

Computer and Network Security

CMSC 414 Fall 2008

Udaya Shankar
shankar@cs.umd.edu

AUTHENTICATION

10/30/2008 shankar

authentication slide 1

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.

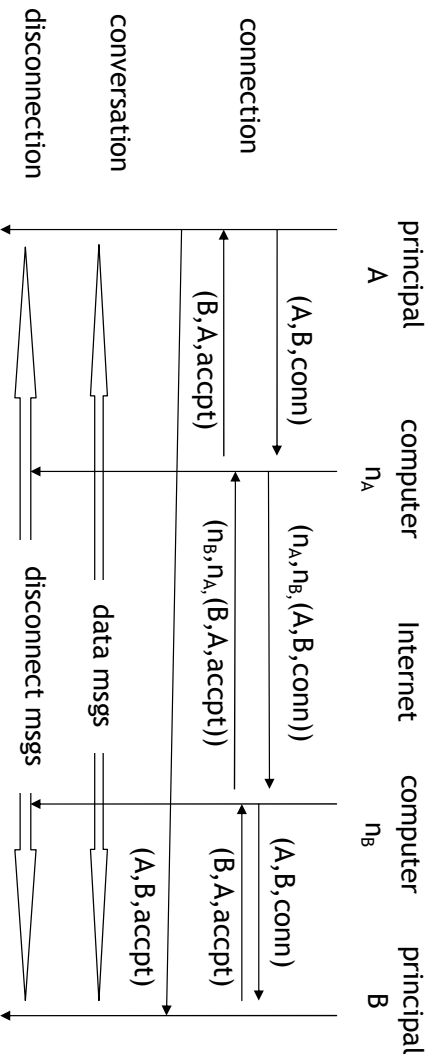
Authentication is about ensuring that:

- 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: $(B, A, acppt)$ sent by B in response to $(A, B, conn)$
 - B authenticates A: $(A, B, acppt)$ 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

10/30/2008 shankar

authentication slide 3

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)

10/30/2008 shankar

authentication slide 4

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

10/30/2008 shankar

authentication slide 5

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)

10/30/2008 shankar

authentication slide 6

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"> • enter $[A, B, pw_A]$ 		<ul style="list-style-type: none"> • send $[n_A, n_B, A, B, pw_A]$ to n_B 		
		<ul style="list-style-type: none"> • check rcvd $[A, pw_A]$ against passwd file • match authenticates A; msgs from n_A until logout assumed to be from A 		

- 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)

10/30/2008 shankar

authentication slide 7

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

- 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 $(X, \text{encrypt}_K(\text{hash}(pw_X)))$ where K is high-quality key maintained only in B's memory and not hard disk (i.e., manually entered when B is activated).

10/30/2008 shankar

authentication slide 8

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

10/30/2008 shankar

authentication slide 9

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)

10/30/2008 shankar

authentication slide 10

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.)

10/30/2008 shankar

authentication slide 11

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

10/30/2008 shankar

authentication slide 12

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 recieved 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)

10/30/2008 shankar

authentication slide 13

Scaling to network of N principals

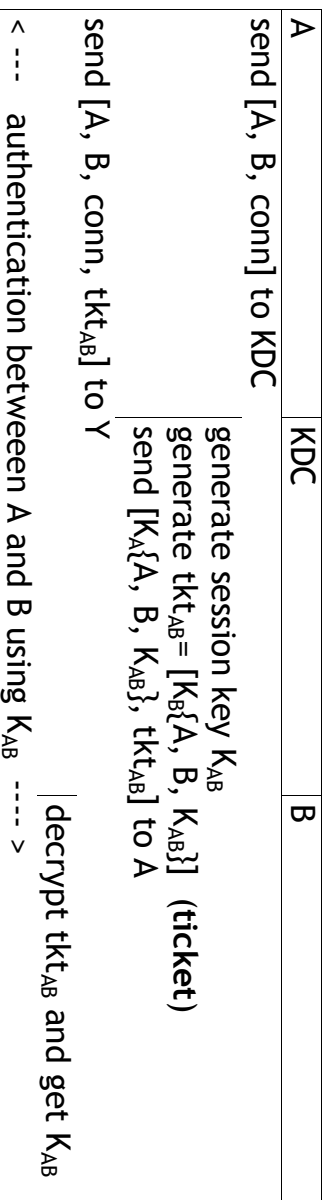
- Straightforward approach:
 - Distinct key for every pair of principals.
 - Not scalable:
 - N^2 storage cost at each node
 - N cost for adding new principal
- Use hierarchy of trusted intermediaries
 - **KDC (key distribution center)** in secret-key crypto
 - **CA (certification authority)** in public-key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

