

## Introduction To Authentication Systems

### Password-based Authentication

*It's not who you know. It's what you know*

- On-line Password attack:

Easy to defend, limit and slow down the number of guesses.

- Off-line Password attack:

Capture a quantity X derived from the password & take your time to guess (e.g., use a **dictionary**) the password that produces X.

#### Storing User Passwords:

- Store password **hashes**.
- **Encrypt** the password database and safeguard the encryption key.
- **Combination:** Encrypt the database of hashed passwords.

### Address-based Authentication

*It's not what you know. It's where you are*

#### In Unix implementations:

- /etc/hosts.equiv:

Contains a list of computers that have identical user accounts. Allow users on these hosts to login (rsh) without providing passwords.

- **\$HOME/.rhosts:**

e.g., if at **cs.unc.edu** file **/home/wahab/.rhosts** contains:

```
isis.cs.odu.edu wahab  
isis.cs.odu.edu wild
```

allows **wahab** and **wild** to login as **wahab** to any host at **cs.unc.edu** from host **isis.cs.odu.edu** (**wild** does not know **wahab's** password).  
(see `% man hosts.equiv` for details).

### Network address Impersonation:

Generally, it is not difficult for Trudy to claim Alice's address as the source address, but it is more difficult for Trudy to receive messages addressed to Alice's network address.

In IP protocol, Trudy may use *source routing* to achieve that by sending a message with source route:

```
<Alice, Trudy, Dist>
```

and **Dist** will reply with source route:

```
<Dist, Trudy, Alice>
```

Thus Trudy can get the reply!

### Passwords as Cryptographic Keys:

To convert a text string that is memorable by a human e.g., **oducsc**, into **cryptographic secret key**, e.g., DES key, do a cryptographic hash of the password and take the 56 bits of the result.

## Eavesdropping & Server Database Reading

### Public Key Cryptography:

Makes it easy to perform authentication that *both*:

- Secure against *eavesdropping* &
- Protects against an intruder *reading the server database*.

**Alice**

**Bob**

I am Alice -----> get Alice public key: e  
R <----- R (random)  
sign R:  $X = [R]_d$  -----> verify:  $R = [X]_e$

### Password Hashing:

**Alice**

**Bob**

I am Alice, oducsc ---> knows hash of Alice Pwd, **h**  
check that:  $h = MD(oducsc)$

**Problem:** eavesdropping.

### Shared Secret:

**Alice**

**Bob**

I am Alice -----> get Alice secret key: K  
R <----- R (random)  
encrypt R:  $X = K\{R\}$  -----> decrypt:  $R = K[X]$

**Problem:** reading the server database.

## Trusted Intermediaries

*If we have  $N$  nodes:*

If each node keeps  $N-1$  secrets,  
then adding a new node involves adding  $N$  new secrets,  
one at each node.

**Clearly not practical for large  $N$ .**

## KDC (Key Distribution Center):

**KDC** knows  $N$  keys, one for each node.  
Adding a new node involves only adding one key at KDC.  
If Alice like to talk to Bob:

**Alice**

**KDC**

**Bob**

Need to talk to Bob --->

generate andom **R**

$R = K_A[X]$  <---  $X = K_A\{R\}$   
 $Y = K_B\{R\}$  --->  $R = K_B[Y]$

$C1 = R\{M1\}$  ----->  $M1 = R\{C1\}$   
 $M2 = R\{C2\}$  <-----  $C2 = R\{M2\}$

## Disadvantages of KDC:

- If compromised, all Keys are compromised.
- Single point of failure
- Performance bottleneck.

## CA (Certificate Authority):

- Each node keeps its *private key*.
- The CA **certifies** (*sign*) that the public key belong to the node and everyone trust the CA that he checked this fact for each node.
- All public key certificates may be kept in one place or each node keeps its own certificate and presents it to whoever asks for it.
- Certificates *expire* after a reasonable period (e.g., 1 year) but can be *revoked* at any time and the CA periodically publish a **CRL** (certificate revocation list) that contains all the revoked certificates.
- Clients should check the latest CRL before trusting a certificate.

## Session Key Establishment

It is a good idea to generate a *separate key for each session* to use for encryption/decryption of session data following the session authentication phase.

### Why?

- Keys are kind of "wear out" if used a lot!  
The availability of more cipher text, the more likely an intruder may find the key.
- Prevent **replay** and decryption of previously recorded message.

## Delegation

*It's not who you are. It's who you're working for*

Sometime it is necessary to have some entity act on your behave.  
One possible means of allowing this is to give your password to this entity.  
This is not usually a good idea (**please never do that!** oducsc).

