

# Computer and Network Security

## CMSC 414 Fall 2008

Udaya Shankar  
shankar@cs.umd.edu

### AUTHENTICATION

October 17, 2008

## Authentication Overview (NS chapter 9)

### Scenario:

- Large set of principals attached to an open channel (eg, Internet).
- Each principal repeatedly
  - attempts to initiate a connection (or session) with a specified principal;
  - upon successful connection establishment, exchanges messages
  - closes the connection.

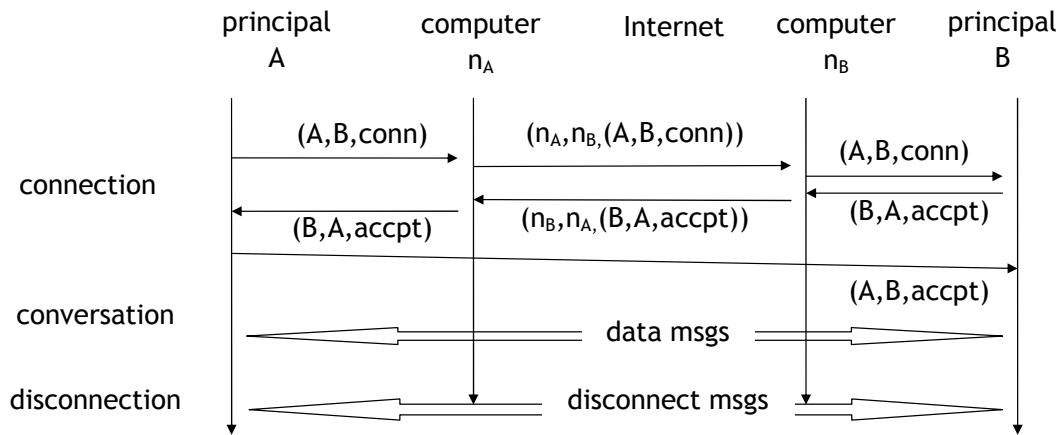
### Authentication is about ensuring that:

- When a principal A assumes it has connected to a principal B, A is indeed exchanging messages with B, and not some attacker C.
- When principal A assumes confidentiality/integrity of the message exchange, this is indeed the case.

### Principal can be a human or an executing computer program

- Program principals can use high-quality secrets (eg, from space of  $2^{64}$ )
- Human principals are restricted to low-quality secrets (eg, space of  $2^{32}$ ) and cannot do cryptographic operations.
- When we say a program principal A **assumes** it is connected to B, we mean that A's program's variables indicate that A is connected to B.

## Typical authentication scenario



### Achieves following inspite of attacks

- Connection establishment:
  - A authenticates B: (B,A,accpt) sent by B in response to (A,B,conn)
  - B authenticates A: (A,B,accpt) . . . A . . . . . (B,A,conn)
  - Simultaneously establish a shared secret (session key) for conversation
- Conversation: encryption / MAC / nothing (eg, if network is secure)
- Disconnection: A and B close their connection and forget any session key
- Same as traditional (eg, TCP) connection management except for attacks

## Attacks

- An authentication protocol must identify the attacks it is supposed to handle
- An authentication attack cannot handle all attacks, eg,
  - Overrun (take over) a human principal.
  - Overrun main memory while program principal is doing login authentication

### Network-based attacks (in order of increasing difficulty)

- Spoofing messages:
  - C at  $n_c$  sends messages with sender network address  $n_A$
- Eavesdropping: observing messages in the channel.
  - Very easy in WLANs and LANs because of their broadcast nature
  - Not easy in wired point-to-point links
    - Tap router ports
    - Compromise route computation algorithm
- Intercepting messages, changing them, resending them.
  - Easy in WLANs and LANs because of their broadcast nature
  - Not easy in point-to-point (but possible)