10/30/2008 shankar

authentication slide 14

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:



- 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 alleviated by having KDC replicas, but
 - need to protect all replicas
 - when a principal's master key is changed, need to sync replicas

10/30/2008 shankar

authentication slide 15

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:
 - $[X, pubkey_X, expdate], privkey_{CA}\{X, pubkey_X, expdate}\}]$.
- 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.

10/30/2008 shankar

authentication slide 16

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 numbers of unexpired revoked certificates]

10/30/2008 shankar

authentication slide 17

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 $tk_{A,Y} = [K_{X,Y}\{A, X, K_{A,Y}\}]$ send [$K_{A,Y}\{A, Y, K_{A,Y}\}$, $tk_{A,Y}$] to A	generate session key $K_{A,B}$ generate $tk_{A,B} = [K_{B,Y}\{A, B, K_{A,B}\}]$ send [$K_{A,Y}\{A, B, K_{A,B}\}$, $tk_{A,B}$] to A	send [A, B, conn, $tk_{A,Y}$] to Y

10/30/2008 shankar

authentication slide 18

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_{A-X2}] to talk to X_2
 - A talks to X_2 and gets [session-key, ticket_{A-X3}] to talk to X_3 and so on until
 - A talks to X_N and gets [session-key, ticket_{A-B}] 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.

10/30/2008 shankar

authentication slide 19

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">• 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.	<ul style="list-style-type: none">• 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	
	<ul style="list-style-type: none">• A can now send messages to B encrypted with B's public key		

10/30/2008 shankar

authentication slide 20

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

10/30/2008 shankar

authentication slide 21

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)

10/30/2008 shankar

authentication slide 22

Authentication of People (KPS 10: mar 20 2006)

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.

10/30/2008 shankar

authentication slide 23

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.

10/30/2008 shankar

authentication slide 24

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)

10/30/2008 shankar

authentication slide 25

Authentication 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)

10/30/2008 shankar

authentication slide 26

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

10/30/2008 shankar

authentication slide 27

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

10/30/2008 shankar

authentication slide 28

Security Handshake Pitfalls (NS chapter 11)

Assume A initiates connection to B.

Can classify the authentication protocols along following features:

- One-way authentication:
 - B authenticates A (eg, login) or
 - A authenticates B (server B with public key, client A w/o public key)
- Mutual-authentication:
 - B authenticates A and A authenticates B
- Session-key:
 - Authentication may also establish session key
- Secret-key crypto vs Public-key crypto

10/30/2008 shankar

authentication slide 29

One-Way Authentication

Solution 1.1: one-way auth, secret-key (K_{AB})

A	B
send [A, B, conn] -->	
	<-- send challenge R
send response $\{f\{K_{AB}, R\}$ -->	

Note

- Response $\{f\{K_{AB}, R\}$ is a keyed-hash of R or R encrypted with K_{AB}
- Challenge R must be new (a nonce) so that $\{f\{K_{AB}, R\}$ has not been sent before (by A or by B) and hence has not been seen by attacker.
- If challenge R is obtained from a clock or a counter and if B may have received past msgs m to which it sent $\{f\{K_{AB}, m\}$ responses (eg, another authentication protocol with A using K_{AB}) then
 - B must ensure that challenge R is not among these msgs, or
 - response should also indicate the sender (eg, $\{f\{K_{AB}, A, R\}$)
- These problems are not there if R is obtained from a random number generator.

