Authentication

S.R. Shinde

Authentication

Prove continuity in relationship

- Basis of trust
- Identification

Who you are (biometrics)

Physical authentication: where you are

What you have (tokens)

What you know

Password: snoopy1

Mother's maiden name: jones

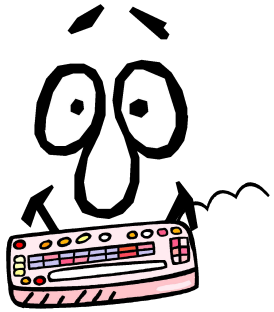
Pets name: snoopy

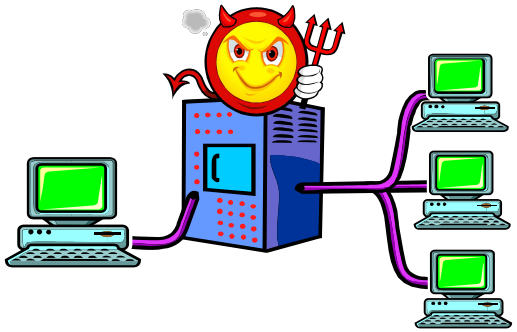
Network Authentication

- Password
- One-time Passwords (ex. tokens)
- Network address
  - Caller-id - credit card
  - IP address
  - MAC address – banks
- Cryptographic protocols

Concerns

- Impersonation
- Malicious insiders
- Eavesdropping
  - Keyboard sniffers
  - Shoulder-surfing
  - Network sniffers
  - Trojan horses





Mutual-Way Authentication

- Authentication often needed in both directions
- Server trusting user is not only concern
  - User must trust server
  - Ex. User accessing online bank account
- Variety of “solutions” in user applications

Password-based Authentication

- Proof by sharing
- Doesn't prevent insider attacks (system admin)
- What is an appropriate password
  - length? snoopy, snoopy1, snoopy12
  - reusable ? snoppy1, snoopy2, …. snoopy10, snoopy1
  - timeframe?
- How to do initial password distribution? lastname123, employee#
- Simple approach, works with humans

  … until user has too many to remember
  - reuse across systems
  - Variations of something common: dog's name
  - post-it on monitor
  - inconvenient to update, varying rules on what is appropriatel how often to change

    snoopy1, Snoopy1, snoopy-1

Password Guessing

- Online
  - Limited tries, exponential delays, alarm
  - But attacker can temporarily disable a user's account – ex. 3 tries and account locked until user calls help desk

- Offline: "dictionary" attack
  - Must capture password file
  - Try "obvious" passwords: snoopy, snoopy1, 1snoopy
  - Significant dates, easy patterns, personal information
  - While most systems disallow "dictionary words", complexity rules still give information – know need a digit, punctuation character …

Snoopy-12

Passwords as Keys

- Directly as the key
- Convert to secret key
  - Transient, one-way transformation (hash)
  - Increase work of attacker
- Seed to pseudo-random number generator

One-Time Passwords

- List of passwords used once only
- Need to re-initialize periodically

Lamport's Hash

- One-time passwords
- Server stores n, $hash^n$(password)
- User sends x = $hash^{n-1}$(password)
- Server computes hash(x), if = stored value, replaces stored value with n-1 , x

- Safe from eavesdropping, server compromise not a problem for user
- No public key cryptography
- Authentication not mutual
- Add salt to password before hashing
  - In case Alice uses same password on multiple systems
  - Salt must be stored on Alice's system
  - Server uses $hash^n$(password | salt)

OTP Generators (Tokens)

- Examples
  - RSA
  - VASCO Digipass

- Use a block cipher
  - Repeatedly encrypt
  - Continuously update every x seconds
  - Update each time user presses button
- Some work in both directions
  - Customer enters OTP
  - Server returns OTP, customer (manually) compares it to value on token

Tokens - Issues

- Help desk required
  - Synchronization not perfect
  - Premature battery death
- Cost
  - $15-$25
  - banks with million customers
- User still needs pin (something you know + something you have)
- “Necklace of Tokens” issue
  - Only recently integrated with cell phones
  - Still rare to have multiple tokens on single device
- Non-standard algorithms
  - OATH effort

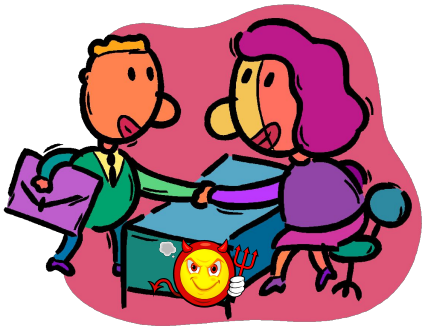
KDC

- Can impersonate any node
- Single point of failure
- Potential bottleneck

Certificate Authority

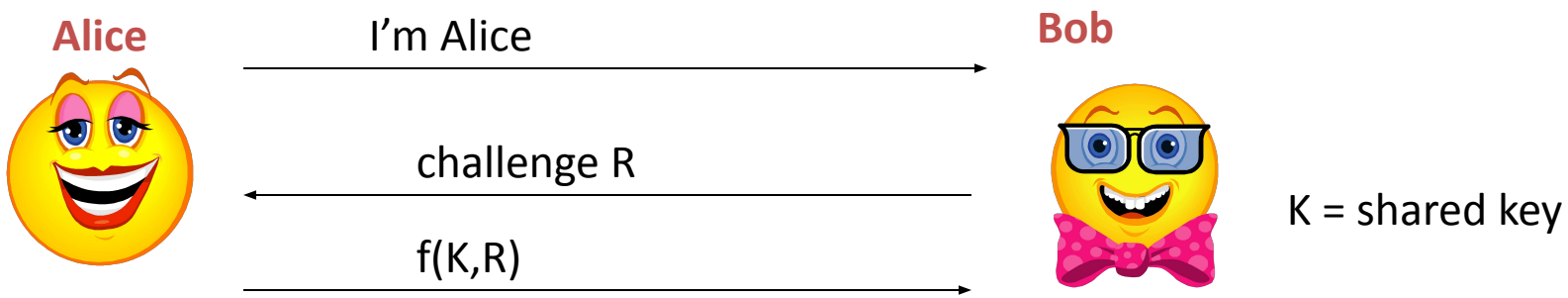
- Central point for certificates
- Signs cert for Alice containing her public key
- Others need only CA's public key
- Revocation?
  - Real time with online KDC
  - Offline CA –expiration date, certificate revocation list

Security Handshakes



- Considerations when creating protocols
  - Attacks/Information leakage
  - One-way
  - Mutual Authentication

One-way Challenge-Response



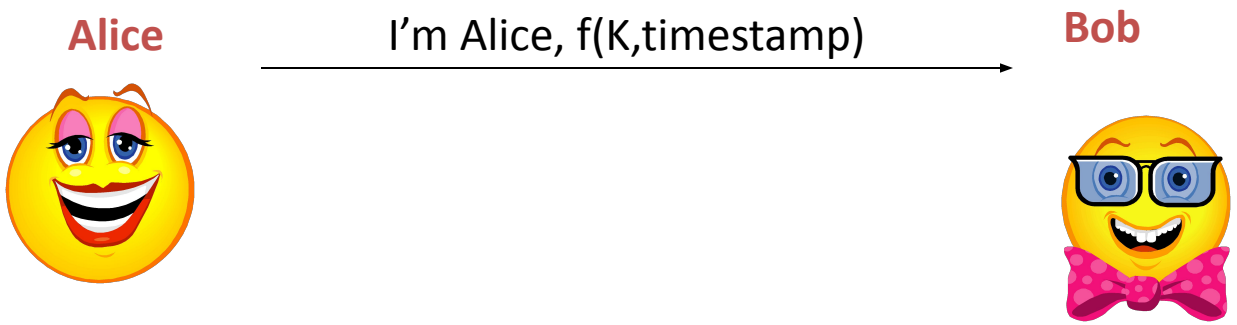
f:

- encryption function – Bob just decrypts and verifies time in within allowed skew
- hash – Bob needs to hash all times in allowable interval or Alice sends time

Problems?

- Authentication not mutual
- Connection hijacking after authentication – attacker spoofs Alice or Bob's source address and send packets if conversation not encrypted
- Off-line password/key attack – depends on length of K
- Compromise of database/disk if K is stored, or temporary memory access

One-Way using Timestamp



- Problems?
  - Impersonate Alice if intercept and send message – race condition
  - Keep list of used time stamps to prevent quick replay, but if use same K with multiple servers, could send message to another server and impersonate Alice
  - Clock skew/synchronization

One-way Using Public Key

Bob decrypts with Alice's public key and verifies R was returned.

Alice proves to Bob she has her private key by returning R

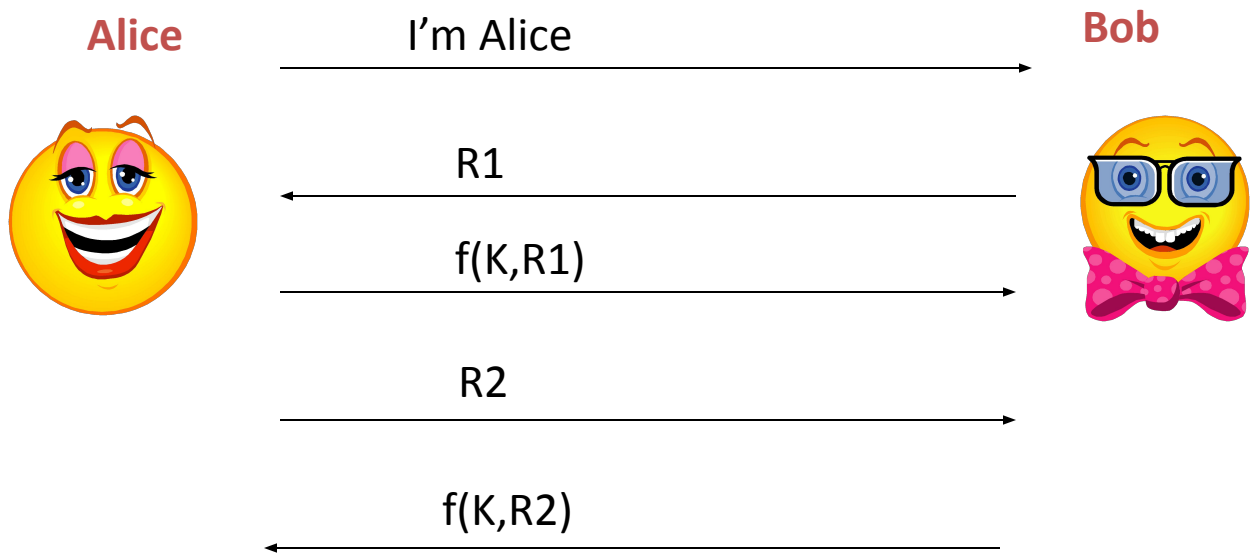
$[R]_{Ax}$ = R signed with Alice's x key, where x is private (priv) or public (pub) key

One-way Problems

- First case:
  - Can send anything to Alice as R and get Alice to sign it
- Second case:
  - Intercepted an encrypted message for Alice, send it and get Alice to decrypt it

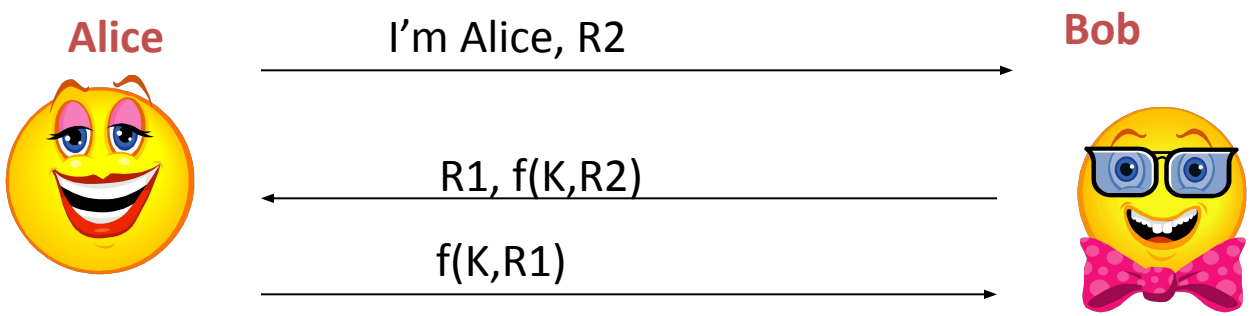
Mutual Authentication

Mutual Authentication with Secret Key



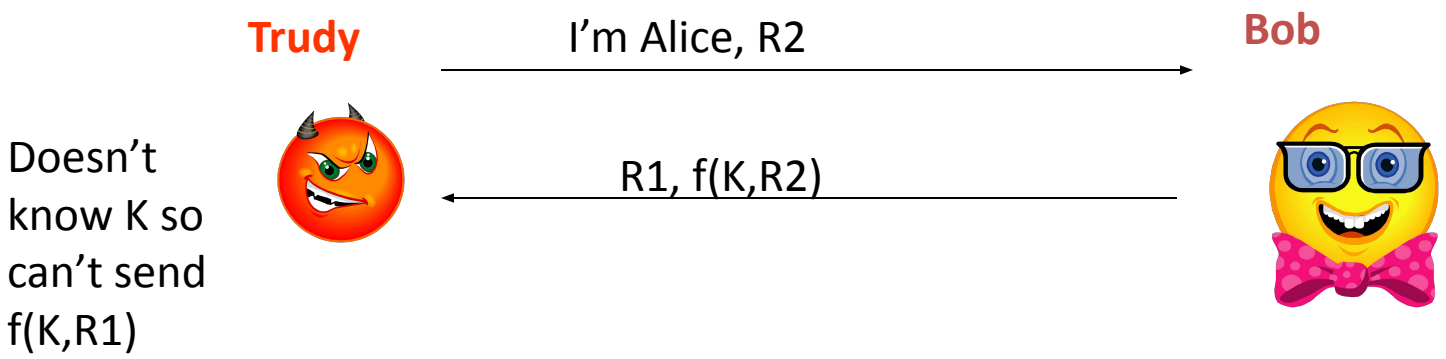
Mutual Authentication with Secret Key

More efficient version:



Mutual Authentication with Secret Key

Reflection attack:

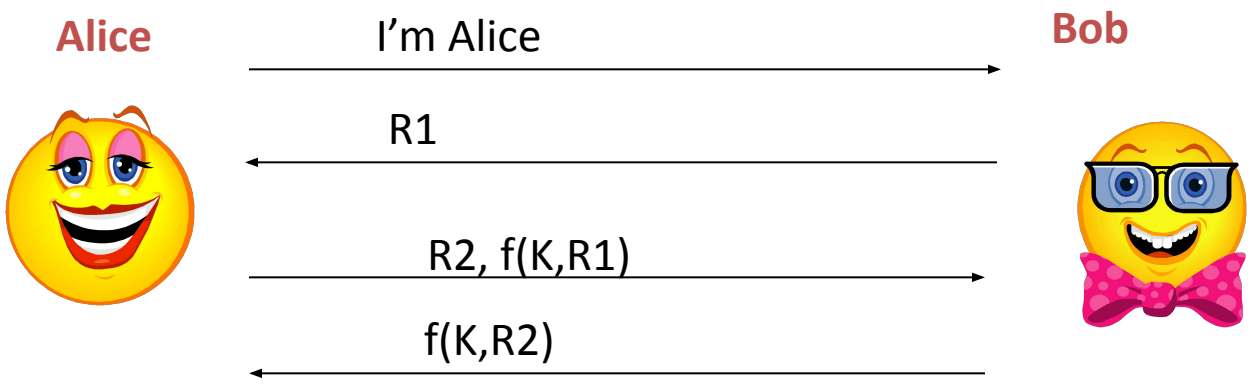


Solutions:

- Separate keys for each direction
- Requirements on R values: odd in one direction, even in the other, concatenate with senders' name