### End-host based attacks (in order of increasing difficulty)

- Principal C pretends to be principal A on computer  $n_A$  (eg, public workstation)
  - online password guessing
- Read data on hard disk (or back up tapes) of  $n_A$  or A
  - Obtain old keys (encrypted or plaintext) password files, ...
  - Obtain current keys (encrypted or plaintext) password files, ...
  - offline password guessing on encrypted passwords
- Overrun computer  $n_A$ 
  - while A is not at  $n_A$
  - while A is at  $n_A$
- Read data in memory of  $n_A$  while A is executing (unlikely)
- Overrun a program principal
  - mail client, web browser

### Some attacks may span both classes

- Overrun network servers (files, time-of-day clocks, authentication servers, ...)

### Can sequentially mount attacks of different classes

- Eg, record encrypted conversation; much later learn session key

### Vulnerability to attacks is usually associated with a time window

- Session key should be forgotten after session
- Principal's master key should be used for login only and forgotten after login

## Three types of authentication

- **Password-based authentication**
  - Authenticating oneself by showing a secret password to the remote peer (and to the network)
  - Always vulnerable to eavesdropping attack and on-line password guessing
- **Address-based authentication**
  - authenticating oneself by using a physically-secured terminal/computer
- **Cryptography-based authentication**
  - authenticating oneself by showing evidence of a secret key to the remote peer (and to the network)
  - Does not expose the secret itself to the peer (or to the network)

## Password-based authentication

- A authenticates itself by supplying a password.
- Always vulnerable to eavesdropping attack and on-line password guessing

### Approach 1:

A (passwd $pw_A$ )	$n_A$	channel	$n_B$	B (passwd file with $[X, pw_X]$ for each X)
<ul style="list-style-type: none"> <li>• enter [A, B, <math>pw_A</math>]</li> </ul>				
<ul style="list-style-type: none"> <li>• send [<math>n_A, n_B, A, B, pw_A</math>] to <math>n_B</math></li> </ul>				
<ul style="list-style-type: none"> <li>• check rcvd [A, <math>pw_A</math>] against passwd file</li> <li>• match authenticates A; msg from <math>n_A</math> until logout assumed to be from A</li> </ul>				

- Vulnerable to eavesdropping and to on-line password guessing
  - Defense against latter: limit number of successive failed attempts
- Vulnerable to exposure of password file (overrun of  $n_B$  or B)

### Approach 2:

- Like approach 1 except B's password file has entries  $(X, \text{hash}(pw_X))$  for each X

A (passwd $pw_A$ )	$n_A$	channel	$n_B$	B (passwd file with $[X, \text{hash}(pw_X)]$ for each X)
<ul style="list-style-type: none"> <li>• enter [A, B, <math>pw_A</math>]</li> </ul>				
<ul style="list-style-type: none"> <li>• send [<math>n_A, n_B, A, B, pw_A</math>] to <math>n_B</math></li> </ul>				
<ul style="list-style-type: none"> <li>• check <math>\text{hash}(\text{rcvd } pw_A)</math> against passwd file entry for A</li> <li>• match authenticates A</li> </ul>				

- Vulnerable to eavesdropping and to on-line password guessing (as before)
- Vulnerable to password file exposure but requires **off-line** password guessing
  - Defense 1: store  $(X, \text{salt}, \text{hash}(pw_X, \text{salt}))$
  - Defense 2: store  $(X, \text{encrypt}_K(pw_X))$  where K is high-quality key maintained only in B's memory and not hard disk (i.e., manually entered when B is activated).