10/30/2008 shankar

authentication slide 30

Some vulnerabilities:

- If K_{AB} is derived from password, an eavesdropper can do off-line password guessing attack.
- If attacker gets B's password file, it can impersonate A
 - Protecting password file is harder if B is replicated or A uses same password on different servers.
- Attacker can impersonate B since this is one-way authentication only (but this is a “feature” rather than a vulnerability).

10/30/2008 shankar

authentication slide 31

Solution 1.2: one-way auth, secret-key (K_{AB})

A	B
send [A, B, conn] to B -->	
send [R] -->	<--- send challenge [$K_{AB}\{R\}$]

Note

- Requires challenge to be reversible (ie, encryption, not keyed-hash).
- R should not only be a nonce but unpredictable (ie, randomly generated).
 - Eg, if R is obtained from a counter, an attacker can impersonate A because it would know that the next challenge generated by B is $R+1$.

Vulnerabilities: as in solution 1.1 plus the following:

- If K_{AB} derived from password and R has structure, then a spoofer (w/o eavesdropping) can get $K_{AB}\{R\}$ and do offline guessing.
 - Note: R is randomly generated and need not have structure.

Feature

If A and B have clocks that are within D seconds of each other and R has a timestamp (in addition to the random number), then this also authenticates B to A in the following sense:

- A assured that $K_{AB}\{R\}$ message was originally sent by B within last D seconds
- A not assured that $K_{AB}\{R\}$ was sent in response to its [A,B,conn] msg
 - Can be fixed by including a nonce in [A,B,conn] and in R.

10/30/2008 shankar

authentication slide 32

Solution 1.3: one-way auth, secret-key (K_{AB}), timestamp-based

Assuming A and B have clocks that are within D seconds of each other.

A	B
send [A, B, conn, $K_{AB}\{ts\}$] to B -->	B decrypts, checks that ts within D

Note

- Single transmission suffices, no handshake needed
- B does not need to maintain state per active connection

Vulnerable

- Replay attack within clock skew D
 - defense: B remembers ts sent by A within last D seconds (requires state)
 - Replay attack if K_{AB} used with multiple servers
 - defense: include server id along with ts
- May not be doable if servers are replicas of B (with same external id)
- B's clock being set back

- If encryption is replaced by keyed-hash, B has much more work
- B has to get keyed-hash of every possible value in D and compare.
 - Can overcome by A including unencrypted ts in conn msg. (Is this as secure?)

Solution 1.4: one-way auth, public-key (open challenge, signed response)

A	B
send [A, B, conn] to B -->	
send [$[R]_A$] -->	<-- send challenge R
($[R]_A$ is R encrypted with A's private key)	

Note

- B's pw file contains A's public key; can be readable (but not modifiable)
- Need to ensure that R has distinct structure that is not used for signing messages

Solution 1.5: one-way auth, public-key (encrypted challenge, open response)

A	B
send [A, B, conn] to B -->	<-- send encrypted challenge $\{R\}_A$
send R -->	$\{R\}_A$ is R encrypted with A's public key)

Note

- B's pw file contains A's public key; can be readable (but not modifiable)
- Need to ensure that R has distinct structure that is not used for sending confidential messages to A
- Why is ok to send response R in the open, instead of say $\{R\}_B$

Mutual (two-way) Authentication (A initiates connection to B)

Solution 2.1: two-way auth, secret key (K_{AB})

A	B
1 send [A, B, conn] to B -->	
2	<-- send challenge R_1
3 send [$\{K_{AB}, R_1\}$] -->	
4 send challenge R_2 -->	
5	<-- send [$\{K_{AB}, R_2\}$]

Note

- Consists of two 2-way handshakes
- Messages 3 and 4 can be combined into one message
- Vulnerable to B's passwd file being read
- If K_{AB} obtained from passwd, vulnerable to off-line pw guessing
 - by attacker who can eavesdrop
 - by attacker who can impersonate B
 - Impersonating server B is harder than impersonating client A (assuming server is always connected whereas client is momentary)
- Interchanging order of R_1 and R_2 introduces further vulnerability (below)

