Alin Tomescu, CSE408
Thursday, April 14th, Lecture #21

More public key infrastructure

Transitive trust

As always C_{ta} is defined as a certificate from Trent to Alice, proving her identity to the world. This certificate also includes a lifetime.

$$C_{ta} = \left(\text{alice}, pk_a, lifetime, Sig_{sk_t}(\ldots)\right)$$

Alice obtains a certificate (offline) C_{ta} from Trent.

Trent, referred to as $Trent_0$, can delegate his authority to other CA's like $Trent_1$ and $Trent_2$

$Trent_0$ issues $C_{t_0t_1}$ to $Trent_1$ and $C_{t_0t_2}$ to Trent2 (out of band).

And so now Alice can get a certificate C_{t_1a} from $Trent_1$ who will also send his certificate $C_{t_0t_1}$. Alice sends these to Bob. Since Bob trusts $Trent_0$ and $Trent_1$ has a certificate from $Trent_0$ proving his identity and trustworthiness, Bob will trust $Trent_1$.

The format for a certificate from one Trent to another Trent is:

$$C_{Trent_0Trent_1} = \left(\text{Trent}_1, pk_{Trent_1}, 0/1, Sig_{Trent_0}(\ldots)\right)$$

0 would mean $Trent_1$ should not be necessarily trusted, 1 means he should be trusted.

Secure Sockets Layer PKI

Identifier = www.amazon.com

In SSL there are about 650 $Trent_0$'s. These are Certificate Authorities (CAs) such as VeriSign.

- all of their public keys are preloaded into your browser

SSL doesn't enable you to easily customize who you trust. A global trust decision is made for you: you trust all the CAs.

Extended validation certificates (green bar in your browser) – extra checking that CAs are doing additional work when issuing certificates.

Certificate revocation

CA's can issue lists of bad certificates. Unfortunately, they can't force you to download that list. Most web browsers don't get the list automatically.

Two ideas proposed for fixing the SSL PKI

1. Multiple certificates (no reason for Alice to just get one certificate from $Trent_1$, she could get more certificates from a lot of CAs and if Bob trusts one of them then it's good)
2. Perspectives
    a. new trust

Alin Tomescu, CSE408
Thursday, April 14th, Lecture #21

DNSsec

DNS is distributed, fast, lightweight and cached.

There's no authentication in DNS. An attacker can tell you made a request for *amazon.com* and can do a MITM responding with a bad IP address. Your computer will connect to that illegitimate website thinking it's on the legitimate one.

Resolver asks .com NS for the *amazon* NS, he gets $r = IP\ of\ amazon\ NS, pk_{amazon\ NS}, Sig_{sk_{.com}}(\dots)$

How does he get the public key for the .com NS? He gets it from the previous query to the top-level domain (TLD) NS. How does he get the public key of the TLD NS? There are only 13 of top-level domain servers, so he can remember them.