## Password-based authentication (contd)

### Handling situation where A may interact with many servers

- Store A's password in every server that A may access.
  - Inconvenient for handling changes to password.
  - All password files need to be protected well.
- Store A's password in a special **authentication storage node**.
  - Server authenticates A by getting A's password from this node (and presumably forgetting it after authenticating A).
  - Disadvantage: performance bottleneck.
  - Advantage: single node to protect
- Store A's password in a special **authentication facilitator node**.
  - Server authenticates A by supplying A's password to this node and getting yes/no reply.
  - Advantage: single node to protect

## Address-based authentication

- A uses only a special set of computers
- A is authenticated by the address (network, link level, etc) of its computer.
- Valid if
  - Access to special computers is well-guarded
  - Network is protected against spoofing/interception of messages
- Examples:
  - Unix: os-wide /etc/hosts.equiv file, per-user .rhosts file.
  - VMS: PROXY database
  - Early main-frame machines accessed by dumb terminals.
  - Operator console on many workstations (eg, single-user mode in Linux)

## Cryptographic authentication

- A authenticates itself to B by performing a cryptographic operation on a quantity composed of a part supplied by B and a secret shared by A and B.
- Because operation is cryptographic, the secret is not disclosed by eavesdropping.

### Limitations if A is human

- A can only remember low-quality secret, ie, password, not key.
- A cannot do cryptographic operations.
- So A inputs password into computer  $n_A$  which converts password to key. Hence vulnerable to overrun of  $n_A$ .

### Approach 1 to transform password to key

- Obtain key by (say) hashing password (and, for DES, taking specified 56 bits).
- Ok for secret-key encryption, where any 56-bit string is a valid key.
- Generally not used for public key crypto, where keys have certain constraints. One way to do this for RSA:
  - Use password as seed to specified pseudo-random number generator, and choose first two primes generated.
  - Can reduce computational cost by having user remember and input more info, eg, indices  $j_1$  and  $j_2$  of the two primes in pseudo-random sequence. (Text says  $j_1$  and  $j_2$  need not be secret, but they do.)

### Approach 2 to transform password to key (generally used for public key pair)

- Use password to decrypt a high-quality key kept in a directory service.
  - Let  $K_A$  be A's master key.
  - Let  $K_{Apw}$  be the (low-quality) key obtained from A's password (eg, by hashing).
  - Directory service stores  $K_{Apw}\{K_A\}$  (ie,  $K_A$  encrypted by  $K_{Apw}$ ).
  - Computer  $n_A$  gets  $(A, K_{Apw}\{K_A\})$  from directory service,  $K_{Apw}$  from A's password, and decrypts to get  $K_A$
- Is this vulnerable to off-line (dictionary) attack?
  - Attacker guesses candidate password  $p$ .
  - Attacker obtains candidate master key  $X_A = \text{decrypt}(\text{hash}(p), K_{Apw}(K_A))$ . But cannot decide whether  $X_A$  is correct because  $K_A$  has no structure. Note: in RSA, encrypt  $[d]$ , not  $[d,n]$  (because latter has structure).
  - But it is vulnerable with a bit more work in some cases, eg,
    - If A uses a session key encrypted with  $K_A$ , use  $X_A$  to obtain candidate session key, and check if it can decrypt conversation.
    - If A's signature on a document produced using  $K_A$  is available, check if  $X_A$  matches verify A's signature on an available document

## Protecting against eavesdropping and server passwd file exposure (spfe)

### Easy with public key crypto

- A has private key.
- B stores A's public key (so exposing B's database does no damage).
- Authentication:
  - B sends a random value to A
  - A encrypts using A's private key and sends back
  - B checks received value using A's public key

### Handling spfe (but not eavesdropping) with hash/secret-key crypto

- B stores hash of A's password
- Authentication:
  - A sends password to B
  - B compares hash of received password with stored hash

### Handling eavesdropping (but not spfe) with secret-key crypto

- A and B share a secret  $K_{AB}$  (eg, A's password).
- Authentication:
  - A sends (A, login) to B
  - B sends random number R to A
  - A responds with  $K_{AB}\{R\}$

### Handling both with secret-key crypto

- Lamport hash scheme (later)