Solution 2.2: solution 2.1 with R_1 - R_2 order interchanged

A	B
1 send [A, B, conn, R_2] to B -->	
2	<-- send [challenge R_1 , response $\{K_{AB}, R_2\}$]
3 send [$\{K_{AB}, R_1\}$ -->	

Note

- Reduces solution 2.1 to one 3-way handshake
- As usual, vulnerable to B's passwd file being read
- Usual off-line pw guessing attack if C eavesdrops and K_{AB} obtained from passwd
- If C can spoof A, then C can do off-line pw guessing (without eavesdropping)

10/30/2008 shankar

authentication slide 37

Solution 2.2 vulnerable to reflection attack

C	B
1 send [A, B, conn, R_2] to B -->	<-- send [R_1 , $\{K_{AB}, R_2\}$]
2	
1' send [A, B, conn, R_1] to B -->	<-- send [S_1 , $\{K_{AB}, R_1\}$]
2'	
3 send [$\{K_{AB}, R_1\}$ -->	

C has successfully impersonated A to B

Possible defenses:

- B remembers R_1 and does not accept it (difficult with replicated servers)
- R has structure indicating sender of challenge (but then offline-pw guessing)
- Use different keys for each direction:
 - K_{AB} (for $A \rightarrow B$) and K_{BA} (for $A \leftarrow B$)
 - K_{BA} can be predictably related to K_{AB}
[eg, K_{AB+1} , K_{AB-1} , $-K_{AB}$, or $K_{AB} \oplus (F0F0\dots F0)_{16}$]

Thumb-rule: Initiator should be first to authenticate itself

10/30/2008 shankar

authentication slide 38

Solution 2.3: two-way auth, secret key, timestamps

A	B
1 send [A, B, conn, $f(K_{AB}, ts)$] -->	
2	<-- send [$f(K_{AB}, ts + 1)$]

Note

- One 2-way handshake suffices
- Msg 1 assures B that msg was generated by A and sent within clock skew
- “ts+1” can be replaced by any predicatable function of ts
 - response should include structure indicating sender (to defend against replay attack), or
 - B must remember timestamp values ts and ts+1 (to defend against replay attack)

10/30/2008 shankar

authentication slide 39

Solution 2.4: two-way auth, public keys

A	B
1 send [A, B, conn, $\{R_2\}_B$] -->	
2	<-- send $[R_2, \{R_1\}_A]$
3 send $[R_1]$ -->	

Note

- More rugged than secret-key: not vulnerable to overrunning B.
- Is it necessary to encrypt response R_1 ?
- Human A has to obtain its private key and B’s public key (already discussed):
 - Directory service supplies A’s private key encrypted by A’s pwd
 - B supplies B’s public key signed by A’s private key
 - etc

Solution 2.5: two-way auth, public keys, variant of solution 2.4

A	B
1 send [A, B, conn, R_2] -->	
2	<-- send $[[R_2]_B, R_1]$
3 send $[[R_1]_A]$ -->	

10/30/2008 shankar

authentication slide 40

Establishing session key with secret-key authentication

- Consider A and B with shared key K_{AB} .
During authentication, A and B have exchanged challenges, eg:
 - R_1 (in one-way auth)
 - R_1, R_2 (in two-way auth)

- Session key can be R_1 and/or R_2 encrypted by a specified function g of K_{AB} , eg,
 - $g(K_{AB})\{R_1\}$ or $(g(K_{AB}))\{R_1 \oplus R_2\}$
 - $g(K_{AB})$ is $K_{AB}+1, K_{AB}-1, -K_{AB}$, etc

Attack: if C obtains K_{AB} later, C can decrypt (recorded) conversation.

- Session key should not be $g(R_1)$ or $g(R_1, R_2)$ encrypted by K_{AB} , eg, $K_{AB}\{g(R_1)\}$.
Otherwise, later C can impersonate B, send $g(R_1)$ as a challenge to A, get back $K_{AB}\{g(R_1)\}$, and decrypt earlier conversation between A and B.
- Session key can be obtained by Diffie-Hellman after/during authentication (the Diffie-Hellman exchange messages are encrypted by K_{AB}).
Then even if C obtains K_{AB} later, it still cannot decrypt conversation.