The best mechanism to achieve that is

*delegation* (or *authentication forwarding*).

Generate a special message, signed by you (using public key cryptography, or through the use of KDC), specifying:

- To *whom* you are delegating the rights,
- *Which* rights are being delegated &
- For *how long*.

---

## Authentication of People

User Authentication can be achieved using:

- What you **know**: e.g., password.
- What you **have**: e.g., a Physical key.
- What you **are**: e.g., Biometrics such as voice/face/eye/fingerprint.

## Passwords

### Problems:

- **Eavesdropping.**
- **Read** stored file.
- Easy to **guess** on-line.

- Easy to **crack** off-line.
- Users may **write it down**.

## On-Line Password Guessing

### Odd cases:

Some banks set passwords as the last 4 digits of SSN.  
 Some drivers licenses uses SSN as the license number.  
 When you write a check at a store, the clerk usually writes your driver's license number on the check.  
 Thus the clerk may get all the information for your bank account!

### Helpful Tips:

- Set limit on the **number of trials**.
- Process incorrect passwords **s l o w l y**
- **Report** to users of unsuccessful attempts.
- Assign users an easy to **pronounce** strings as passwords.
- Do not let users choose **easy-to-guess** passwords.
- Force users to **change** passwords frequently and prevent them from using old ones.

## Off-Line Password Guessing

Obtaining a hash of a password  $h$ ,  
 an attacker can guess the password  $w$  and  
 checks to see if  $h = MD(w)$ .

If some one obtains a file  $F$  containing the hashes of many passwords,  
 e.g., **/etc/passwd** he can perform a **dictionary attack**:

```

for each word  $w$  in dictionary  $D$  do
  Compute  $h = MD(w)$ 
  for each  $e$  in  $F$  do
    if  $e = h$  then  $w$  as a password
  done
done

```

**The number of performed hashes is:  $|D|$**

Storing a random number **s** (**salt**) with  $e = MD(w|s)$  makes it harder for a dictionary attack:

```
for each entry  $\langle s, e \rangle$  in F do  
  for each word w in the dictionary D do  
    Compute h = MD(w|s)  
    if e = h then w as a password  
  done  
done
```

**The number of performed hashes is:  $|D| \cdot |F|$**

## How long should a password be?

### To protect against **on-line attack**:

**short** password is fine.

E.g., ATM systems have 4 digits (10,000 different PIDs), This it is OK since you only have 3 guesses before rejecting/capturing your card.

### To protect against **off-line attack**:

**64** bits of randomness makes the number of trials  $2^{64}$  which is considered computationally hard:

- In decimal this is about **20 digits** to remember.
- If we select **random characters** we need **11 characters**.  
(upper case, lower case, digits, punctuations)
- If we generate **pronounceable** passwords we need **16 characters**.  
(case-insensitive and every third char is one of the 6 vowels)

## Eavesdropping

- **Low tech**: e.g., watch someone type a password.

**Protection:** use *shift/control* chars, and don't display the typed chars.

➤ **High-Tech:** e.g., wire-tapping, software-based keystroke logging.

**Protection:** use *one-time password* list (use new one each time) & *numbered list of passwords* (system asks for one at random).

## **Passwords & Careless Users**

### **Professor:**

**Q:** "Are there any advantages of passwords over biometric devices?"

### **Student:**

**A:** "When you want to let someone use your account, with password you just give it them, while with a biometric device you have to go with them until they are logged in!"

Users should be educated of the importance of security.

### **General Tips:**

- Do not exchange passwords using email.
- Use different passwords on different systems or accounts.
- Change your password frequently.
- Abort Login Trojan Horses (e.g., type Alt-Ctrl-Del).

### **Initial Password Distribution:**

**One popular scheme:** generate a **pre-expired** random password and hand it to the user. It must be changed as part of the first login process.

### **Authentication Tokens:**

Physical devices that a person carries around and uses in authentication. People tend to be less willing to "loan" a token to a friend than to share a

password.

**Example:** *Cryptographic calculator*, encrypt current time and display the result. The user types this number in place of a password. The computer does the same (takes into consideration that clocks drift).

### **Physical Access:**

The location from which access is requested can be part of the authentication process. For Example, many bank transactions can only be initiated at teller's terminals inside the bank.

### **Biometrics:**

Measure physical characteristics and match them against a profile.

#### ***Examples:***

- Retinal scanner.
- Fingerprint readers.
- Face recognition.
- Iris scanner.
- Voiceprints.
- Keystroke timing.
- Signatures.