## Scaling to network of N principals

- Straightforward approach:
  - Distinct key for every pair of principals.
  - Not scalable:
    - $N^2$  storage cost at each node
    - N cost for adding new principal
- Use hierarchy of trusted intermediaries
  - **KDC (key distribution center)** in secret-key crypto
  - **CA (certification authority)** in public-key crypto

## KDC (for shared-secret keys) in single-domain case

- KDC shares a key  $K_X$  for every principal  $X$  of its domain
- When  $A$  wants to talk to  $B$ :

A	KDC	B
send [A, B, conn] to KDC	generate session key $K_{AB}$ generate $\text{tk}_{AB} = [K_B\{A, B, K_{AB}\}]$ (ticket) send [ $K_A\{A, B, K_{AB}\}$ , $\text{tk}_{AB}$ ] to A	
send [A, B, conn, $\text{tk}_{AB}$ ] to B		decrypt $\text{tk}_{AB}$ and get $K_{AB}$
< --- authentication between A and B using $K_{AB}$ ---- >		

- Advantages:
  - Adding new principal:  $O(1)$  interaction between principal and KDC
  - Revocation of principal:  $O(1)$ , deactivate principal's master key at KDC
- Disadvantages:
  - KDC can impersonate anyone to anyone.  
KDC compromise makes the whole network vulnerable.
  - KDC failure means no new sessions can be started.
  - KDC can be a performance bottleneck.
  - Last two can be alleviate by having KDC replicas, but
    - need to protect all replicas
    - when a principal's master key is changed, need to sync replicas

## CA (for public keys) in single-domain case

- Each principal has a public key pair.  
Remembers its own private key and CA's public key.
- CA generates **certificate** (signed public key) for each principal  $X$ :
  - $[(X, \text{pubkey}_X, \text{expdate}), \text{privkey}_{CA}\{(X, \text{pubkey}_X, \text{expdate})\}]$ .
- Certificates are publicly disseminated (e.g., at directory services).
- $A$  authenticates  $B$  as follows (ignoring certificate revocation):
  - Obtain certificate for  $B$  from anywhere, typically from  $B$ .
  - If certificate not expired and signature verifies (using CA's public key), then  $A$  has  $B$ 's public key.
  - $A$  sends challenge and expects challenge encrypted by  $B$ 's private key, after which  $A$  and  $B$  settle on a session key.
- Advantages
  - CA does not need to be on-line or networked, so can be more secure.
  - CA crash does not stop new sessions from starting until expiration date.
  - Certificates need not be secured (except for deletion of certificates).
  - Compromised CA cannot decrypt conversations (unlike KDC). But it can serve false public keys and thus impersonate any principal.

## CA (for public keys) in single-domain case (cont)

- Disadvantages
  - Certificate revocation is more complex than in KDC.
  - CA periodically (eg, hourly) issues CRL (Certificate Revocation List)
    - signed {issue time, list of certificates revoked at issue time}
- A authenticates B (in presence of CRL) by obtaining (typically from B)
  - a certificate for B that has not expired (as above), and
  - a CRL that does not have B and was issued sufficiently recently, eg, at the start of the current period.
  - A sends a challenge and awaits challenge encrypted by B's private key, after which A and B settle on a session key.
- X.509 format for certificate and CRL
  - Certificate = [user name, user public key, expiration time, serial number, CA's signature on entire contents of certificate]
  - CRL = [ issue time, list of serial numers of unexpired revoked certificates]

## KDCs for multi-domain case

**Case 1:** A in domain (with KDC X) wants to talk to B in domain (with KDC Y), and X and Y share a key, say  $K_{X-Y}$ .