10/30/2008 shankar

authentication slide 41

Establishing session key with public-key authentication

- A chooses random R as session key and sends $\{R\}_B$ to B.

Attack: C spoofs A (after authentication) and choose its own R_1 as session-key.
So important to have R be part of authentication.

- A chooses R as session key and sends $[\{R\}_B]_A$
Here C cannot inject spurious R_1 as session-key
- Attack:** If C later obtains B's private key, C can extract R and decrypt conversation.

- A picks R_1 , B picks R_2 , they exchange $\{R_1\}_B$ and $\{R_2\}_A$, set $R_1 \oplus R_2$ as session key.
Attack: Here C has to overrun both A and B to obtain session key.

- Session key can be obtained by Diffie-Hellman after/during authentication (the Diffie-Hellman exchange messages are encrypted or signed).
Then even if C overruns A and B, it still cannot decrypt conversation.

10/30/2008 shankar

authentication slide 42

Extensions for dynamic context

Dynamic context:

- users join and leave domains
- users do not share pre-assigned keys
- users rely on KDCs / CAs / directory services
- users change passwords
- replicated KDCs
- etc

New attacks become relevant:

- attacker with an old password of a user (trying to impersonate user)
- others?

New situations have to be handled:

- user A presents user B a ticket issued under old password of B
- user A contacts a KDC that still has an old password of A
- etc

10/30/2008 shankar

authentication slide 43

Authentication with KDC mediator

A	KDC	B
send [A, B, conn] to KDC	generate session key K_{AB} generate $tk_{t_{AB}} = [K_B\{A, B, K_{AB}\}]$ send $[K_A\{K_{AB}\}, tk_{t_{AB}}]$ to A	
send [A, B, conn, $tk_{t_{AB}}$] to B		
	<----- A and B do mutual authentication using K_{AB} -----> (example follows)	
send $R_2, K_{AB}\{R_1\}$ to B		send challenge R_1 to A send $K_{AB}\{R_2\}$ to A
<--- A and B use K_{AB} (or derivative, eg, $(K_{AB}+1)\{R_1 \oplus R_2\}$ as session key data --->		

Note:

- Even if C is spoofing A, C cannot get access to K_{AB} .
- Is authentication between A and KDC needed (or is that already done above)?
- Even if C is spoofing KDC, C cannot give a K_{AB} that B will accept.

10/30/2008 shankar

authentication slide 44

Needham-Schroeder Protocol

Below N_1, N_2, N_3 are nonces.

A	KDC	B
1 send [A, B, N_1] to KDC		
	generate session key K_{AB} generate $tk_{t_{AB}} = [K_B\{A, B, K_{AB}\}]$ send $[K_A\{N_1, B, K_{AB}, tk_{t_{AB}}\}]$ to A	
	3 send [$tk_{t_{AB}}, K_{AB}\{N_2\}$] to B	
		4 send $K_{AB}\{N_2-1, N_3\}$ to A
5 send $K_{AB}\{N_3-1\}$ to B		
<---- use K_{AB} (or derivative, eg, $(K_{AB}+1)\{R_1 \oplus R_2\}$ as session key data ---->		

10/30/2008 shankar

authentication slide 45

Needham-Schroeder (cont)

- Nonce N_1 , used to assure A that msg 2 is response by KDC to msg 1
- Defends against following attack:
 - C records above exchange (refer to them as old msgs 1, 2, 3, 4, 5)
 - C steals K_B , so it can decrypt $tk_{t_{AB}}$ and get K_{AB}
 - B changes key
 - C waits until A initiates connection to B
 - C intercepts A's new msg 1, responds with old msg 2 (= $K_A\{B, K_{AB}, tk_{t_{AB}}\}$)
 - A responds with new msg 2 (= $[tk_{t_{AB}}, K_{AB}\{new\ N_2\}]$ to B
 - C intercepts, responds with $K_{AB}\{new\ N_2 - 1\}$ (C knows K_{AB})
- Msg 2: id B encrypted by K_A ensures that C cannot replay old KDC reply to C (i.e., KDC reply to request by C to talk to B)
- Msg 2: no need to doubly encrypt $tk_{t_{AB}}$

