Cool Problems in Cryptography. Cryptography is full of cool problems that do not seem solvable but actually are. Here are some examples:

- **Secure communication over a public channel:** Suppose Alice and Bob want to communicate over a channel which is controlled by Eve. Can Alice nevertheless communicate a message that can be “understood” only by Bob? Can Bob be assured that Eve did not modify the message or just send a different one claiming it came from Alice?

- **Coin flipping over the telephone:** Suppose Alice’s roommate Bob calls her and asks her whose turn it is to do the dishes that night. Alice can’t remember and suggests they flip a coin for it. But Bob is out having fun and doesn’t want to come all the way home to do the coin flip, and he doesn’t trust Alice to do the coin flip honestly (neither does Bob trust Alice to do so). Is there a way for them to do a “fair” coin flip over the telephone?

- **Proving knowledge without revealing it:** Suppose you discovered a proof of an important theorem and want to convince your friend that you have done so, but you are worried she will steal your proof and claim that she found it instead. Can you convince your friend that you indeed discovered a proof without giving her the ability to claim it as her own?

Ad-hoc vs. Provable Security Design. You could probably come up with some ideas towards solving the above problems, and maybe even some candidate solutions. How do you know if your solution is “right” or not? This is not just a philosophical question. Cryptography is used all over the place, for example every time you log into a secure (https) website. Would you want to put a cryptographic protocol into everyone’s browser you are unsure about? If an attack is found, the consequences could be severe. Moreover, even if one attack is fixed, there is no guarantee that another one won’t be found in the future. Despite the drawbacks of such an ad-hoc approach to cryptographic protocol design, it was done this way for thousands of years and people did not think there was any alternative.

However, a revolutionary alternative approach was developed in the 1980s (although some of the ideas started with Shannon, and it was recently revealed that John Nash sent a prescient letter to the NSA in the 1950s describing some of the ideas as well), called **provable security**. Suppose we want to solve some cryptographic task, say private communication over a public channel. We first start by defining what algorithms a solution consists of, and what correctness requirements they should fulfill in order to serve their intended purpose in the absence of an adversary. In this case, we define a cryptosystem as \((K, E, D)\) where \(K\) is a key generation algorithm, \(E\) is an encryption algorithm, and \(D\) is a decryption algorithm. We require that for all messages \(m\), \(D(K, E(K, m)) = m\) for all \(K\) output by \(K\). Can you see how Alice and Bob would use these algorithms?

Next we define a **security model** for such a scheme can captures what they adversary is allowed to do in trying to attack the scheme and what constitutes a breach of security. We will see many examples of security models in the course. For example, in the case of encryption the adversary
could be given the encryption of a message and try to guess the message, or she could be the
encryption of many messages and try to guess some partial information about them NB: In any
security model, we always assume the scheme’s algorithms themselves are known by the adversary
(this is called Kerchoff’s Principle). Finally, we propose a candidate scheme (meaning, candidates
for the algorithms we require) and prove that no adversary can break it in our security model. An
important point here is that such security proofs are “computational” in nature. A typical security
theorem looks something like this:

**Theorem 1.** Cryptographic scheme $X$ above cannot be broken in security model $Y$ with probability
more than $\epsilon$ by adversaries running in time at most $t$, assuming computational problem $Z$ cannot
be solved with probability more than $\epsilon'$ by algorithms running in time $t'$.

In other words, security proofs do not give absolute assurance that a scheme can’t be broken (not
that they would anyway, since they are always relative to some security model), but rather they
say that it can’t be broken by resource-bounded adversaries assuming some underlying problem is
hard. Why is this?

**One-Time Pad and the Need for Computational Assumptions.** It turns out, if we want
security against unbounded adversaries and without making computational assumptions, we can’t
do too much. To illustrate this we consider a classical encryption scheme called the one-time pad.
First we define a notion of perfect secrecy due to Shannon.

**Definition 0.1.** A cryptosystem $(K, E, D)$ is perfectly secure if for all distributions $D$ on messages
and every message $g$ and every ciphertext $c$

$$\Pr [ g = m \mid E(K, m) = c ] = \Pr [ m = g ]$$

where the probability is over $K \leftarrow \mathcal{K}$ and $m \leftarrow \mathcal{D}$.

About notation: we use the notation $x \leftarrow A(\ldots)$ to denote that $x$ is assigned the output of
running randomized algorithm $A$ on the elided inputs with fresh random coins. If $A$ is deterministic
we drop the dollar sign. If $S$ is a finite set then $s \leftarrow S$ denotes that $s$ is assigned a random element
from $S$, and if $X$ is a random variable or distribution (on some set) then $x \leftarrow X$ denotes that $x$ is
assigned an element sampled according to this distribution.

A simple scheme achieves perfect security is the one-time pad (also called a Vernam cipher).
Let messages be $k$-bit string and define $K$ to consist of $k$-bit strings as well. Algorithm $K$ outputs
a random $K \in \{0, 1\}^k$. On inputs $K, m$ algorithm $E$ outputs $K \oplus m$ (where ‘$\oplus$’ denotes bit-wise
exclusive-or). Decryption is defined in the obvious way. To see that this scheme achieves perfect
security we can define a simpler security notion as follows.

**Definition 0.2.** A cryptosystem $(K, E, D)$ is Shannon secure if for all messages $m_0, m_1$ and cipher-
texts $c$

$$\Pr [ E(K, m_0) = c ] = \Pr [ E(K, m_1) = c ]$$

where the probability is over $K \leftarrow \mathcal{K}$.

Shannon proved that these two definitions are equivalent. The proof is not difficult and is a
good excursion. Moreover, it is a good exercise to prove that the one-time pad is Shannon secure.
(Showing that is perfectly secure directly is rather more complicated; the point of Shannon security
is that it is easier to work with yet equivalent to perfect security).

Finally, note that the scheme is very computationally efficient except that it requires keys as long as the messages. Shannon proved that this is inherent for any scheme meeting perfect security. But this is not practical, motivating the computational approach discussed above.