A	X (KDC of $D_1$ )	Y (KDC of $D_2$ )	B
send [A, B in $D_2$ , conn] to X			
	generate session key $K_{A-Y}$ generate $\text{tk}_{A-Y} = [K_{X-Y}\{A, X, K_{A-Y}\}]$ send [ $K_A\{A, Y, K_{A-Y}\}$ , $\text{tk}_{A-Y}$ ] to A		
send [A, B, conn, $\text{tk}_{A-Y}$ ] to Y			
		generate session key $K_{A-B}$ generate $\text{tk}_{A-B} = [K_{B-Y}\{A, B, K_{A-B}\}]$ send [ $K_{A-Y}\{A, B, K_{A-B}\}$ , $\text{tk}_{A-B}$ ] to A	
send [A, B, $K_{A-B}\{A, B, \text{conn}\}$ , $\text{tk}_{A-B}$ ] to B			

## KDCs for multi-domain case (cont.)

### Case 2: KDCs chain from source to destination

- In a large internetwork with many domains, unlikely that every two domains will have a shared key.
- But if there is a sequence of domains  $D_1, D_2, \dots, D_N$  such that for every  $i$ , KDC of  $D_i$  and KDC of  $D_{i+1}$  have a shared key then A of  $D_1$  can securely obtain a session key to talk to B of  $D_N$ :
  - Let  $X_i$  be the KDC of  $D_i$
  - A talks to  $X_1$  and gets [session-key, ticket<sub>A-X2</sub>] to talk to  $X_2$
  - A talks to  $X_2$  and gets [session-key, ticket<sub>A-X3</sub>] to talk to  $X_3$
  - and so on until
  - A talks to  $X_N$  and gets [session-key, ticket<sub>A-B</sub>] to talk to B
- How does A get the sequence  $X_1, X_2, \dots, X_N$ .
  - Static hierarchy with additional links (perhaps cached) for efficiency.
  - Good if A also passes along the sequence of domains to be traversed, so that B can see whether it trusts every KDC on the chain.

## CAs for multi-domain case

**Case 1:** A in domain with CA X wants to talk to B in domain with CA Y, and X and Y have certificates for each other.

A	X directory service	Y directory service	B
<ul style="list-style-type: none"><li>• Gets from X's directory service a certificate for Y signed by X; A can verify certificate because A has X's public key; so A now has Y's public key.</li><li>• Gets from Y's directory service a certificate for B signed by Y; A can verify certificate because A now has Y's public key; so A now has B's public key</li><li>• A can now send messages to B encrypted with B's public key</li></ul>			

## Case 2: CA chain from source to destination

- In a large internetwork with many domains, unlikely that the CAs of every two domains will have a certificate for each other.
- But if there is a sequence of domains  $D_1, D_2, \dots, D_N$  such that for every  $i$ , directory services of  $D_i$  and  $D_{i+1}$  have certificates for each other signed by their CA's  
then A of  $D_1$  can securely obtain the public key of B of  $D_2$  by iterating:
  - Let  $X_i$  be the CA of  $D_i$
  - A gets certificate for  $X_2$  signed by  $X_1$
  - A gets certificate for  $X_3$  signed by  $X_2$
  - and so on until
  - A gets certificate for  $X_N$  signed by  $X_{N-1}$
  - A gets certificate for B signed by  $X_N$

## Session key (if used)

- Should be different from long-term shared key used for authentication
  - so long-term key does not “wear out” (off-line crypto attack)
- Should be unique for each session
  - If compromised, only affects data sent in that session.
  - Can be given to relatively untrusted software
- Session key should be forgotten after session ends

## Delegation or authentication forwarding

- If A, when logged into B, wants to access C (eg, printer), then B needs to authenticate itself as A to C.
  - A can log into C explicitly (too much trouble)
  - A can give B its password (too risky)
  - A can give B a ticket (called **delegation** or **authentication forwarding**) with
    - types of access allowed by B (eg, A's print queue)
    - expiry time (typically short)

### Constraints when authenticating human:

- Can only remember **low-quality** secret (eg, 10 letter “pronounceable” password).
- Cannot perform cryptographic operations.