10/30/2008 shankar

authentication slide 46

Needham-Schroeder (cont)

- If EBC is used (instead of CBC) and each nonce fits in an encryption block, then C can impersonate A to B with reflection attack
 - C eavesdrops and gets msgs 3 and 4
 - Later C replays msg 3
 - B replies with $K_{AB}\{N_2 - 1, N_4\}$ where $N_4 \neq N_3$
 - C needs to get $K_{AB}\{N_4 - 1\}$, which it does as follows:
 - C replays msg 3 with $K_{AB}\{N_4\}$ replacing $K_{AB}\{N_2\}$ and gets $K_{AB}\{N_4 - 1\}$ from B
 - Replacing EBC with CBC makes attack not possible (but then there is no need for $N_3 - 1$; can just use N_3)

10/30/2008 shankar

authentication slide 47

Needham-Schroeder (cont)

Vulnerability if N_1 sequential

1. Attacker C overhears $N_1 = n$ during normal session between A and B

A	KDC	B
1	send [A, B, N_1] to KDC	
	generate session key K_{AB} generate ticket $T_{AB} = [K_B\{A, B, K_{AB}\}]$ send [$K_A\{N_1, B, K_{AB}, T_{AB}\}]$ to A	
2		
3	send [$T_{AB}, K_{AB}\{N_2\}$] to B	
4		send $K_{AB}\{N_2 - 1, N_3\}$ to A
5	send $K_{AB}\{N_3 - 1\}$ to B	
	<----- A and B exchange data, close ----->	

10/30/2008 shankar

authentication slide 48

Needham-Schroeder vulnerability if N_1 sequential (cont)

2. Attacker C learns K_B , spoofs A to KDC with $N_1 = n+1$ as follows

attacker C	KDC	B
6	send [A, B, $N_1 = n+1$] to KDC	
	generate session key J_{AB} generate ticket $S_{AB} = [K_B\{A, B, J_{AB}\}]$ send [$K_A\{N_1, B, J_{AB}, S_{AB}\}]$ to A (rcvd by C)	
7		

3. C steals K_B . B changes its key.

C waits for A to connect to B, then impersonates KDC and then B

A	attacker C	B
8	send [A, B, $N_1 = n+1$] to KDC (intercepted by C)	
9	send [$K_A\{N_1, B, J_{AB}, S_{AB}\}]$ to A (replay msg 7)	
10	send [$S_{AB}, J_{AB}\{L_2\}$] to B (intercepted by C)	
		C decrypts S_{AB} (encrypted using (old) K_B and obtains J_{AB})
	<----- C can now complete the authentication and impersonate B ----->	

Needham-Schroeder has another vulnerability

If C gets A's master key, C can impersonate A to B even after A changes master key (because B never talks to KDC).

Below J_A is old A master key, J_{AB} is old session key:

A	KDC	B
1	send [A, B, N_1] to KDC	
	generate session key J_{AB} generate $tk_{AB} = [K_B\{A, B, J_{AB}\}]$ send [$J_A\{N_1, B, J_{AB}, tk_{AB}\}]$ to A	
2		
3	send [$tk_{AB}, J_{AB}\{N_2\}$] to B	
4		send $J_{AB}\{N_2-1, N_3\}$ to A
5	send $J_{AB}\{N_3-1\}$ to B	

C records above. Then C obtains J_A and A changes master key to $K_A (\neq J_A)$.

