The definition of semantic security only requires that you can't distinguish messages of the *same size*.

The Public-Key Infrastructure

When I go to Facebook and type in my password, how do I know my password is only going to the real Facebook? We make certain assumptions:

- We assume that Firefox is behaving correctly (subsumes malware), i.e., is not sending information to other sites.
- HTTPS certification is working correctly.
- Facebook's URL is indeed facebook.com. (E.g., there's the *IDN homograph attack*, where we could have wikipedia.com actually contain a non-standard letter that just looks lot like wikipedia.com, e.g., w?kipedia.com.)

Are these reasonable? If we look at the first one, we clearly had to download Firefox at some point. This leads to more assumptions:

- We assume that Internet Explorer (which we used to download Firefox) is behaving correctly.
- HTTPS certification is working correctly.

Why does Firefox believe that FB is legit? FB presents a certificate signed by a certificate authority (CA), VeriSign in this case. VeriSign is the root of trust, because Firefox has hard-coded into the browser its trust of VeriSign. We can come up with a large chain of trust that describes who we need to trust and why. Some observations:

- There's deep recursion in this chain of trust, but the trust is not transitive. You need to trust *everyone* in the tree; it's insufficient to just trust someone that someone you trust trusts.
- The roots are often brands and small in number. Ideally, the user doesn't have to remember their URLs.
- MITM is not pose because of the authenticated key exchange and the use of the TLS/SSL protocol. In a way, cryptography removes the need for trusting the network–instead, you just need to trust the other parties.

NSA & a "rogue CA"

Here's how someone can MITM you:

1. Issue some rogue certificates for target sites.
2. Select users to target and install MITM boxes at their ISPs.
3. When the user sends a request to their desired site, you intercept the traffic and send back a signed response using a rogue certificate.

How can we detect this?

- Have the browser remember old certificates and alert the server if something's wrong ("key pinning").
- Have the server notice that lots of users are logging in from the same IP (or, better yet, the same device).

The conclusion is that reports of the NSA routinely circumventing SSL are probably unfounded.

What is the public-key infrastructure?

The basic structure is as follows:

- Infrastructure to create, distribute, validate, and revoke certificates (certificate authorities).
- A small number of roots hierarchically certify various entities.
- Clients trust roots, transitively follow chain of trust from server back to root.
- The standard is known as *X.509*.

The traditional view of the CA hierarchy is tat you have a root CA who verifies an intermediate CA who verifies a leaf CA who releases end-entity certificates. In real life, it's much more complicated; the graph can even have cycles.

Certificate

The certificate **binds** an entity to a public key. The certificate must be signed by some issuer (CA) and contains the identity of this issuer.

To obtain a certificate:

1. Server generates a key pair, signs public key and ID info with the private key (this proves that the server actually knows the private key). The server holds onto the private key and provides message authentication.

2. The CA verifies the server's signature by using the server's public key to verify itself.

3. The CA signs the server's public key with the CA key. This creates a binding.

4. The server verifies the key, ID, and the CA's signature.

5. The server proudly hangs new certificate on the wall.

The X.509 spec itself is thousands of pages long. SSL/TLS utilize X.509 for key distribution, then perform key exchange and encryption. The spec is so long, though, that it's impossible for mortals to get right. There are always flaws and the implementations rely on attacks and fixes.

In fact, it was found that almost all SSL clients except browsers and key SSL libraries are broken. This includes shopping cart libraries, instant messengers, mobile banking apps. They were actually broken in hilarious ways, like not checking that the URL is what they expect it to be.

Hard problems

There are three primary problems here:

- Naming and identity verification.
- Anchoring.
- Revocation.

Naming and identity verification

For any naming system, you want three things:

1. Uniqueness.

2. Human-memorable.

3. Decentralized.

This is known as **Zooko's triangle**, and the argument is that you can only get two of the three. For example, the human naming system is not unique, while domain names are not decentralized.

Domain validation is typically automated and email-based. But email is still not secure? This raises the attacker's bar but is not completely safe.

Revocation: Two hard problems

1. Authenticating the revocation request.
    - If not careful, it's easy DOS.
    - Can't ask for the old key.
2. Keeping clients up to date.
    - Offline model: keep a certificate revocation list.
    - Online model: online certificate status protocol (OCSP).