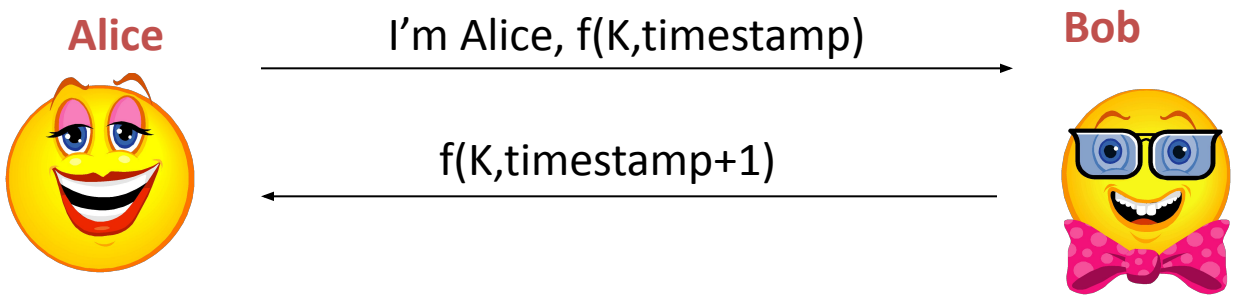
Password/Key Guessing

- Also note, Trudy can get Bob to encrypt a value (or a several of values) and then try an offline attack to guess K
- Have Bob return R1 value for Alice to encrypt



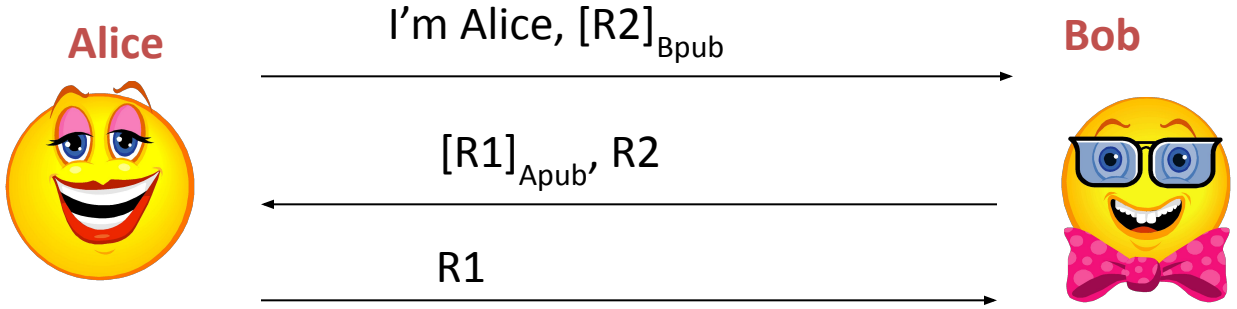
Now Bob would have to reuse R1 in order for Trudy, who eavesdrops, to be able to use f(K,R1)

Timestamps



- Same issues as before plus clock skew
- Any modification to timestamp will work

Mutual Authentication with Public Keys



- how to obtains/store/validate Bob's public key

Session Key

- Once authentication occurs, want to encrypt data exchanged
- Session key
- If key to one session obtained, can’t decrypt all sessions
- Don’t want known/easy to derive relationship among session keys

Centralized Authentication

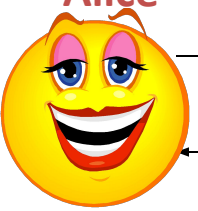
- Needham-Schroeder

Needham-Schroeder

- N1, N2, N3 are nonces ("number used once")

Expanded Needham-Schroeder

- Issue - ticket doesn't expire
- If Trudy obtains Alice's key and ticket, Trudy can connect to Bob even if Alice changes key.



Alice

Bob

I want to talk to you

$E_{kb}(Nb)$

N1, Alice wants to talk to Bob, $E_{kb}(Nb)$

KDC

$E_{Ka}(N1, \text{"Bob"}, Kab, \text{ticket})$

$\text{ticket} = E_{Kb}(Kab, \text{"Alice"}, E_{kb}(Nb))$

Nb is unique per session so can't reuse ticket

$E_{Kab}(N2)$, ticket

$E_{Kab}(N2 - 1, N3)$

$E_{Kab}(N3 - 1)$

Needham-Schroeder

- Reflection attack in last steps if ECB mode (and nonce size = block size)

  Trudy->Bob: E_{Kab}(N2), ticket

  Bob->Trudy: E_{Kab}(N2 - 1, N4)

  Trudy->Bob: E_{Kab}(N4), ticket

  Bob->Trudy: E_{Kab}(N4 - 1, N5)

  Extract E_{Kab}(N4 - 1) and use in response of first run

Trudy doesn't have Kab to obtain N4, needs N4-1 in next step, so get Bob to encrypt N4-1 Extract 1st block of ciphertext

Kerberos

- Based on Needham-Schroeder
- Uses time and includes ticket expiration

Nonces

- Random number
  - 128-bit value negligible chance of being repeated
- Timestamp
  - Synchronization
  - Predictable
- Sequence number
  - Maintain state
  - Predictable?

Random Numbers

- Be careful with seed
- Size
- Easily known value: timestamp
- Divulging seed – don’t use some value included unencrypted in message

Performance

- Number of messages exchanged
- Number of operations
  - using secret key algorithm
  - using public key algorithm
  - using hash
- And number of bytes involved

Checklist

- Eavesdropping
- Initiation of conversation/partial conversations
- Impersonation by accepting a connection
- Access to disk/database at either end
- Man-in-middle

Overview

- An essential part of Windows servers
- Authentication infrastructure
  - Designed to authenticate users to servers
  - Users must use their password as their initial key and must not be forced to retype it constantly
- Based on Needham-Schroeder, with timestamps to limit key lifetime

Kerberos

- trusted key server system from MIT
- provides centralised private-key third-party authentication in a distributed network
  - allows users access to services distributed through network
  - without needing to trust all workstations
  - rather all trust a central authentication server
- two versions in use: 4 & 5

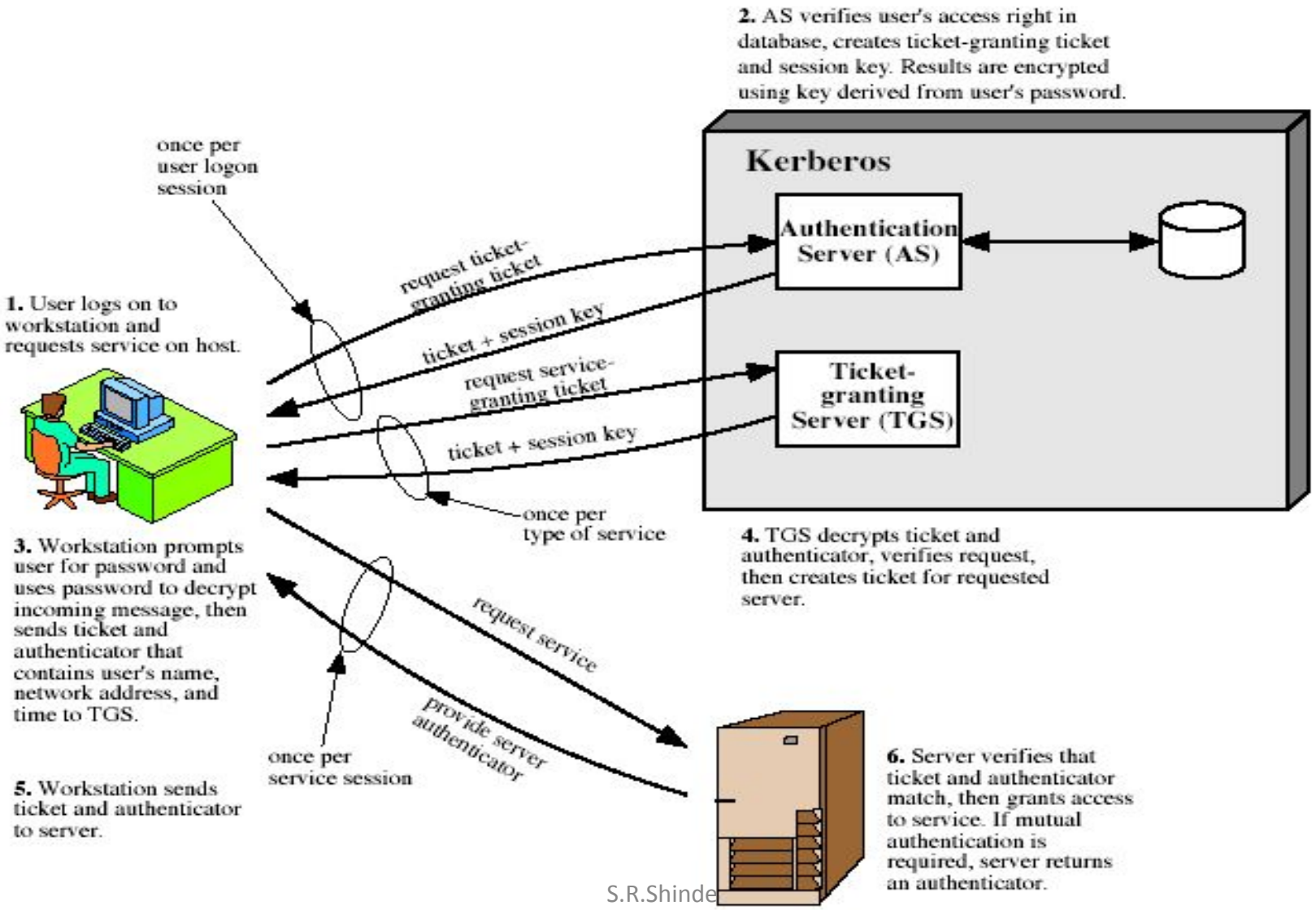
Kerberos Requirements

- first published report identified its requirements as:
  - security
  - reliability
  - transparency
  - scalability
- implemented using an authentication protocol based on Needham-Schroeder

Kerberos 4 Overview

- a basic third-party authentication scheme
- have an Authentication Server (AS)
  - users initially negotiate with AS to identify self
  - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
- have a Ticket Granting server (TGS)
  - users subsequently request access to other services from TGS on basis of users TGT

Kerberos 4 Overview



Kerberos Realms

- a Kerberos environment consists of:
  - a Kerberos server
  - a number of clients, all registered with server
  - application servers, sharing keys with server
- this is termed a realm
  - typically a single administrative domain
- if have multiple realms, their Kerberos servers must share keys and trust

Kerberos Version 5

- developed in mid 1990's
- provides improvements over v4
  - addresses environmental shortcomings
    - encryption alg, network protocol, byte order, ticket lifetime, authentication forwarding, interrealm auth
  - and technical deficiencies
    - double encryption, non-std mode of use, session keys, password attacks
- specified as Internet standard RFC 1510

Delegation

- Delegation of rights
  - Alice wants Bob to access resource X on behalf of Alice for time t.
    - Example: Alice logs into host Bob then wants to log into host X from Bob
  - Alice can request ticket with Bob's address or a list of addresses
  - Ticket can include application specific data – not used by Kerberos but used by host
- Can set to not allow delegation

Ticket Lifetime

- V4: 21 hours max
- V5: up to Dec. 31, 9999
- Lifetime in seconds
- Not revocable – be careful
- Time ticket granted, start time and stop time
- Renew until – instead of long lifetime, give option to keep renewing
  - If stop using/needing ticket, won't remain open
- Postdating
- Grant ticket to run some process in future
  - Batch job at end of week but requested ticket at beginning of week

Key Version

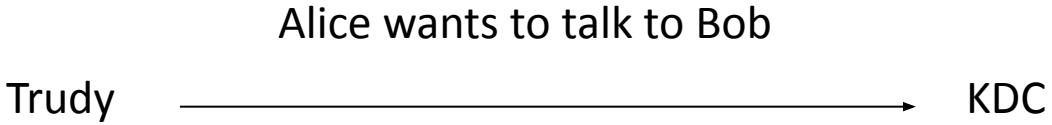
- Suppose Alice has ticket to Bob
- Bob changes his key with KDC
- KDC keeps versions both versions of Bob's key (key, version)
- Alice's ticket keeps working until it expires
- Any other renewable or post-dated ticket will work with old key

Master Keys and Realms

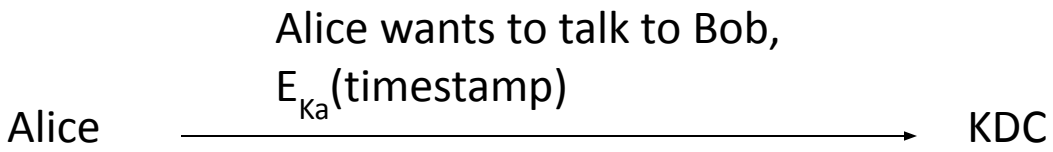
- Master keys – key between entity (such as Alice) and KDC
- Alice registered to realms R1, R2
  - Uses same password
- Hash password with realm name to get master key
- If attacker gets Alice's password, still can compute both master keys
- But R1 and R2 don't have the other's master key for Alice. If attacker breaks into one, won't get both keys.

Password Guessing

- V4: anyone can send ticket request to KDC



- V5: include encrypted timestamp



Password-Guessing

- Kerberos tickets have verifiable plaintext
- An attacker can run password-guessing programs on intercepted ticket-granting tickets (Merritt and Bellovin invented EKE while studying this problem with Kerberos.)
- Kerberos uses *passphrases* instead of *passwords*
- Does this make guessing harder? Not sure

Password Guessing

- On many Kerberos systems, anyone can ask the KDC for a TGT
- There's no need to eavesdrop to get them — you can get all the TGTs you want over the Internet.
- Solution: *preauthentication*
- The initial request includes a timestamp encrypted with Kc
- It's still verifiable plaintext, but collecting TGTs becomes harder again

Multiple Sessions – Added Security

- Alice opens two sessions to Bob
- Don’t want Trudy swapping messages between sessions
- Alice specifies different session key to use for each

Hash function

- A hash function is a reproducible method of turning some kind of data into a (relatively) small number that may serve as a digital "fingerprint" of the data.
- The algorithm "chops and mixes" (for instance, substitutes or transposes) the data to create such fingerprints.

Secure Hash Algorithm

- The SHA hash functions are five cryptographic hash functions - SHA-1, SHA-224, SHA-256, SHA-384, SHA-512.
- Hash algorithms compute a fixed-length digital representation (known as a message digest) of an input data sequence (the message) of any length.
- Any change to a message will, with a very high probability, result in a different message digest.

SHA

Algorithm	Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations	Collision
SHA-0	160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rotl	Yes
SHA-1	160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rotl	2^{63} attack
SHA-256/224	256/224	256	512	$2^{64} - 1$	32	64	+,and,or,xor,shr,rotr	None
SHA-512/384	512/384	512	1024	$2^{128} - 1$	64	80	+,and,or,xor,shr,rotr	None

Cryptanalysis of SHA-0

- At CRYPTO 98, two French researchers, Florent Chabaud and Antoine Joux, presented an attack on SHA-0 (Chabaud and Joux, 1998): collisions can be found with complexity 2^61, fewer than the 2^80 for an ideal hash function of the same size.

- In 2004, Biham and Chen found near-collisions for SHA-0 — two messages that hash to nearly the same value; in this case, 142 out of the 160 bits are equal. They also found full collisions of SHA-0 reduced to 62 out of its 80 rounds.

- Subsequently, on 12 August 2004, a collision for the full SHA-0 algorithm was announced by Joux, Carribault, Lemuet, and Jalby. This was done by using a generalization of the Chabaud and Joux attack. Finding the collision had complexity 2^51 and took about 80,000 CPU hours on a supercomputer with 256 Itanium 2 processors.

- On 17 August 2004, at the Rump Session of CRYPTO 2004, preliminary results were announced by Wang, Feng, Lai, and Yu, about an attack on MD5, SHA-0 and other hash functions. The complexity of their attack on SHA-0 is 2^40, significantly better than the attack by Joux et al.
- In February 2005, an attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu was announced which could find collisions in SHA-0 in 2^39 operations.