C	B
send [$tk_{AB}, J_{AB}\{M_2\}$] to B	
send $J_{AB}\{M_3-1\}$ to B	send $J_{AB}\{M_2-1, M_3\}$ to A

Expanded Needham-Schroeder fixes this at cost of two additional messages

	A	KDC	B
1a	send (A, B, conn) to B		
1b			send $K_B\{N_B\}$ to A
1	send [A, B, N_1 , $K_B\{N_B\}$] to KDC	generate session key K_{AB} generate $tk_{AB} = [K_B\{A, B, K_{AB}, N_B\}]$ send [$K_A\{N_1, B, K_{AB}, tk_{AB}\}$] to A	
2			
3	send [tk_{AB} , $K_{AB}\{N_2\}$] to B		
4			send $K_{AB}\{N_2-1, N_3\}$ to A (as before)
5	send $K_{AB}\{N_3-1\}$ to B (as before)		
	<---- A and B establish data session key (eg, $(K_{AB+1})\{R_1 \oplus R_2\}$ ---->		

10/30/2008 shankar

authentication slide 51

Otway-Rees authentication protocol

Does mutual authentication and handles ticket invalidation in 5 messages

	A	KDC	B
1	generate nonces N_A and N_C send [A, B, N_C , $K_A\{N_A, N_C, A, B\}$] to B		
2			generate nonce N_B send [$K_A\{N_A, N_C, A, B\}$, $K_B\{N_B, N_C, A, B\}$] to KDC
3		if N_C same in $K_A\{\dots\}$ and $K_B\{\dots\}$ generate session key K_{AB} extract N_B send [N_C , $K_A\{N_A, K_{AB}\}$, $K_B\{N_B, K_{AB}\}$] to B	
4			send $K_A\{N_A, K_{AB}\}$ to A
5	send $K_{AB}\{\text{anything recognizable}\}$ to B		
	<---- A and B establish data session key (eg, $(K_{AB+1})\{R_1 \oplus R_2\}$ ---->		

Note:

- Msg 3 assures B that request 1 was by A
- Msg 4 assures A that sender is B

10/30/2008 shankar

authentication slide 52

Otway-Rees nonce N_C must be unpredictable, o/w C can impersonate B to A.
 Suppose A uses sequence numbers for N_C and $N_C=007$ in one attempt.
 C observes this and sends request to B with $N_C=008$.

C	KDC	B
1	send [A, B, $N_C=008$, grbge] to B	send [grbge, $K_B\{N_B, N_C=008, A, B\}$] to KDC (C records this)
2	KDC rejects message 2	
Later A wants to connect to B		
A	KDC	C
3	send [A, B, $N_C=008$, $K_A\{N_A, N_C=008, A, B\}$] to B	C intercepts this msg 3 Forwards msg 3 and msg 2 to KDC
4	accept [msg 3, msg 2] (since their N_C 's match) send [N_C , $K_A\{N_A, K_{AB}\}$, $K_B\{N_B, K_{AB}\}$] to B	C intercepts msg 4 Forwards $K_A\{N_A, K_{AB}\}$ to A
5	send $K_{AB}\{\text{anything recognizable}\}$ to B	

At this point C has successfully impersonated B to A.

- If A attempts to use a data session key obtained from K_{AB} , C won't succeed
 O/w C can continue to impersonate B to A during the data exchange.

Nonce types:

- Large random number: best nonce
- crypto operations are the best way to generate them
- Timestamp: not as good
- clocks must have adequate synchronization and resolution
- must recover from crashes
- Sequence numbers
- requires non-volatile storage

Example 1: using seq number nonce when unpredictable nonce is needed

A	B
send [A, B, conn] to B	send challenge $K_{AB}\{R_1\}$ to A
send [R_1] to B	
If B obtains R_1 form a counter, C can impersonate A to B as follows	
C	B
send [A, B, conn] to B	send $K_{AB}\{R_2\}$ to A, where $R_2=R_1+1$
send [R_1+1] to B	

Example 2: using seq number nonce when unpredictable nonce is needed

A	B
send [A, B, conn] to B	
	send R_1 to A
send $[K_{AB}\{R_1\}]$ to B	

C lies in wait for A to initiate to B

- When A initiates to B,
C intercepts and sends challenge R_{i+1} to A and gets $K_{AB}\{R_{i+1}\}$.
- Then C initiates connection to B impersonating A.
- B sends challenge R_{i+1} , for which C now has the correct response.

Worse than man-in-middle: A does not have to be active for C to do attack.

Example 3: where sequence number nonce is adequate

A sends (A,B, conn);

B sends challenge $K_{AB}\{R\}$

A sends response $(K_{AB}+1)\{R\}$.
