Encryption:
Alice and Bob want to communicate state secrets over the internet. But **Evil Eve** is listening in. How can Alice and Bob communicate without Eve understanding? Alice and Bob need to use a code.

Examples of Codes:

1 Caesar cipher: To **encode**, replace a letter by the letter k letters past it. So for $k = 3$, $a \mapsto d$, $b \mapsto e$, ..., $y \mapsto b$, $z \mapsto c$. To **decode**, shift by $-k$ places. But there's a problem with this method of encryption: it's particularly easy to crack! Indeed, Eve just tries all 25 shifts.

2 Substitution cipher: Alice and Bob agree on a dictionary replacing each letter with a different letter (i.e. a bijection from [a-z] to [a-z]). Oliver noticed that a bijection is necessary, because to decode we'll need to use a reverse dictionary (i.e. the inverse of the bijection). Kimberly pointed out that the Caesar cipher is an example of this. To **encode** a message, simply pass it through the dictionary. If a↦f and b↦a and d↦g, then 'a bad dad' becomes 'f afg fgf'. We could be even trickier and also include spaces and punctuation as part of the alphabet, and then scramble it too. To **decode** we use the dictionary in reverse. Unfortunately, it turns out that this is still unsecure: Eve can use frequency analysis to crack it. For example, in English, 'e' is the most common letter, followed by 't', etc. We can also look at two letter pairs (e.g. 'th' or 'sh').

3 AES is a current method of symmetric encryption, that seems to be secure. Symmetric in this context means that Alice and Bob have to share a common key (just as in 1 and 2) that is used to both encode and decode.

For any symmetric cipher to work, Alice and Bob need to have the same key. How do they agree on a key? One method is the RSA algorithm, invented by Rivest, Shamir, and Adelman in 1977. The main idea is something like a 'locker' where only Bob has access, so Alice can communicate with Bob securely. The main idea here is that, unlike a symmetric cipher in which the key also plays the role of the lock, in RSA the key and the lock are distinct.

How RSA Works:
Step 0: Bob (secretly) generates two gigantic primes, p and q (here 'gigantic' means 100 digits each, say).
Step 1: Bob computes $N := pq$. He then publicly announces both N and a positive integer e (to be discussed below).

Step 2: Alice writes her message in the form of a positive integer, X (for example A=010, B=020, Z=260, etc; the method of turning text into a number is not kept secret from Eve). Alice then computes $Y \coloneqq X^e \pmod{N}$, and sends Y to Bob.
Step 3: Bob receives Y, and wants to find X. In other words, Bob needs to solve the congruence $X^e \equiv Y \pmod{N}$. We know how to do this! Bob simply finds $d \coloneqq 1/e \pmod{\varphi(N)}$, and then computes $Y^d \equiv X^{ed} \equiv X \pmod{N}$.

Mia asked the fundamental question: Why can Bob do this but not Eve?
Answer 1: No one knows how to solve $X^e \equiv Y \pmod{N}$ without finding d.
Answer 2: No one knows how to compute d without knowing $\varphi(N)$.
Answer 3: No one knows how to compute $\varphi(N)$ without knowing its prime factorization (i.e. p and q).
Since Bob is the only person who knows the factorization of N, Eve cannot figure out X.

Note that all the above answers, taken literally, are false: we can solve any of them by brute force. But for numbers with, say, 200 digits, this is computationally infeasible (i.e. would take on the order of a century, even with powerful computers).

Issues in implementing RSA:

1. Bob needs to be able to find enormous p and q, otherwise factorization will be possible. Even though it seems to be a hard problem to factor large numbers, it turns out to be possible to quickly check whether a given large number is prime. Moreover, primes are frequent enough that, after testing out a reasonably small number of 100-digit numbers, one will almost certainly stumble upon a prime. Thus, in practice, one can generate enormous primes fairly easily.

2. X needs to be less than N, otherwise Alice's message will be corrupted as soon as it's reduced (mod N). This is easy to arrange, though: Alice and Bob can publicly agree on the length of the key Alice will send to Bob, and then Bob simply chooses p and q large enough to guarantee that the key is smaller than N.

3. Bob's approach to decrypting Alice's message crucially uses that $X^{\varphi(N)} \equiv 1 \pmod{N}$. But we only know this in the case that $X \in \mathbb{Z}_N^\times$. What if X isn't relatively prime to N?

   Amazingly, RSA *still works* even if this happens. To see why this is, first observe that, since we can assume (thanks to point 2 above) that $X \not\equiv 0 \pmod{N}$, we must have exactly one of p or q divides X. Without loss of generality, say $p \mid X$ but $q \nmid X$. As

before, we have $Y^d \equiv X \cdot X^{k\varphi(N)} \pmod{N}$. But looking at this (mod q) we have

$$X^{k\varphi(N)} = X^{k(p-1)(q-1)} \equiv 1 \pmod{q}$$

by Fermat's Little Theorem. Thus we can write $X^{k\varphi(N)} = 1 + \ell q$, whence

$$Y^d \equiv X(1 + \ell q) \equiv X + X\ell q \equiv X \pmod{N},$$

since $p \mid X$. Thus we see that RSA still works even in the unlikely event that $X \notin \mathbb{Z}_N^\times$.

4. For the decryption to work we require e and $\varphi(N)$ to be coprime. But Bob knows $\varphi(N)$ and gets to pick e, so he can simply choose an e to satisfy this.

5. To prevent a brute-force attack by Eve, Bob must make sure that d isn't too small. (Otherwise, Eve could just raise Y to a bunch of powers and see whether any of them work as a key.) Thus to pick e, Bob starts by (secretly) picking a huge d relatively prime to $\varphi(N)$, and then publicly announces $e \coloneqq 1/d \pmod{\varphi(N)}$.

Thus we see that RSA is a fairly straightforward application of our algorithm for solving power congruences. The main take-away is this: if you can find some problem such that

- it's very hard to find a solution to the problem, but
- it's very easy to *verify* whether a proposed solution is legitimate,

then you can probably come up with some encryption scheme based on that. In the case of RSA, the problem we're solving is that of a power congruence (mod pq); given p and q it's easy to solve, but given only the product pq it seems to be difficult to solve.

Recall that last time, we had some trouble solving some power congruences, in particular $x^2 \equiv a \pmod{n}$; the issue is that no matter what power we raise x^2 to, the exponent will always be even, and will therefore never be congruent to 1 (mod $\varphi(n)$). So how do we solve quadratic congruences? This will be our next topic of exploration.