Cryptanalysis of SHA-1

- In light of the results on SHA-0, some experts suggested that plans for the use of SHA-1 in new cryptosystems should be reconsidered. After the CRYPTO 2004 results were published, NIST announced that they planned to phase out the use of SHA-1 by 2010 in favor of the SHA-2 variants.
- In early 2005, Rijmen and Oswald published an attack on a reduced version of SHA-1 — 53 out of 80 rounds — which finds collisions with a complexity of fewer than 2^80 operations.
- In February 2005, an attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu was announced. The attacks can find collisions in the full version of SHA-1, requiring fewer than 2^69 operations. (A brute-force search would require 2^80 operations.)
- The authors write: "In particular, our analysis is built upon the original differential attack on SHA0 , the near collision attack on SHA0, the multiblock collision techniques, as well as the message modification techniques used in the collision search attack on MD5. Breaking SHA1 would not be possible without these powerful analytical techniques.". The authors have presented a collision for 58-round SHA-1, found with 2^33 hash operations. The paper with the full attack description was published in August 2005 at the CRYPTO conference.
- In an interview, Yin states that, "Roughly, we exploit the following two weaknesses: One is that the file preprocessing step is not complicated enough; another is that certain math operations in the first 20 rounds have unexpected security problems."
- On 17 August 2005, an improvement on the SHA-1 attack was announced on behalf of Xiaoyun Wang, Andrew Yao and Frances Yao at the CRYPTO 2005 rump session, lowering the complexity required for finding a collision in SHA-1 to 2^63. On 18 December 2007 the details of this result were explained and verified by Martin Cochran.
- Christophe De Cannière and Christian Rechberger further improved the attack on SHA-1 in "Finding SHA-1 Characteristics: General Results and Applications,"receiving the Best Paper Award at ASIACRYPT 2006. A two-block collision for 64-round SHA-1 was presented, found using unoptimized methods with 2^35 compression function evaluations.
- As this attack requires the equivalent of about 2^35 evaluations, it is considered to be a theoretical break.To find an actual collision, however, a massive distributed computing effort or very large parallel supercomputer such as those possessed by the NSA would be required. To that end, a collision search for SHA-1 using the distributed computing platform BOINC is currently being made.
- At the Rump Session of CRYPTO 2006, Christian Rechberger and Christophe De Cannière claimed to have discovered a collision attack on SHA-1 that would allow an attacker to select at least parts of the message

SHA-1 algorithm

- **Note: All variables are unsigned 32 bits and wrap modulo 232 when calculating**

- **Initialize variables:**
- **h0 = 0x67452301**
- **h1 = 0xEFCDAB89**
- **h2 = 0x98BADCFE**
- **h3 = 0x10325476**
- **h4 = 0xC3D2E1F0**

- **Pre-processing:**
- **append the bit '1' to the message**
- **append k bits '0', where k is the minimum number ≥ 0 such that the resulting message**
- **length (in bits) is congruent to 448 (mod 512)**
- **append length of message (before pre-processing), in bits, as 64-bit big-endian integer**

- **Process the message in successive 512-bit chunks:**
- **break message into 512-bit chunks**
- **for each chunk**
- **break chunk into sixteen 32-bit big-endian words w[i], 0 <= i <= 15**

- **Extend the sixteen 32-bit words into eighty 32-bit words:**
- **for i from 16 to 79**
- **w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1**

- **Initialize hash value for this chunk:**
- **a = h0**
- **b = h1**
- **c = h2**
- **d = h3**
- **e = h4**
- **….**

SHA-1 algorithm

- **Main loop:**
- **for i from 0 to 79**
- **if 0 ≤ i ≤ 19 then**
- **f = (b and c) or ((not b) and d)**
- **k = 0x5A827999**
- **else if 20 ≤ i ≤ 39**
- **f = b xor c xor d**
- **k = 0x6ED9EBA1**
- **else if 40 ≤ i ≤ 59**
- **f = (b and c) or (b and d) or (c and d)**
- **k = 0x8F1BBCDC**
- **else if 60 ≤ i ≤ 79**
- **f = b xor c xor d**
- **k = 0xCA62C1D6**

- **temp = (a leftrotate 5) + f + e + k + w[i]**
- **e = d**
- **d = c**
- **c = b leftrotate 30**
- **b = a**
- **a = temp**

- **Add this chunk's hash to result so far:**
- **h0 = h0 + a**
- **h1 = h1 + b**
- **h2 = h2 + c**
- **h3 = h3 + d**
- **h4 = h4 + e**
- **Produce the final hash value (big-endian):**
- **digest = hash = h0 append h1 append h2 append h3 append h4**

SHA-2 algorithm

- **Initialize variables**
- **(first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):**
- **h0 := 0x6a09e667**
- **h1 := 0xbb67ae85**
- **h2 := 0x3c6ef372**
- **h3 := 0xa54ff53a**
- **h4 := 0x510e527f**
- **h5 := 0x9b05688c**
- **h6 := 0x1f83d9ab**
- **h7 := 0x5be0cd19**

- **Initialize table of round constants**
- **(first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311):**
- **k[0..63] :=**
- **0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,**
- **0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,**
- **0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,**
- **0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,**
- **0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,**
- **0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,**
- **0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,**
- **0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2**

- **Pre-processing:**
- **append the bit '1' to the message**
- **append k bits '0', where k is the minimum number >= 0 such that the resulting message**
- **length (in bits) is congruent to 448 (mod 512)**
- **append length of message (before pre-processing), in bits, as 64-bit big-endian integer**

SHA-2 algorithm

- **Process the message in successive 512-bit chunks:**
- **break message into 512-bit chunks**
- **for each chunk**
- **break chunk into sixteen 32-bit big-endian words w[0..15]**

- **Extend the sixteen 32-bit words into sixty-four 32-bit words:**
- **for i from 16 to 63**
- **s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15] rightshift 3)**
- **s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2] rightshift 10)**
- **w[i] := w[i-16] + s0 + w[i-7] + s1**

- **Initialize hash value for this chunk:**
- **a := h0**
- **b := h1**
- **c := h2**
- **d := h3**
- **e := h4**
- **f := h5**
- **g := h6**
- **h := h7**

SHA-2 algorithm

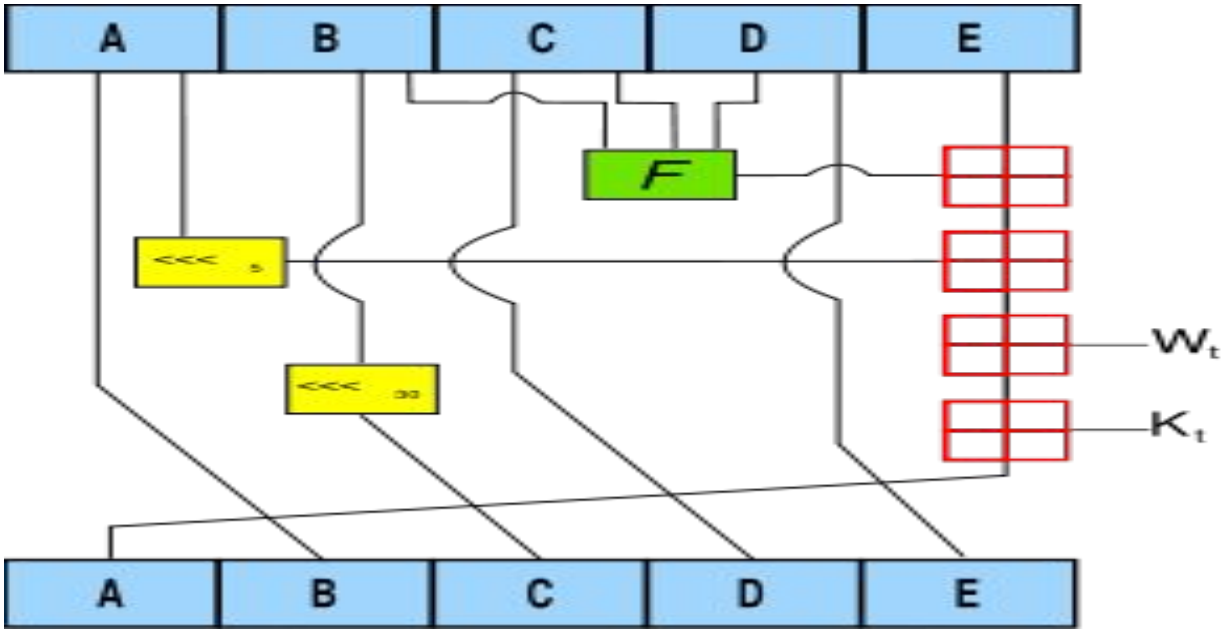
- **Main loop:**
- **for i from 0 to 63**
- **s0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)**
- **maj := (a and b) xor (a and c) xor (b and c)**
- **t2 := s0 + maj**
- **s1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)**
- **ch := (e and f) xor ((not e) and g)**
- **t1 := h + s1 + ch + k[i] + w[i]**

- **h := g**
- **g := f**
- **f := e**
- **e := d + t1**
- **d := c**
- **c := b**
- **b := a**
- **a := t1 + t2**

- **Add this chunk's hash to result so far:**
- **h0 := h0 + a**
- **h1 := h1 + b**
- **h2 := h2 + c**
- **h3 := h3 + d**
- **h4 := h4 + e**
- **h5 := h5 + f**
- **h6 := h6 + g**
- **h7 := h7 + h**

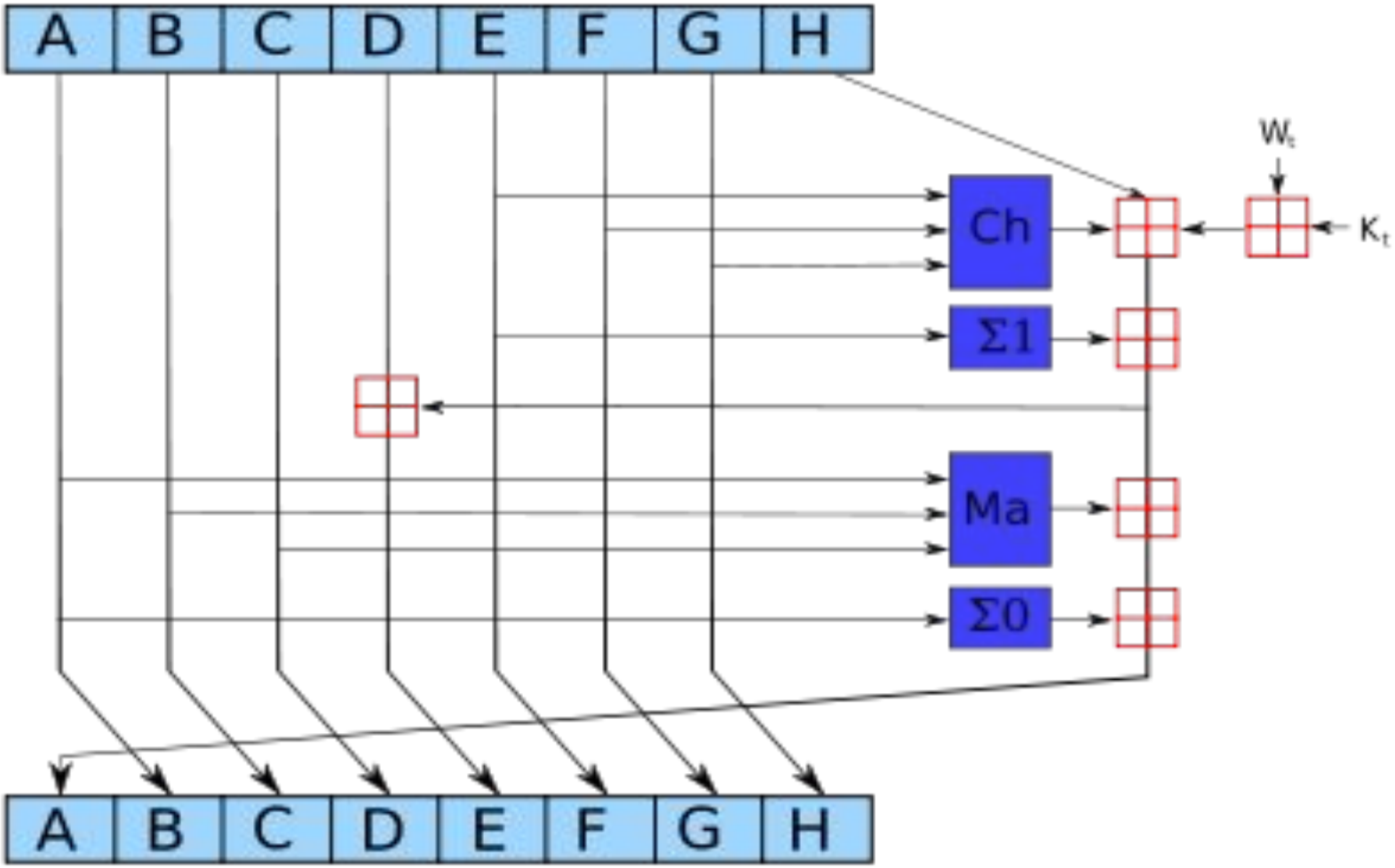
- **Produce the final hash value (big-endian):**
- **digest = hash = h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7**

SHA-1



- One iteration within the SHA-1 compression function. A, B, C, D and E are 32-bit words of the state; F is a nonlinear function that varies; n denotes a left bit rotation by n places; n varies for each operation. denotes addition modulo 2^32. Kt is a constant.