### Human authentication based on one or more of

- What you know: password
- What you have: authentication tokens, eg,
  - physical keys, ATM card
- What you are: biometric features, eg,
  - fingerprint, voice recognition, retina scan

### Password limitations

- Eavesdropping
- Online password guessing
  - defense: limit number of attempts after which user must talk to admin
    - problem: vandal can easily lock up accounts (denial-of-service)
  - defense: limit speed of attempts
- Exposure of password file on server
  - Doing offline guessing if password file is hashed.
- Exposing passwords in email, script files, etc.

### Good password, ie, random 64-bit, not feasible

- 20 random digits
- 11 random chars (from 0-9, a-z, A-Z, couple of punctuation marks)
- Computer-generated random pronounceable password
  - Case insensitive: 4.5 bits of randomness per character
  - Every third character a vowel, 6 vowels: 2.5 bits of randomness per vowel
  - Requires 16 characters
- Human-generated passwords
  - About 2 bits of randomness per character
  - So require about 32 character password
- If password is too good, users write it down

### Workable approach

- “pass-phrase” with intentional misspelling, punctuation marks, symbols (eg, \$ for S), odd capitalization, etc.

## Login Trojan Horse to capture passwords

- Leave program running on public terminal that imitates login prompt
  - gets password from naive user and attempts to exit inconspicuously
    - eg, exit with “login failed” message
    - better yet: runs virtual OS for duration of user session
- Defenses by OS/hardware:
  - Have special prompt symbol at any input field by non-login program
  - Allow only login screen to fill entire display
  - Non-mappable key to interrupt any running program
    - eg, alt-ctrl-del (but often OS allows remapping of this)
  - Display number of unsuccessful login attempts since last successful login.
  - **Any defense fails given a sufficiently naive user**

Initial distribution of passwords needs to be secure

Passwords can also be used for non-login purposes (protecting individual files)

## Autentication tokens

Physical device that a person carries around:

- physical key, magnetic strip card, smart card, etc.

### Magnetic strip cards

- Credit cards, debit cards, id cards, money card, etc
- Can hold high-quality secret and other data (usually read-only)
- If card has picture or signature, then also serves as biometric check by human.

### Smart card (embedded CPU and memory)

- can hold high-quality secret
- memory can be password protected
- can do cryptographic operations (challenge/response)

### Advantages, disadvantages, features

- Tokens can be lost or stolen (unless it is attached/embedded in user)
  - So usually needs to be augmented with password
  - When token is lost, need an override that is usually not much less convenient than the override for “I forgot my password”
- People seem less willing to “loan” a token than to share a password
- Requires custom hardware (key slot, card reader, etc) on every access device
  - exception is cryptographic calculator (or readerless smart card)

## Cryptographic calculator (or readerless smart card)

- Smart card that does not require special hardware.
- Has display and keyboard for human interaction
  - User enters password to unlock device
  - User enters challenge into device and reads cryptographic response
- Time-based alternative
  - User enters password to unlock device
  - Card displays encryption of current time, which user enters as authentication information.
  - Authenticating computer checks that result is valid
    - Needs to check for all possible current times within allowed clock drift.
  - Advantages:
    - Saves half the typing
    - Works with password “form-factor” authentication protocols

## Biometric authentication devices

- Retinal scanner
  - scans blood vessels in back of your eye
  - expensive and “psychologically threatening” (look into laser device)
- Iris scanner
  - Less intrusive than retinal scanner (can be done with camera several feet away).
- Fingerprint reader
  - devices available but automation has not been successful for many years
- Face recognition
  - not intrusive but not very accurate susceptible to false negatives
- Handprint readers
  - More false positives than fingerprint readers but less expensive and less problem-prone
- Voiceprints
  - Cheap and can be as accurate as fingerprinting
  - Can be defeated with tape recording
  - False negatives (voice change due to illness)
- Keystroke timing
  - False negatives (injury)
- Signature
  - Not accurate based only on static signature
  - Accurate if also based on timing info