Key Distribution Problems

Public key cryptography depends on the distribution of keys. Everyone having to store everyone else's keys scales badly. One way to solve this problem is by having centralised databases of keys. These are called *Key Distribution Centers* or (KDC).

Key Distribution Centers

Key Distribution Centers have the following properties

- Based on Secret-Key Systems
- Every user shares K_A with the KDC
- When A wants to talk to B, KDC generates $E_{K_A}(K)$ for A, $E_{K_B}(K)$ for B.
- All users must trust KDC completely
- If the KDC is down or compromised, the entire network is fåkked.

Certification Authorities

We can replace the previous example with the following protocol

- Key K_{AB} is replaced by pair (sk_B, pk_B). Here pk_B is public.
- A sends session key $E_{pk_B}(K)$ to B.

How do we distribute these? Assume we have a *Certification Authority* CA with the pair (sk_{CA}, pk_{CA}). Everyone then gets a copy of pk_{CA}. Then begins the hard part of authenticating yourself with the CA. This might even be done in person. The CA then issues cert ID_A, and pk_A and then the signature $S_{sk_{CA}}(ID_A, pk_A)$. Now everyone can check to see if the public key is legitimate.

Certificate Chains: Or Who Verifies The Verifiers

If two people do not have certificates issued from the same CA, they can use what's called Certificate Chains to solve this problem. If CA's certifiy each other's public keys. If user A has CA_1 and B has CA_2. If A receives $Cert_{CA_2}(B, pk)$ then if he also gets $Cert_{CA_1}(CA_2, pk_{CA_2})$ then A can verify that certificate. A chain has the from

$$Cert_{CA_1}(B, pk_B), Cert_{CA_2}(CA_1, pk_{CA_1}).....$$

It should be noted that this requires one to trust the initial part of the chain in the first place.

Who they are? Biometric Security

Biometric Security is when a system makes use of fingerprints, voice, facial recognition and other biological factors unique to an individual to identify them. This has a privacy downsides, and if the information is leaked, it could still be used to attack systems.

Something They Have: Hardware Security

Hardware may have their own keys, such as for RSA encryption, which is only accessible through the CPU. It is also possible to make *tamper evident* hardware, which can tell if it has been tampered with.

Two-factor authentication: Two-factor authentication works by demanding that the user authenticate themselves not only with a password, but also by possession of a piece of hardware, such as a smartphone. This means that even if the password is compromised, it is still possible to deny intruders access.

Password Security

Passwords identity users from what they know. And generally we want to protect passwords from the following 4 attacks

- The adversary can guess and verify their guesses
- The password is transmitted in an insecure manner, and the adversary can intercept it
- The password is stored in an insecure manner, and can be stolen from the user
- The password can be stolen from the verifier

Password security

The previously listed problems can be remedied with the following: **Guessing passwords**: This can be remedied with password rules. However, the more password rules, the fewer combinations people have to guess. Having ample length and characters (but mostly length) and a database of vulnerable passwords that are forbidden seems to be the best policy **Insecure Transmission**: Use encryption **Insecure storage by the user**: Could be helped by using other forms of authentication, such as two-factor authentication. **Stealing from verifier**: Salt and hash passwords, store them encrypted and use good security policies.