SHA-2



Example hashes

- SHA1("The quick brown fox jumps over the lazy dog")
- = 2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12

- SHA1("The quick brown fox jumps over the lazy cog")
- = de9f2c7f d25e1b3a fad3e85a 0bd17d9b 100db4b3

- The hash of the zero-length message is:
- SHA1("")
- = da39a3ee 5e6b4b0d 3255bfef 95601890 afd80709

Example hashes

- SHA1("The quick brown fox jumps over the lazy dog")
- = 2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12

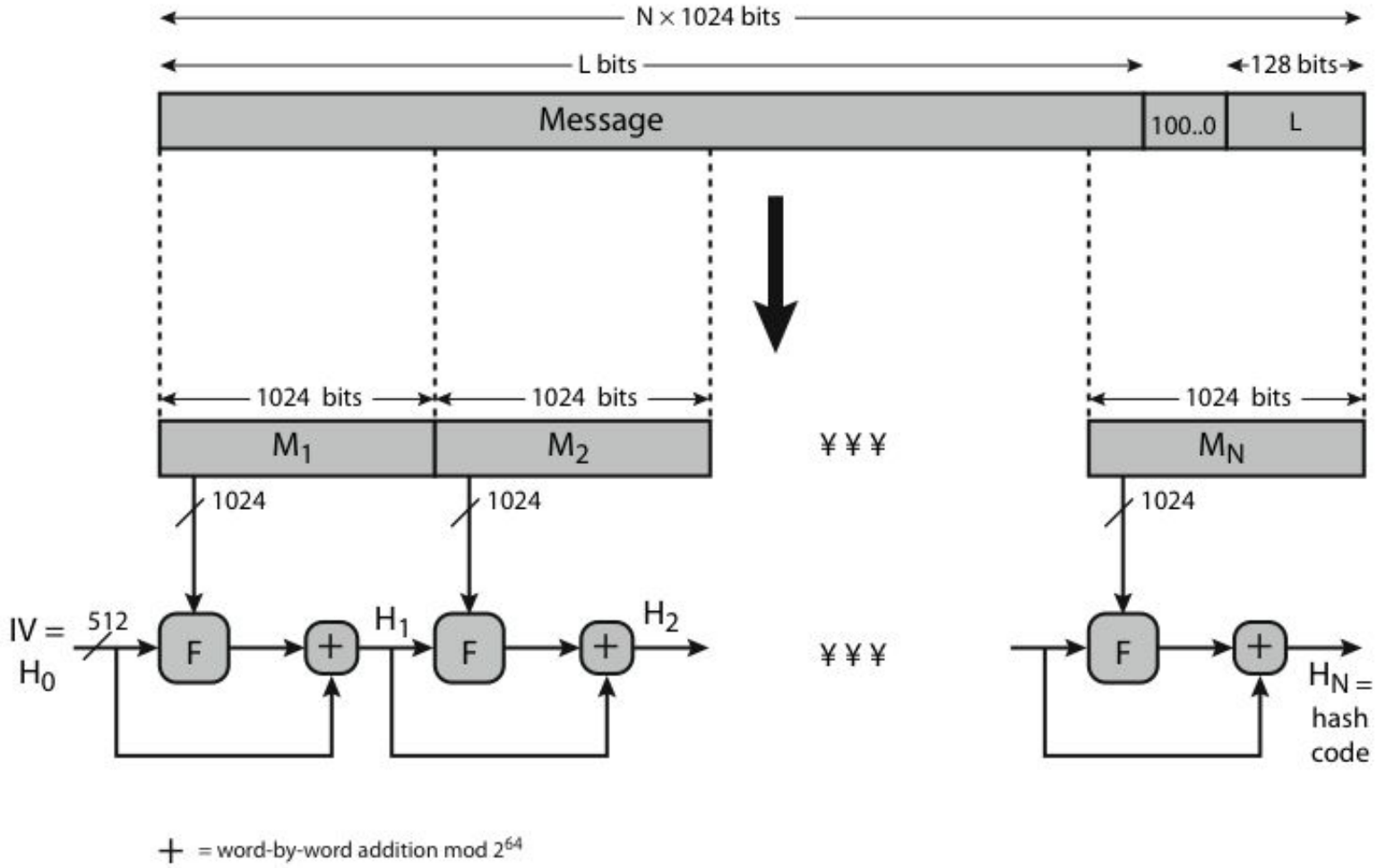
- **SHA-256**
- SHA256("The quick brown fox jumps over the lazy dog")
- = d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592

- **SHA-512**
- SHA512("The quick brown fox jumps over the lazy dog")
- = 07e547d9586f6a73f73fbac0435ed76951218fb7d0c8d788a309d785436bbb64
- 2e93a252a954f23912547d1e8a3b5ed6e1bfd7097821233fa0538f3db854fee6

SHA-512

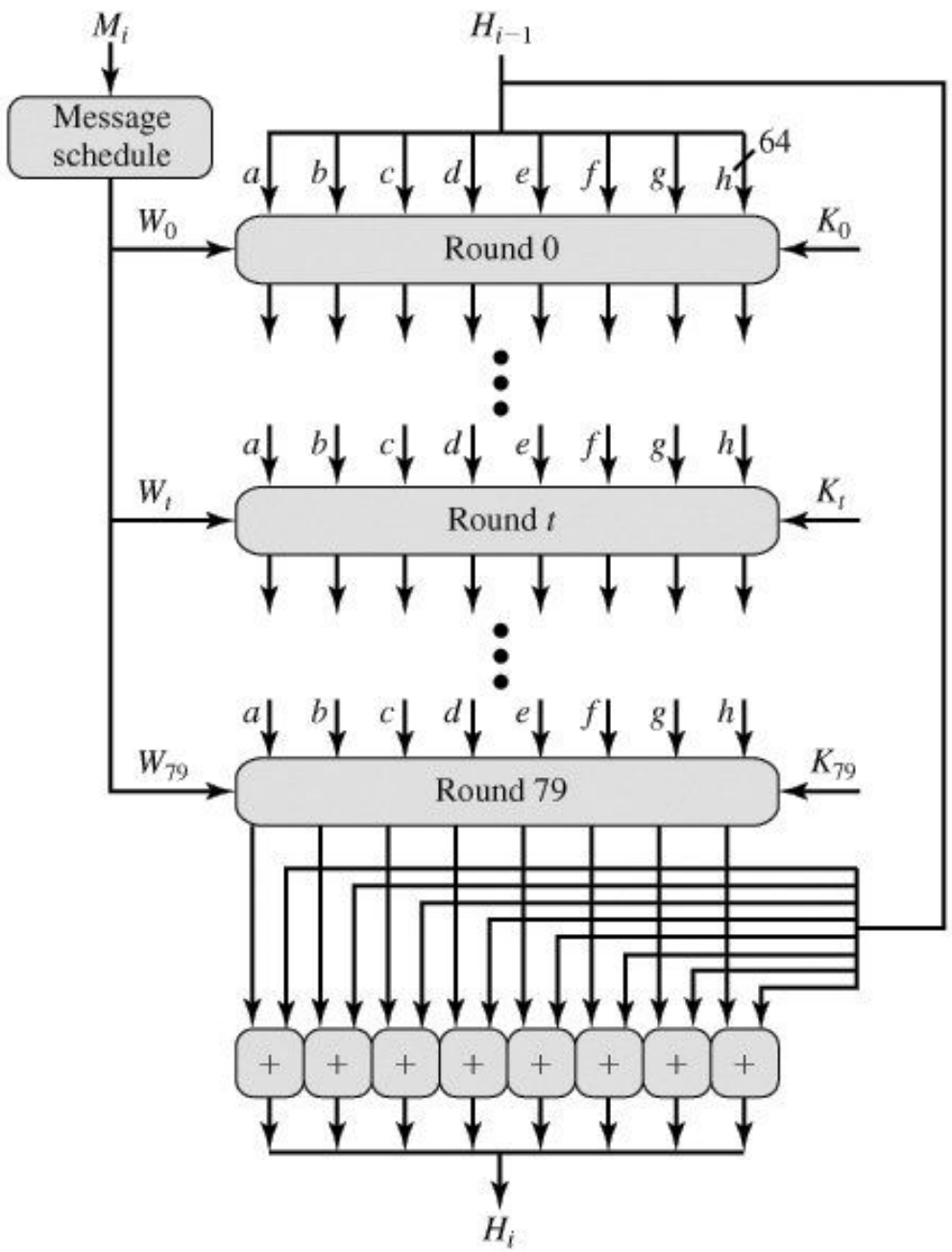
- Step 1: Append padding bits
- Step 2: Append length
- Step 3: Initialize hash buffer
- Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm
- Step 5: Output the final state value as the resulting hash

SHA-512 Overview

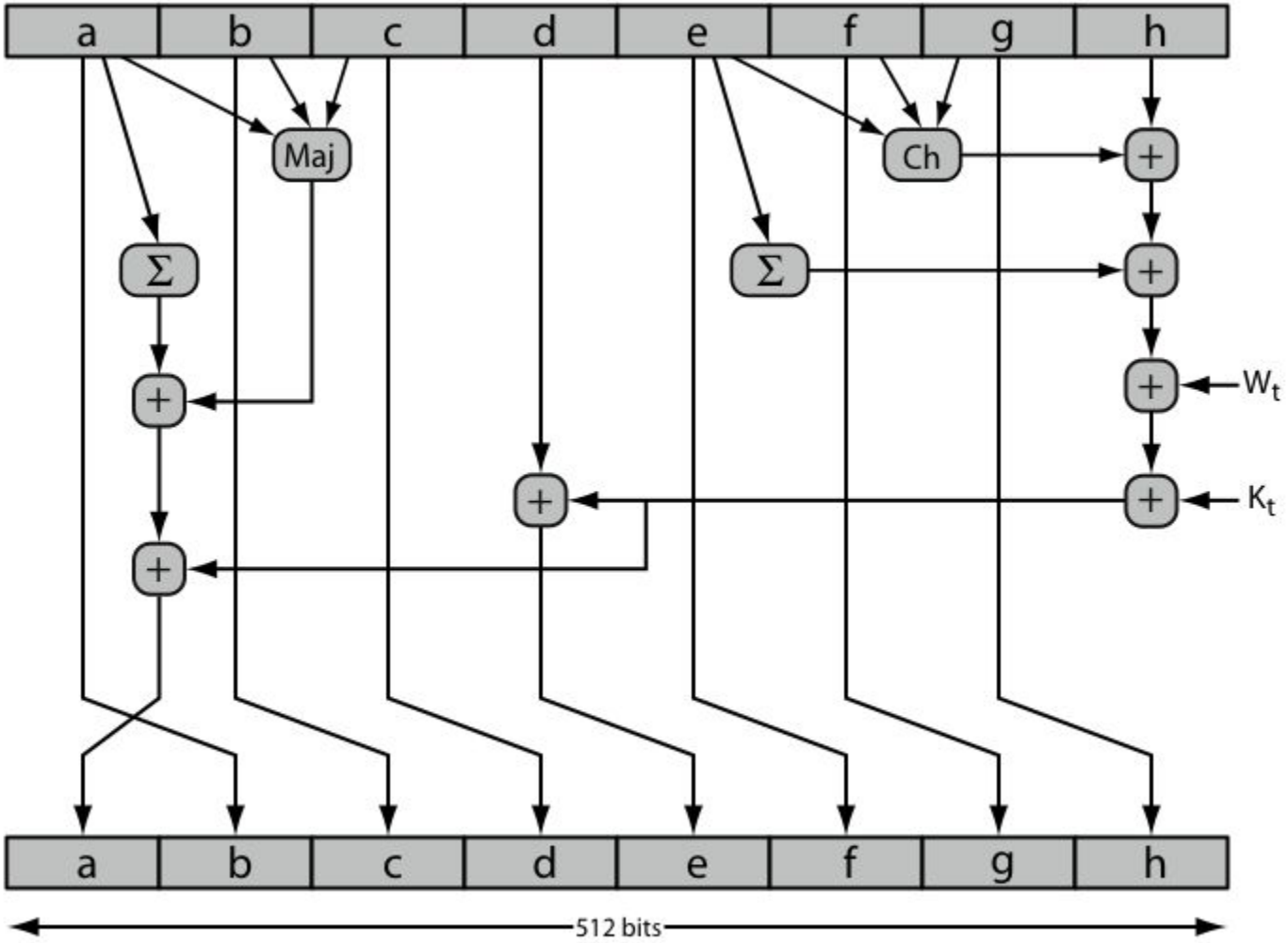


SHA-512 Compression Function

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
  - updating a 512-bit buffer
  - using a 64-bit value Wt derived from the current message block
  - and a round constant based on cube root of first 80 prime numbers


M_i
H_{i-1}
Message schedule
64
a b c d e f g h
W_0
Round 0
K_0
a b c d e f g h
W_t
Round t
K_t
a b c d e f g h
W_{79}
Round 79
K_{79}
+ + + + + + + +
H_i

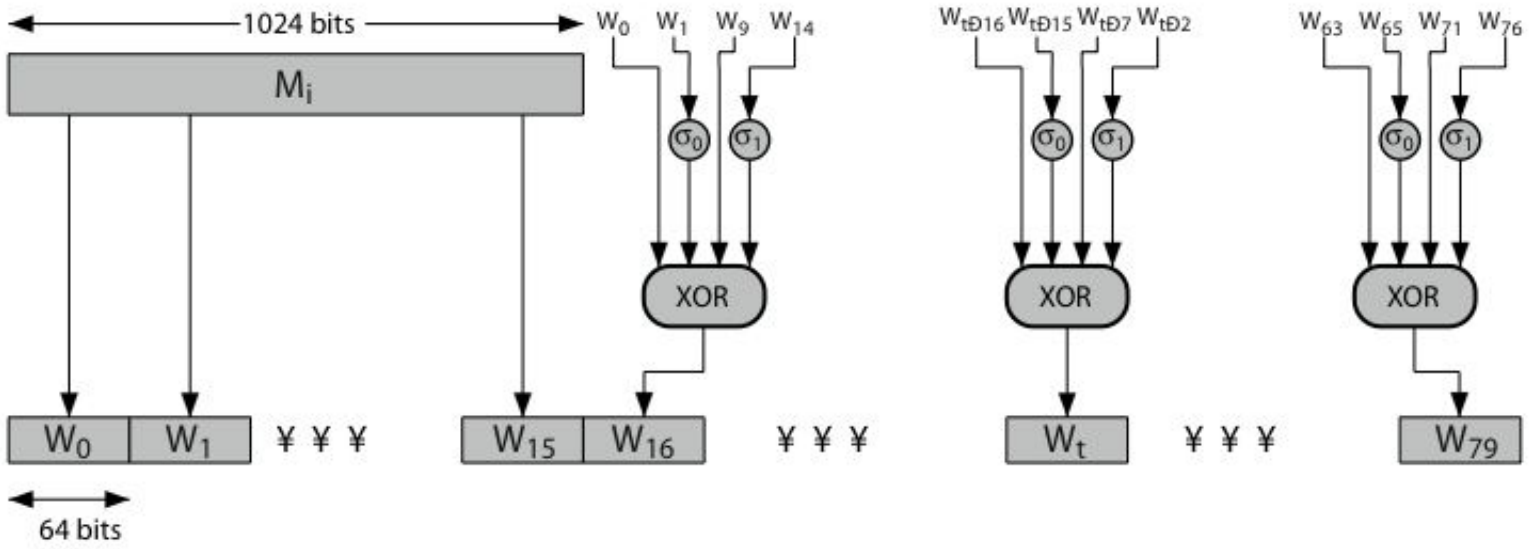
SHA-512 Round Function



Function Elements

- Ch(e,f,g) = (e AND f) XOR (NOT e AND g)
- Maj(a,b,c) = (a AND b) XOR (a AND c) XOR (b AND c)
- ∑(a) = ROTR(a,28) XOR ROTR(a,34) XOR ROTR(a,39)
- ∑(e) = ROTR(e,14) XOR ROTR(e,18) XOR ROTR(e,41)
- + = addition modulo 2^64
  - Kt = a 64-bit additive constant
  - Wt = a 64-bit word derived from the current 512-bit input block.

