determined that we need to find X and Y, both less than p, such that

$$\begin{aligned} B^X \bmod p &= 44^X \bmod 59 \\ &= 57^Y \bmod 59 \\ &= A^Y \bmod p \\ &= \text{secret key.} \end{aligned}$$

To solve this, we wrote the code found in `eve_diffie_hellman.py` in the submission folder. This code works by looping through every possible value of X and Y up to p, and checking whether they satisfy this equation.

Using this, we got $X = 2$, $Y = 25$, either of which can be used to compute the shared key (the equation above), which is $K = 48$.

Points of Failure in Decryption

This method won't work with big integers because the line of code

```
(B**i % p) == (A**j % p)
```

is slow on its own when the numbers involved are big, and has to be computed up to p^2 times. That's a lot of time and resources, and by the time you break it the information might be old anyway.

RSA

Here's Bob's public key:

$$(e_B, n_B) = (13, 5561).$$

To decrypt the message, we need Bob's private key, (d_B, n_B), so we need to figure out d_B. It was generated to satisfy the equation

$$e_B d_B \bmod \lambda(n_B) = 1,$$

which we also use to identify d_B. The first step is to factor $n_B = 5561$, which wolframalpha.com tells us is $67 * 83$. We also used it to find

$$\lambda(n_B) = lcm(67-1, 83-1) = 2706.$$

All that remained was to solve for d_B:

$$13 d_B \bmod 2706 = 1.$$

Again with wolframalpha, this implies $d_B = 2706k + 1249$ for any integer k, so we started with $k = 0$ and found $d_B = 1249$. If this d_B had not worked, we would have increased k until we found one that did.

Once we had the decryption key, we wrote the code in `eve_rsa.py`. The message was encrypted character-by-character, so for each character c we performed the operation $c^{d_B} \bmod n_B$. Then we had a list of integers, which we recognized as ASCII (a standardized way of representing characters as integers), which can be decoded using the python function `chr`. The resulting message 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

Points of Failure in Decryption

If the numbers involved were bigger, factoring n_B into two large primes would have been much more difficult. Solving for d_B could also take quite a while if it were very large, as we would have had to check many integers k. Each check would require decrypting, decoding, and recognizing the message, which is a non-trivial task. For example, someone would have to recognize if the message were in another language, encoded in an obscure way, etc.

Remaining Insecurity

However, no matter the size of the numbers involved in this case, it's still insecure because it's encoded one character at a time. It's essentially a fancy substitution cypher. For example, e = 3860. If you could figure out that it starts with "Hey," you could eventually figure out the rest.