SHA-512 Round Function



Keyed Hash Functions as MACs

- want a MAC based on a hash function
  - because hash functions are generally faster
  - code for crypto hash functions widely available
- hash includes a key along with message
- original proposal:

  `KeyedHash = Hash(Key|Message)`

  - some weaknesses were found with this
- eventually led to development of HMAC

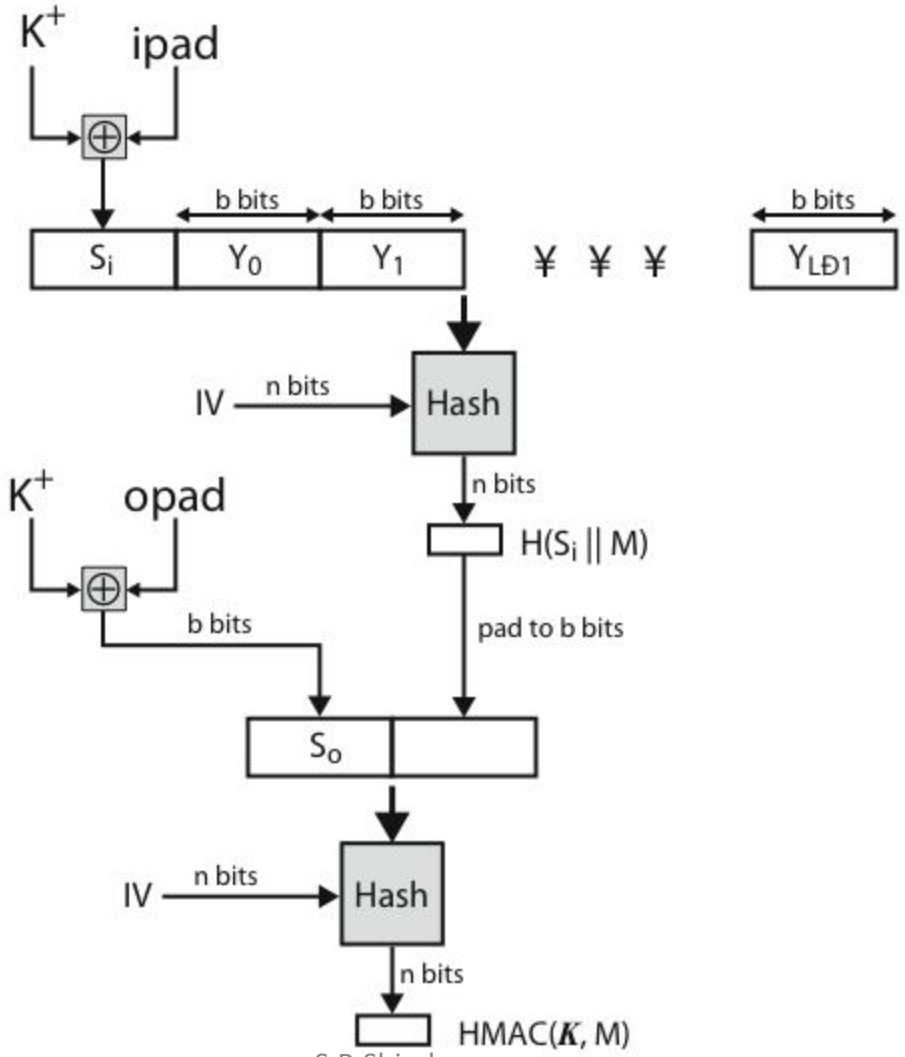
HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

$$HMAC_K = Hash[(K^+ \text{ XOR opad}) \,||\, Hash[(K^+ \text{ XOR ipad})||M)]]$$

- where K^+ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Overview



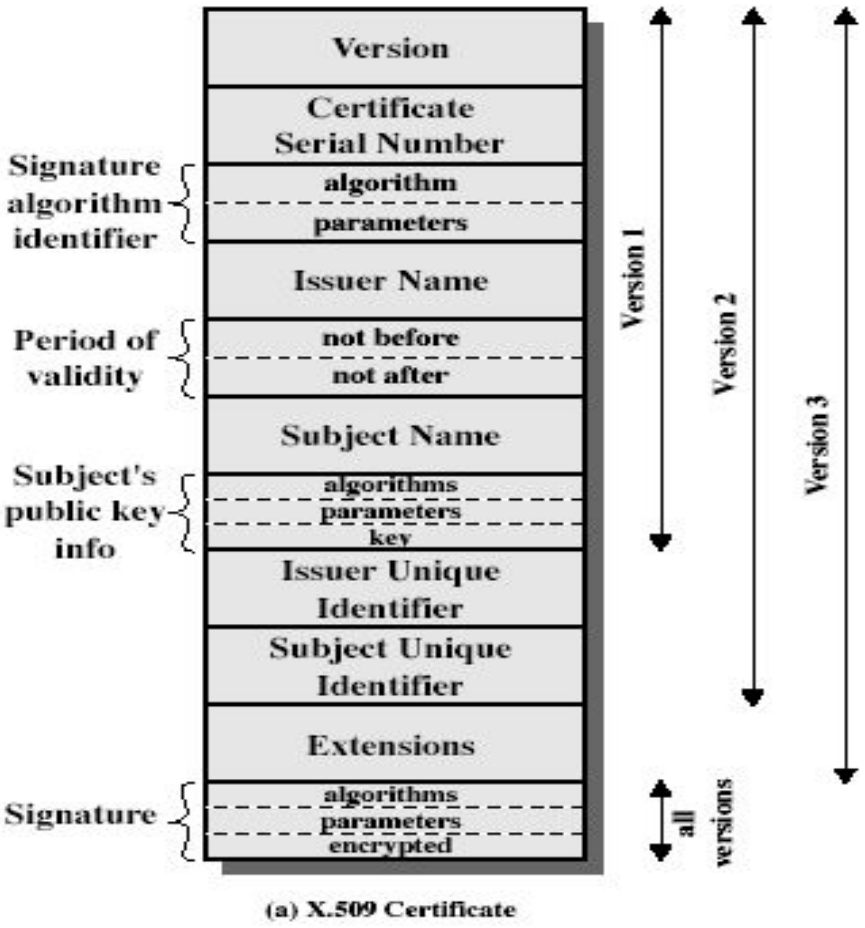
X.509 Authentication Service

- part of CCITT X.500 directory service standards
  - distributed servers maintaining some info database
- defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user
  - signed by certification authority
- also defines authentication protocols
- uses public-key crypto & digital signatures
  - algorithms not standardised, but RSA recommended

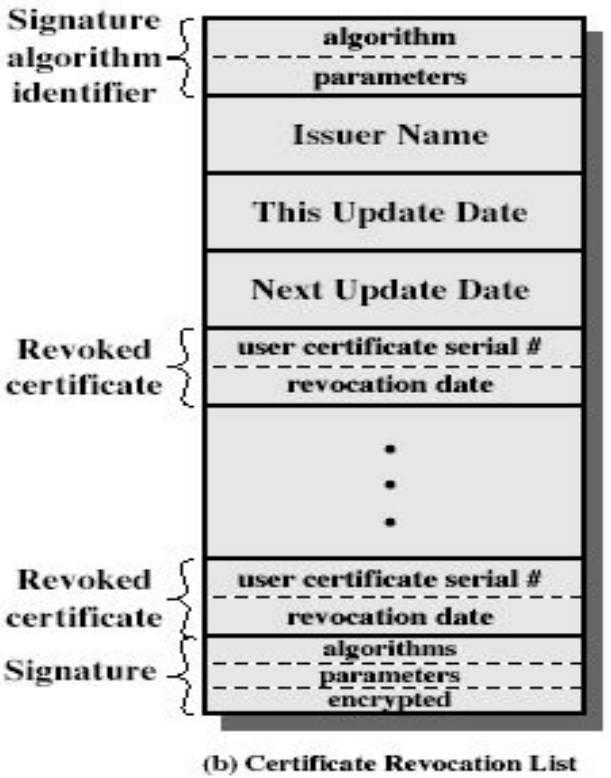
X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - version (1, 2, or 3)
  - serial number (unique within CA) identifying certificate
  - signature algorithm identifier
  - issuer X.500 name (CA)
  - period of validity (from - to dates)
  - subject X.500 name (name of owner)
  - subject public-key info (algorithm, parameters, key)
  - issuer unique identifier (v2+)
  - subject unique identifier (v2+)
  - extension fields (v3)
  - signature (of hash of all fields in certificate)
- notation `CA<<A>>` denotes certificate for A signed by CA

X.509 Certificates



(a) X.509 Certificate



(b) Certificate Revocation List

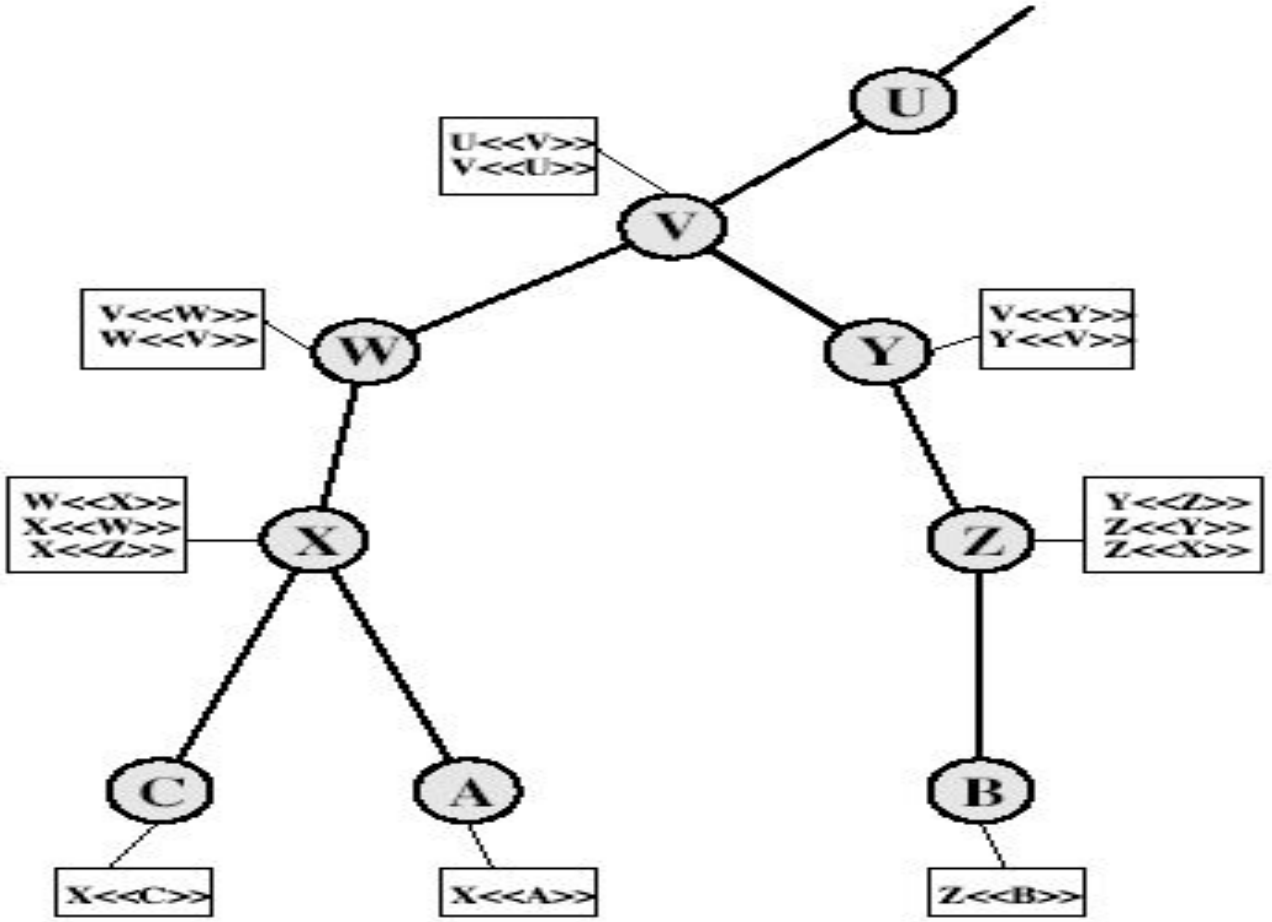
Obtaining a Certificate

- any user with access to CA can get any certificate from it
- only the CA can modify a certificate
- because cannot be forged, certificates can be placed in a public directory

CA Hierarchy

- if both users share a common CA then they are assumed to know its public key
- otherwise CA's must form a hierarchy
- use certificates linking members of hierarchy to validate other CA's
  - each CA has certificates for clients (forward) and parent (backward)
- each client trusts parents certificates
- enable verification of any certificate from one CA by users of all other CAs in hierarchy

CA Hierarchy Use



Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiry, eg:
  1. user's private key is compromised
  2. user is no longer certified by this CA
  3. CA's certificate is compromised
- CA’s maintain list of revoked certificates
  - the Certificate Revocation List (CRL)
- users should check certs with CA’s CRL

Authentication Procedures

- X.509 includes three alternative authentication procedures:
- One-Way Authentication
- Two-Way Authentication
- Three-Way Authentication
- all use public-key signatures

One-Way Authentication

- 1 message (A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply
- reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from A back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upon

X.509 Version 3

- has been recognised that additional information is needed in a certificate
  - email/URL, policy details, usage constraints
- rather than explicitly naming new fields defined a general extension method
- extensions consist of:
  - extension identifier
  - criticality indicator
  - extension value

Certificate Extensions

- key and policy information
  - convey info about subject & issuer keys, plus indicators of certificate policy
- certificate subject and issuer attributes
  - support alternative names, in alternative formats for certificate subject and/or issuer
- certificate path constraints
  - allow constraints on use of certificates by other CA's

Access Control-Introduction

- "Access control" is where security engineering meets computer science.
- Its function is to control which (active) subject have access to a which (passive) object with some specific access operation.

Access Control

- Discretionary Access Control (DAC)
  - Access Matrix Model
  - Implementation of the Access Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

Discretionary Access Control

- Access to data objects (files, directories, etc.) is permitted based on the identity of users.
- Explicit access rules that establish who can, or cannot, execute which actions on which resources.
- Discretionary: users can be given the ability of passing on their privileges to other users, where **granting** and **revocation** of privileges is regulated by an administrative policy.

Discretionary Access Control

- DAC is flexible in terms of policy specification
- This is the form of access control widely implemented in standard multi-user platforms Unix, NT, Novell, etc.

Discretionary Access Control

- Access control matrix
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix
- State of protection system
  - Describes current settings, values of system relevant to protection

Access Control

- Discretionary Access Control
  - Access Control Matrix Model
  - Implementation of the Access Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

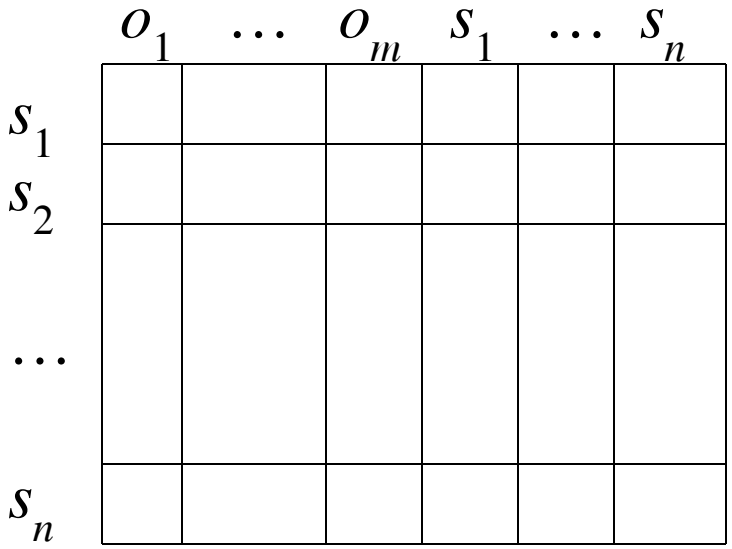
Access Control Matrix Model

- Access control matrix
  - Firstly identify the objects, subjects and actions.
  - Describes the protection state of a system.
  - State of the system is defined by a triple (S, O, A)
    - S is the set of subject,
    - O is the set of objects,
    - A is the access matrix
  - Elements indicate the access rights that subjects have on objects
    - Entry A[s, o] of access control matrix is the privilege of s on o

Description

objects (entities)

subjects



- Subjects $S = \{ s_1,\ldots,s_n \}$
- Objects $O = \{ o_1,\ldots,o_m \}$
- Rights $R = \{ r_1,\ldots,r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject s_i has rights $r_x, \ldots, r_y$ over object o_j

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Boolean Expression Evaluation

- ACM controls access to database fields
  - Subjects have attributes
  - Action/Operation/Verb define type of access
  - Rules associated with objects, action pair
- Subject attempts to access object
  - Rule for object, action evaluated, grants or denies access

Example

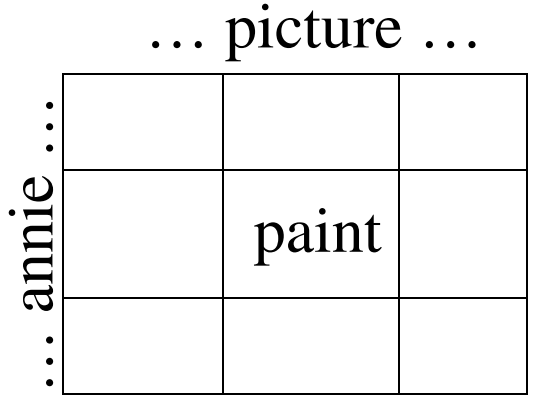
- Subject Annie
  - Attributes role (artist), groups (creative)
- Verb paint
  - Default 0 (deny unless explicitly granted)
- Object picture
  - Rule:

    Annie paint picture if:

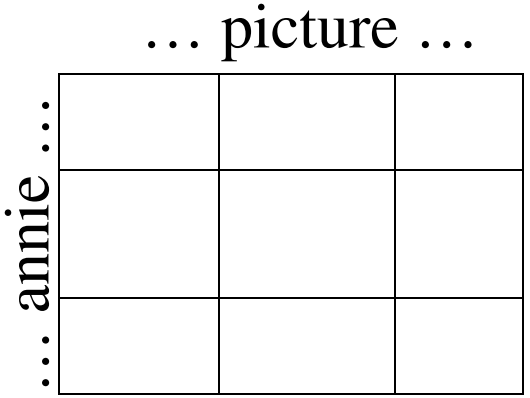
    ‘artist’ in subject.role and
    ‘creative’ in subject.groups and
    time.hour ≥ 0 and time.hour < 5

ACM at 3AM and 10AM

At 3AM, time condition met; ACM is:



At 10AM, time condition not met; ACM is:



Access Controlled by History

- Statistical databases need to
  - answer queries on groups
  - prevent revelation of individual records
- Query-set-overlap control
  - Prevent an attacker to obtain individual piece of information using a set of queries C
  - A parameter *r* (=2) is used to determine if a query should be answered

Name	Position	Age	Salary
Alice	Teacher	45	40K
Bob	Aide	20	20K
Cathy	Principal	37	60K
Dilbert	Teacher	50	50K
Eve	Teacher	33	50K

Access Controlled by History

- Query 1:
  - sum_salary(position = teacher)
  - Answer: 140K
- Query 2:
  - sum_salary(age > 40 & position = teacher)
  - Should not be answered as Matt's salary can be deduced
- Can be represented as an ACM

Name	Position	Age	Salary
Celia	Teacher	45	40K
Leonard	Teacher	50	50K
Matt	Teacher	33	50K

Name	Position	Age	Salary
Celia	Teacher	45	40K
Leonard	Teacher	50	50K

Solution: Query Set Overlap Control (Dobkin, Jones & Lipton '79)

- Query valid if intersection of *query coverage* and *each previous query* < *r*
- Can represent as access control matrix
  - Subjects: entities issuing queries
  - Objects: *Powerset* of records
  - $O_s(i)$: objects referenced by *s* in queries *1..i*
  - M[*s,o*] = read iff

$$\forall_{q \in O_s(i-1)} \left| q \cap o \right| < r$$

M[s,o] = read iff $\forall_{q \in O_s(i-1)} |q \cap o| < r$

- Query 1: O_1 = {Celia, Leonard, Matt} so the query can be answered. Hence
  - M[asker, Celia] = {read}
  - M[asker, Leonard] = {read}
  - M[asker, Matt] = {read}

- Query 2: O_2 = {Celia, Leonard} but $| O_2 \cap O_1 |$ = 2; so the query cannot be answered
  - M[asker, Celia] = ∅
  - M[asker, Leonard] = ∅

Access Control

- Discretionary Access Control
  - Access Matrix Model
  - Implementation of the Access Control Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

ACM Implementation

- ACM is an **abstract** model
  - Rights may vary depending on the object involved
- ACM is implemented primarily in three ways
  - Authorization Table
  - Capabilities (rows)
  - Access control lists (columns)

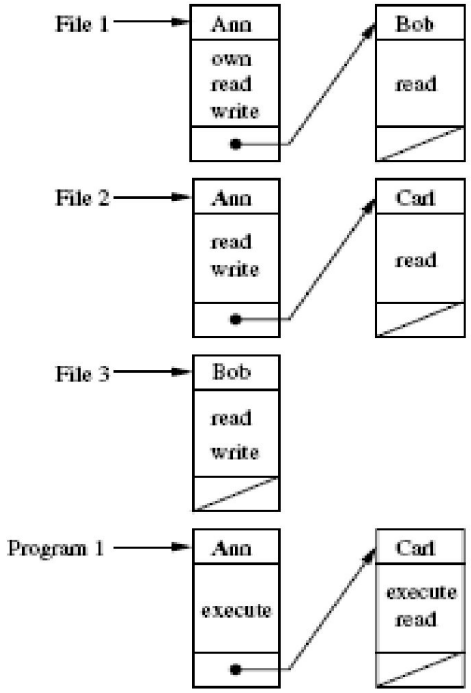
Authorization Table

- Three columns: subjects, actions, objects
- Generally used in DBMS systems

User	Access mode	Object
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 3
Bob	write	File 3
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

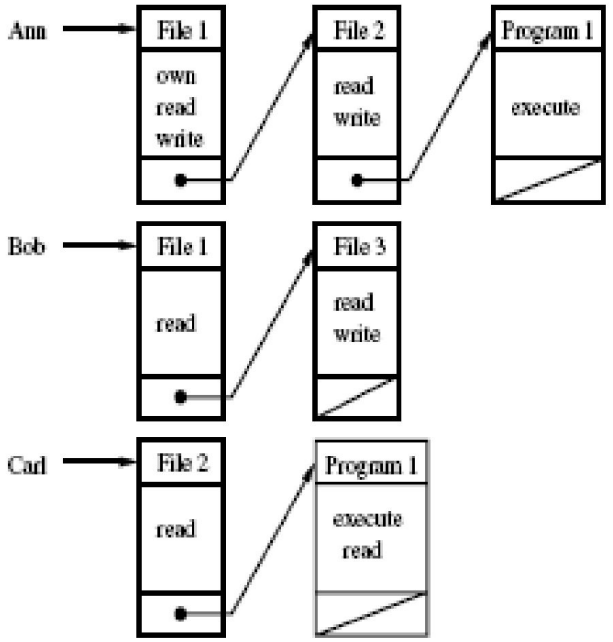
Access Control List (ACL)

- Matrix is stored by column.
- Each object is associated with a list
- Indicate for each subject the actions that the subject can exercise on the object



Capability List

- Matrix is stored by row
- Each user is associated with a capability list
- Indicating for each object the access that the user is allow to exercise on the object



ACLs vs Capability List

- Immediate to check the authorization holding on an object with ACLs. (subject?)
- Immediate to determine the privileges of a subject with Capability lists. (object?)
- Distributed system,
  - authenticate once, access various servers
  - choose which one?
- Limited number of groups of users, small bit vectors, authorization specified by owner.
  - Which one?

Basic Operations in Access Control

- Grant permissions
  - Inserting values in the matrix's entries
- Revoke permissions
  - Remove values from the matrix's entries
- Check permissions
  - Verifying whether the entry related to a subject *s* and an object *o* contains a given access mode

Access Control

Discretionary Access Control

- Access Matrix Model
- State of Protection System
- Implementation of the Access Matrix
- Vulnerabilities of the Discretionary Policies
- Additional features of DAC

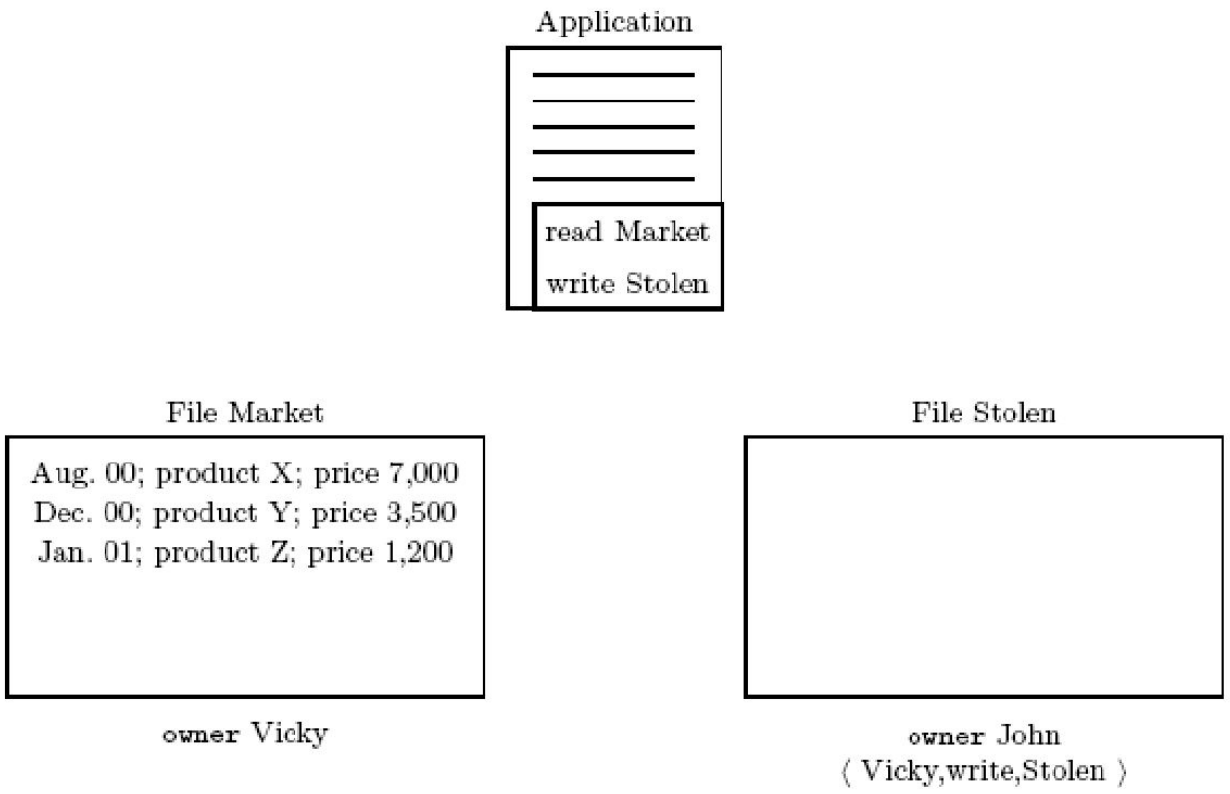
Vulnerabilities of the Discretionary Policies

- No separation of users from subjects
- No control on the flow the information
- Malicious code, i.e., Trojan horse

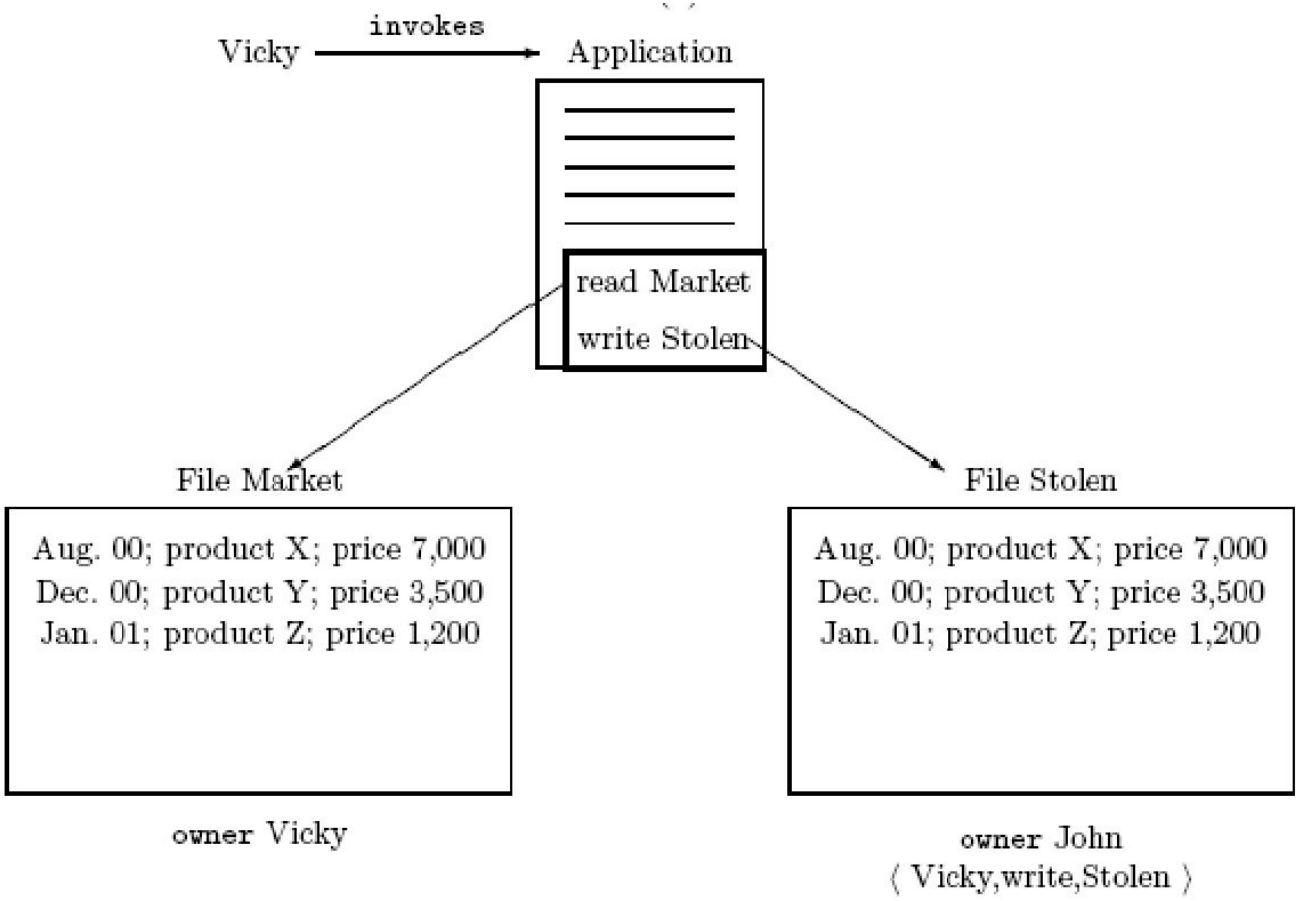
Example

- Vicky, a top-level manager
- A file Market on the new products release
- John, subordinate of Vicky
- A file called “Stolen”
- An application with two hidden operations
  - Read operation on file Market
  - Write operation on file Stolen

Example (cond)



Example (cond)



- Restriction should be enforced on the operations that processes themselves can **execute**.
- Mandatory policies provide a way to enforce **information flow control** through the use of labels

Access Control

- Discretionary Access Control
  - Access Matrix Model
  - State of Protection System
  - Implementation of the Access Matrix
  - Vulnerabilities of the Discretionary Policies
  - Additional features of DAC

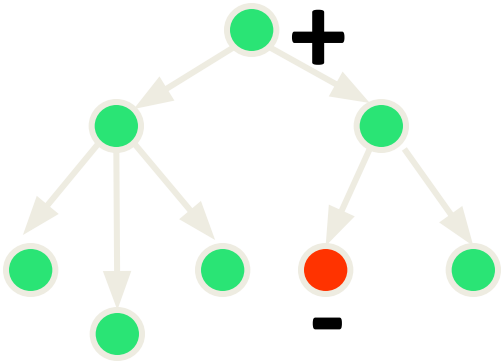
DAC – additional features and recent trends

- Flexibility is enhanced by supporting different kinds of permissions
  - Positive vs. negative
  - Strong vs. weak
  - Implicit vs. explicit
  - Content-based

Positive and Negative Permissions

- Positive permissions Give access
- Negative permissions Deny access
- Useful to specify exceptions to a given policy and to enforce stricter control on particular crucial data items

Positive and Negative Permissions



Main Issue: Conflicts

Authorization Conflicts

- Main solutions:
  - No conflicts
  - Negative permissions take precedence
  - Positive permissions take precedence
  - Nothing take precedence
  - Most specific permissions take precedence

Weak and Strong Permissions

- Strong permissions cannot be overwritten
- Weak permissions can be overwritten by strong and weak permissions

Implicit and Explicit Permissions

- Some models support implicit permissions
- Implicit permissions can be derived:
  - by a set of *propagation rules* exploiting the subject, object, and privilege hierarchies
  - by a set of user-defined *derivation rules*

Derivation Rules: Example

- Ann can read file F1 from a table if Bob has an explicit denial for this access
- Tom has on file F2 all the permissions that Bob has
- Derivation rules are a way to concisely express a set of security requirements

Derivation Rules

- Derivation rules are often expressed according to logic programming
- Several research efforts have been carried out to compare the expressive power of such languages
- We need languages based on SQL and/or XML

Content-based Permissions

- Content-based access control conditions the access to a given object based on its content
- This type of permissions are mainly relevant for database systems
- As an example, in a RDBMS supporting content-based access control it is possible to authorize a subject to access information only of those employees whose salary is not greater than 30K

Content-based Permissions

- Two most common approaches to enforce content-based access control in a DBMS are done:
  - by associating a predicate (or a Boolean combination of predicates) with the permission
  - by defining a *view* which selects the objects whose content satisfies a given condition, and then granting the permission on the view instead of on the basic objects

DAC models - DBMS vs OS

- Increased number of objects to be protected
- Different granularity levels (relations, tuples, single attributes)
- Protection of logical structures (relations, views) instead of real resources (files)
- Different architectural levels with different protection requirements
- Relevance not only of data physical representation, but also of their semantics

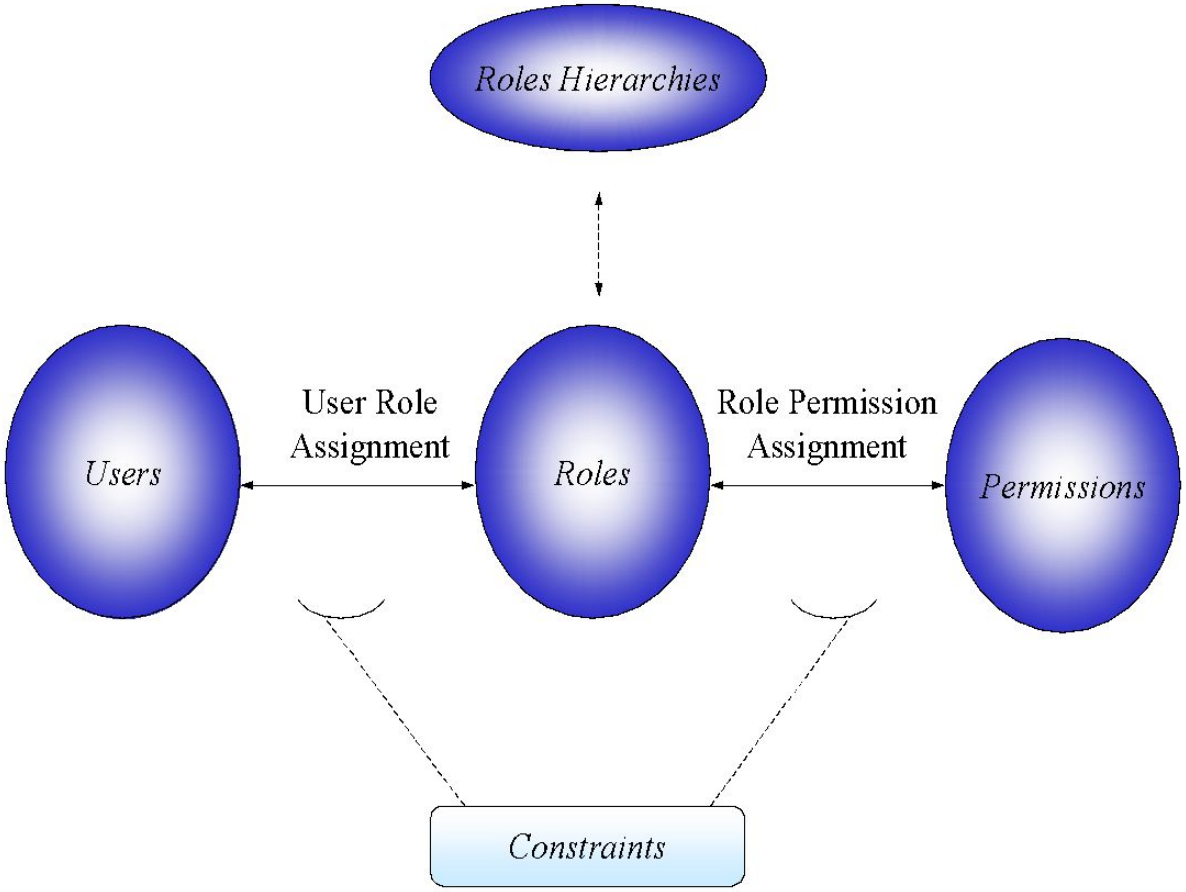
Access Control -- RBAC

RBAC

- Many organizations base access control decisions on "**the roles** that individual users take on as part of the organization".
- They prefer to centrally control and maintain access rights that reflect the organization's protection guidelines.
- With RBAC, role-permission relationships can be predefined, which makes it simple to assign users to the predefined roles.
- The combination of users and permissions tend to change over time, the permissions associated with a role are more stable.
- RBAC concept supports three well-known security principles:
  - Least privilege
  - Separation of duties
  - Data abstraction

Role Based Access Control (RBAC)

- Access control in organizations is based on "roles that individual users take on as part of the organization"
- A role is "is a collection of permissions"



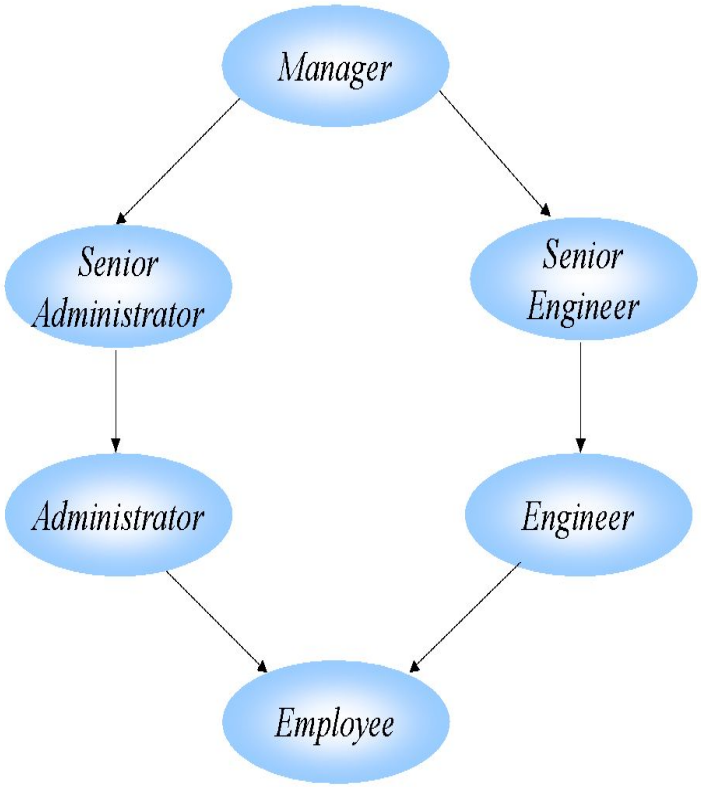
RBAC

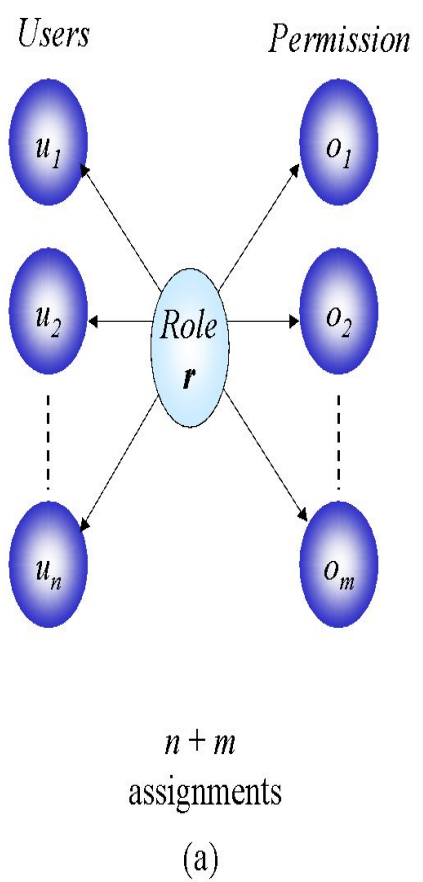
- Access depends on role/function, not identity
  - Example: Allison is **bookkeeper** for Math Dept. She has access to financial records. If she leaves and Betty is hired as the new **bookkeeper**, Betty now has access to those records. The role of "bookkeeper" dictates access, not the identity of the individual.

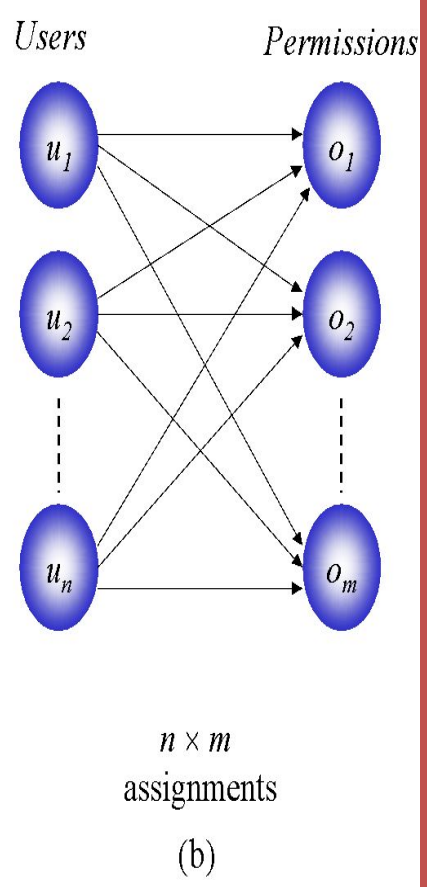
Advantages of RBAC

- Allows Efficient Security Management
  - Administrative roles, Role hierarchy
- Principle of least privilege allows minimizing damage
- Separation of Duties constraints to prevent fraud
- Allows grouping of objects
- Policy-neutral - Provides generality
- Encompasses DAC and MAC policies

RBAC

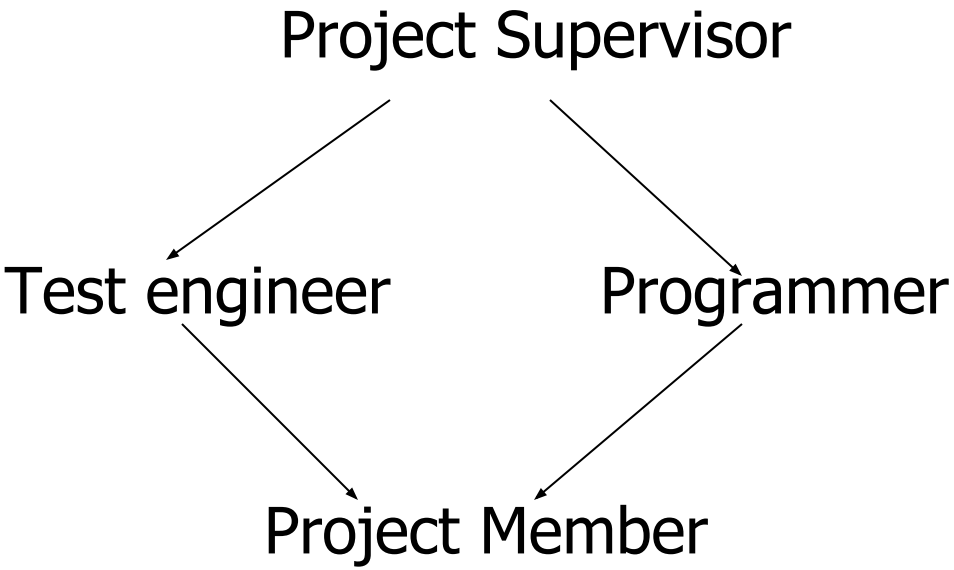

Manager
Senior Administrator
Senior Engineer
Administrator
Engineer
Employee


Users
Permission
u_1
u_2
u_n
Role
r
o_1
o_2
o_m
n + m
assignments
(a)

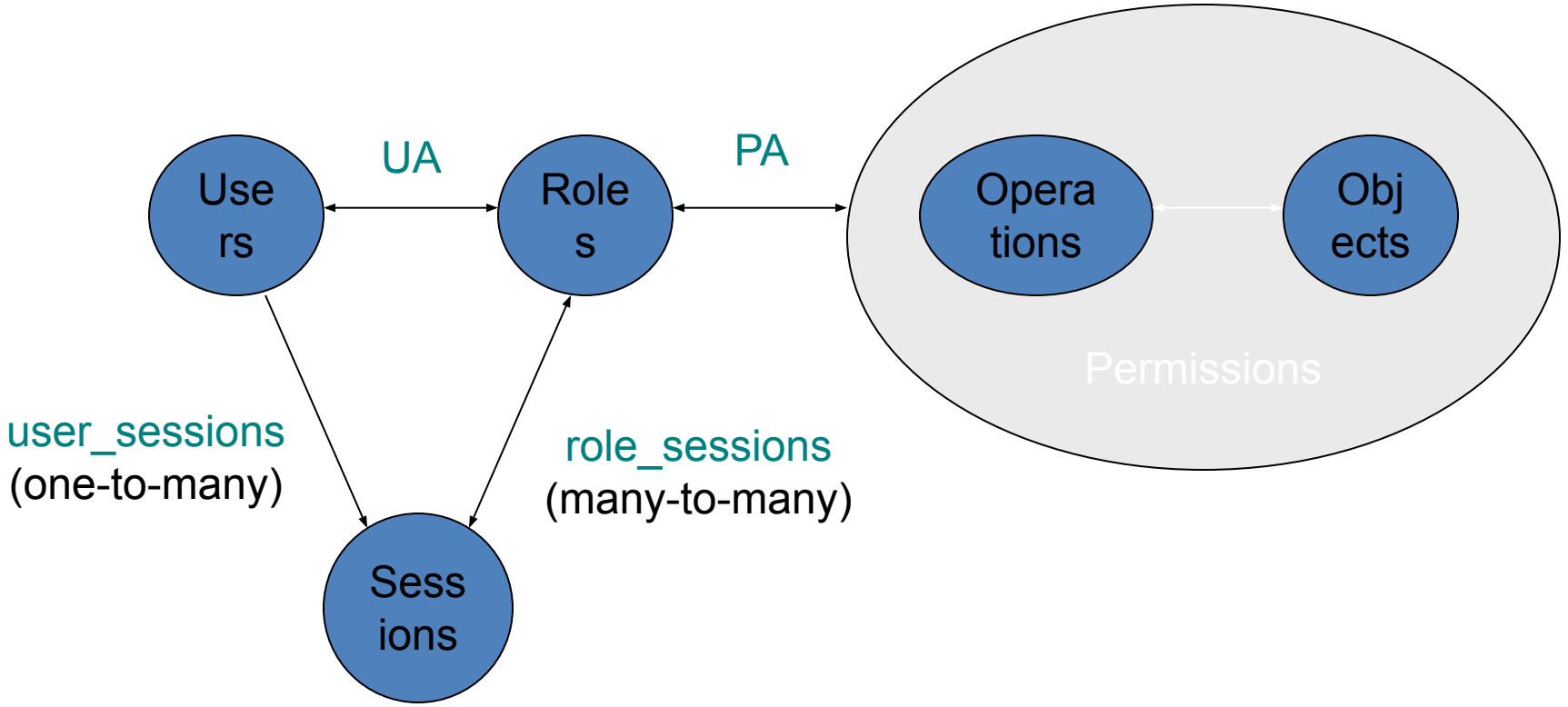

Users
Permissions
u_1
u_2
u_n
o_1
o_2
o_m
n × m
assignments
(b)

RBAC (cont'd)

- Is RBAC a discretionary or mandatory access control?
  - RBAC is policy neutral; however individual RBAC configurations can support a mandatory policy, while others can support a discretionary policy.
- Role Hierarcies
- Role Administration



RBAC (NIST Standard)

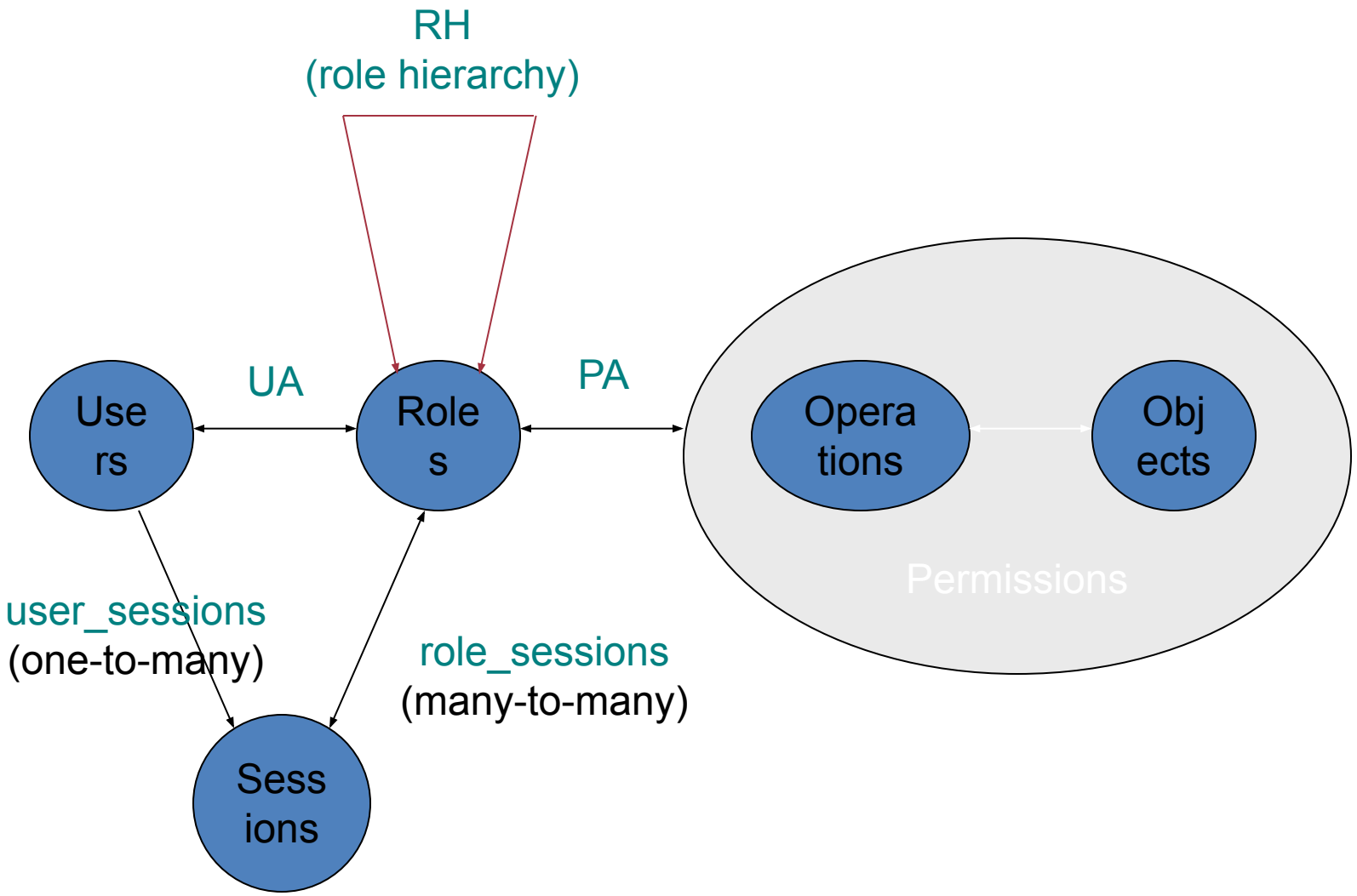


An important difference from classical models is that Subject in other models corresponds to a Session in RBAC

Core RBAC (relations)

- Permissions = $2^{\text{Operations x Objects}}$
- UA $\subseteq$ Users x Roles
- PA $\subseteq$ Permissions x Roles
- *assigned_users*: Roles $\rightarrow 2^{\text{Users}}$
- *assigned_permissions*: Roles $\rightarrow 2^{\text{Permissions}}$
- *Op*(p): set of operations associated with permission p
- *Ob*(p): set of objects associated with permission p
- *user_sessions*: Users $\rightarrow 2^{\text{Sessions}}$
- *session_user*: Sessions $\rightarrow$ Users
- *session_roles*: Sessions $\rightarrow 2^{\text{Roles}}$
  - *session_roles*(*s*) = {*r* | (session_user(*s*), *r*) $\in$ UA)}
- *avail_session_perms*: Sessions $\rightarrow 2^{\text{Permissions}}$

RBAC with General Role Hierarchy



RBAC with General Role Hierarchy

- *authorized_users*: Roles$\rightarrow 2^{\text{Users}}$

  $authorized\_users(r) = \{u \mid r' \geq r \,\&(r', u) \in UA\}$

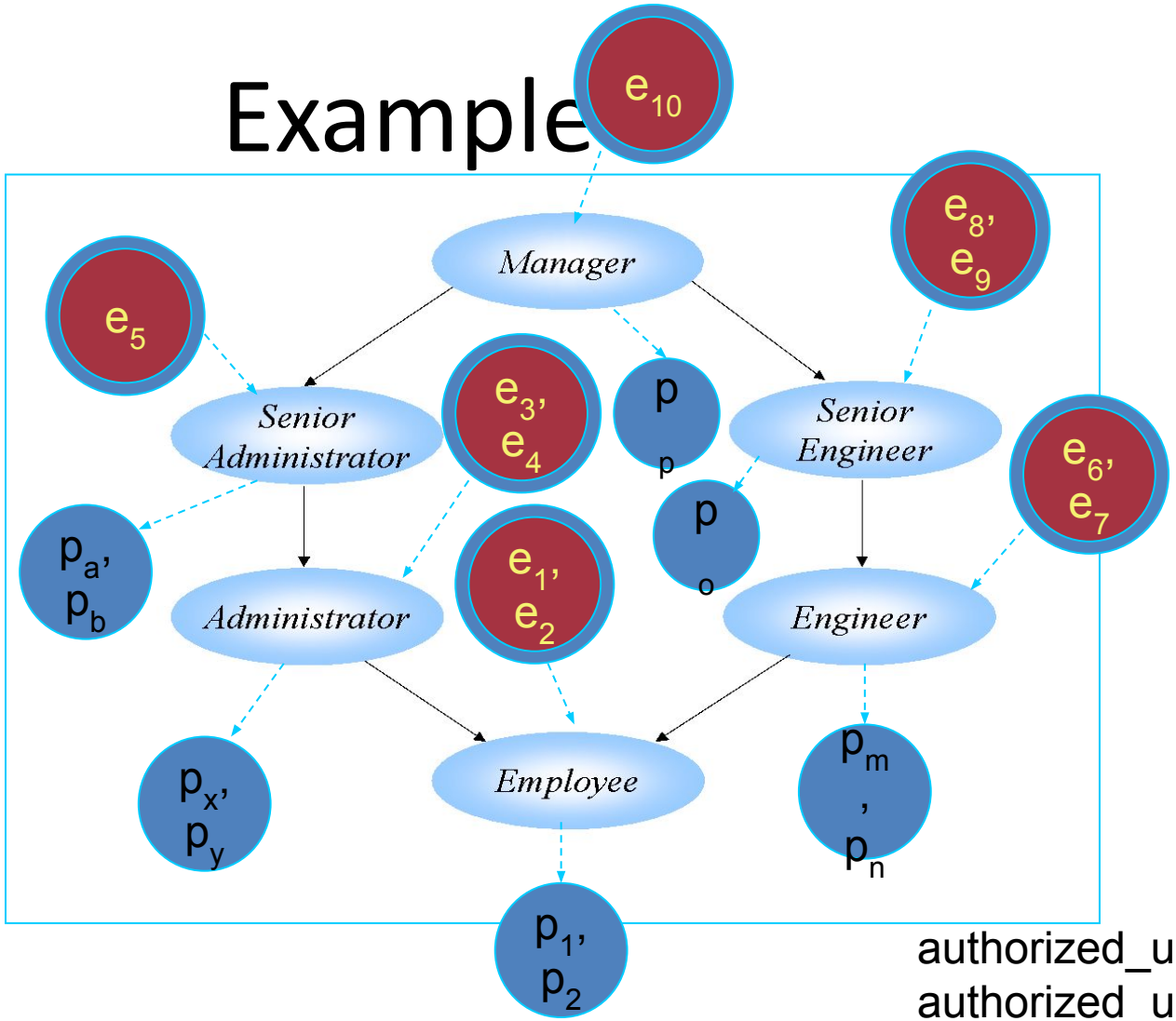
- *authorized_permissions*: Roles$\rightarrow 2^{\text{Permissions}}$

  $authorized\_users(\text{r}) = \{p \mid r' \geq r \,\&(p, r') \in PA\}$

- RH $\subseteq$ Roles x Roles is a partial order
  - called the inheritance relation
  - written as ≥.

  $(r_1 \geq r_2) \rightarrow authorized\_users(r_1) \subseteq authorized\_users(r_2)$ &
  $authorized\_permisssions(r_2) \subseteq authorized\_permisssions(r_1)$

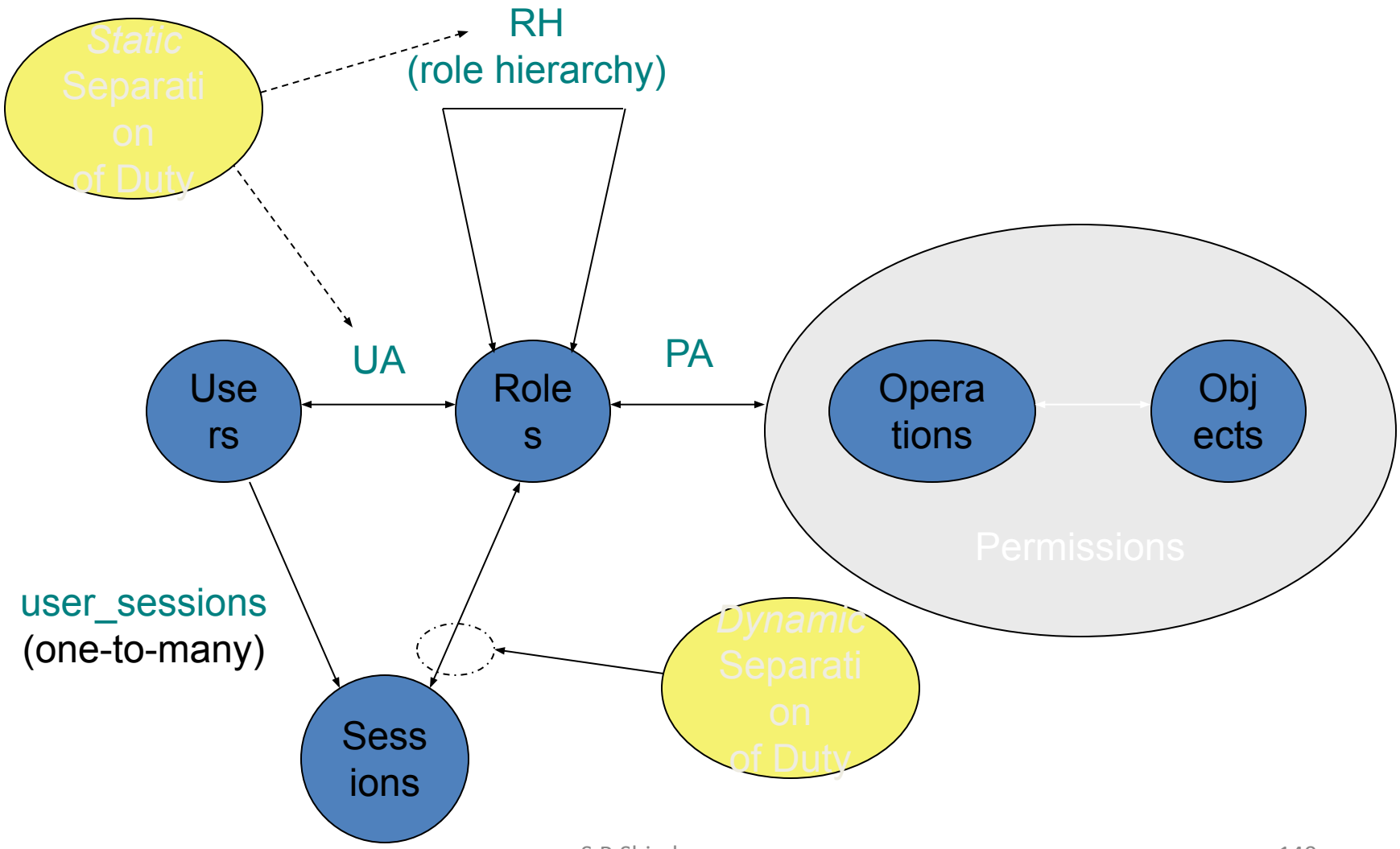
Example



authorized_users(Employee)?
authorized_users(Administrator)?

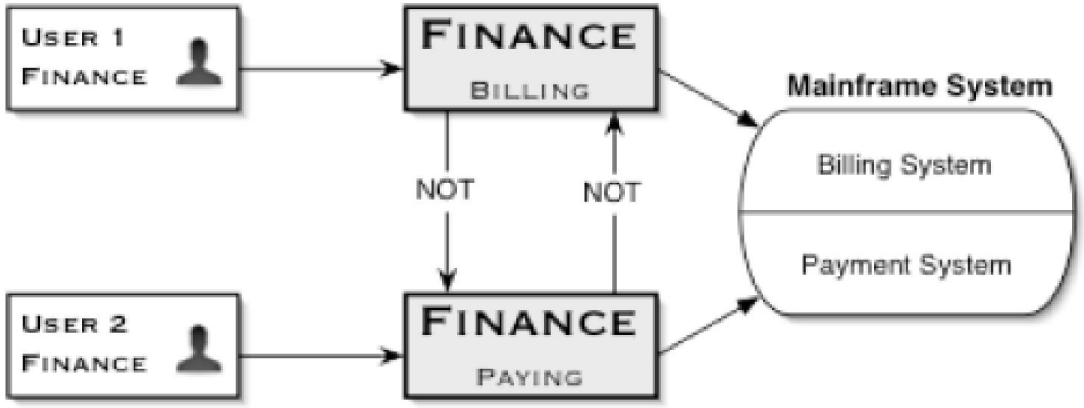
authorized_permissions(Employee)?
authorized_permissions(Administrator)?

Constrained RBAC



Separation of Duties

- No user should be given enough privileges to misuse the system on their own.
- Statically: defining the conflicting roles
- Dynamically: Enforcing the control at access time



RBAC's Benefits

TABLE 1: ESTIMATED TIME (IN MINUTES) REQUIRED FOR ACCESS ADMINISTRATIVE TASKS

TASK	RBAC	NON-RBAC	DIFFERENCE
Assign existing privileges to new users	6.14	11.39	5.25
Change existing users' privileges	9.29	10.24	0.95
Establish new privileges for existing users	8.86	9.26	0.40
Termination of privileges	0.81	1.32	0.51

Cost Benefits

- Saves about 7.01 minutes per employee, per year in administrative functions
  - Average IT admin salary - $59.27 per hour
  - The annual cost saving is:
    - $6,924/1000; $692,471/100,000
- Reduced Employee downtime
  - if new transitioning employees receive their system privileges faster, their productivity is increased
  - 26.4 hours for non-RBAC; 14.7 hours for RBAC
  - For average employee wage of $39.29/hour, the annual productivity cost savings yielded by an RBAC system:
    - $75000/1000; $7.4M/100,000

Thank You