Security

Today

- Introduction
- Secure channel
- Access control
- Security management

Security – dependability revisited

- A component provides services to clients. For this, it may require services from other components, i.e. *depend* on them.
  - Availability – accessible and usable by authorized entities
  - Reliability – continuity of service delivery
  - Safety – very low probability of catastrophes
  - Confidentiality – no unauthorized disclosure of info
  - Integrity – no accidental or malicious alterations of info
- In distributed systems, security is the combination of availability, integrity, and confidentiality. A dependable distributed system is thus fault tolerant and secure.

Security threat

- Subject – entity capable of issuing a request for service as provided by objects
- Channel – the carrier of request/replies for service
- Object – entity providing service to subjects
- Channels and objects are subject to security threats

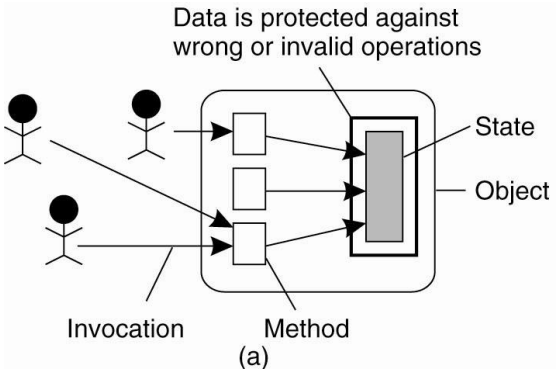
Threat	Channel	Object
Interruption	Preventing message transfer	DOS
Inspection	Reading the content of transferred messages	Reading the data contained in an object
Modification	Changing message content	Changing an object's encapsulated data
Fabrication	Inserting messages	Spoofing an object

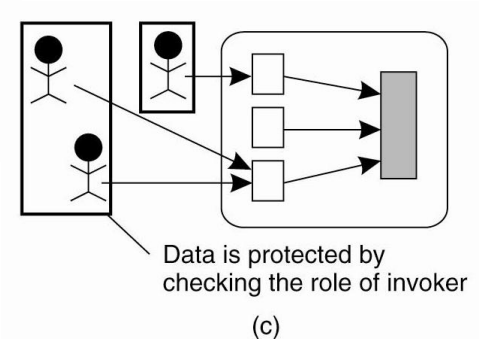
Security mechanisms

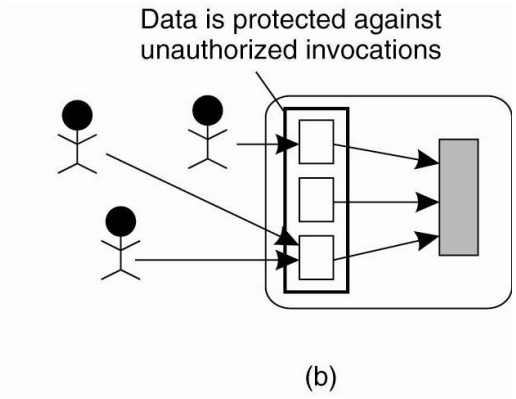
- To protect against security threats, we have a number of security mechanisms at our disposal:
  - Encryption: Transform data into something that an attacker cannot understand (confidentiality). It is also used to check whether something has been modied (integrity).
  - Authentication: Verify the claim that a subject says it is S: verifying the identity of a subject.
  - Authorization: Determining whether a subject is permitted to make use of certain services.
  - Auditing: Trace which subjects accessed what, and in which way. Useful only if it can help catch an attacker.
- Note
  - But what policy do mechanisms enforce? What actions do the entities in a systems are (not) allowed to take? First clarify the security policies

Design issues

- Focus of control – what is our focus when talking about protection: (a) data (e.g. integrity constraints), (b) invalid operations, (c) unauthorized users



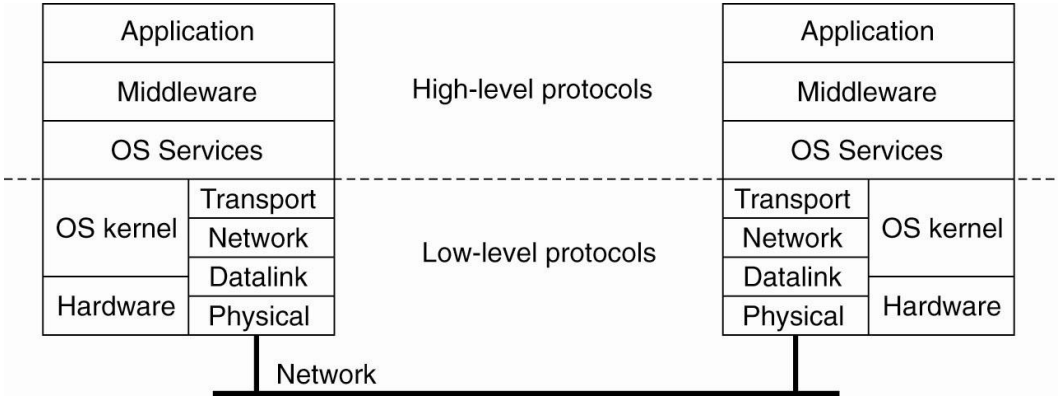




- We generally need all three, but each requires different mechanisms

Design issues

- Layering of security mechanisms – at which logical level are we going to implement security mechanisms?



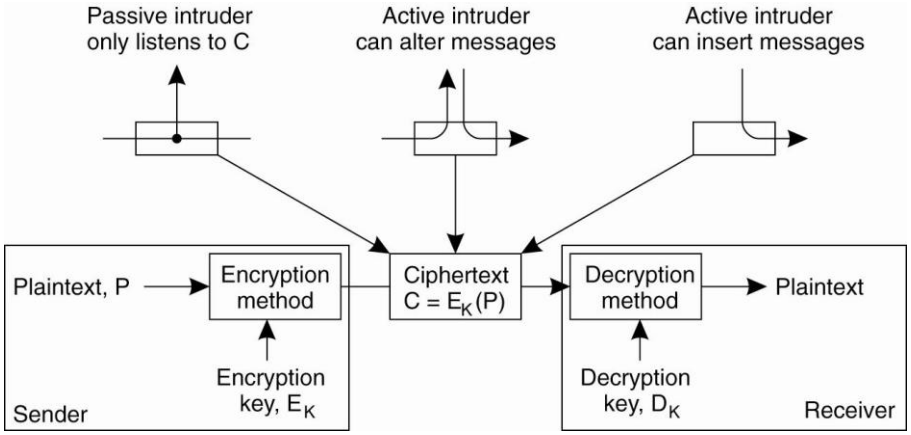
- Whether security mechanisms are actually used is related to the trust a user has in those mechanisms. No trust →implement your own mechanisms.

Design issues

- Distribution of security mechanisms
  - Trusted Computing Base: What is the set of mechanisms needed to enforce a policy. The smaller, the better.
  - Consistent with this – separate security services from other type of services and place them on machines according with needed security (e.g. RISSC, isolate security-critical servers from users)
- Simplicity
  - As always, simpler is better but not always possible
  - If application is inherently complex, little to do

Cryptography

- Fundamental to security in distributed systems
  - Symmetric system: Use a single key to encrypt & decrypt ; requires sharing the secret key
  - Asymmetric system: Use different keys for encryption & decryption – private and public
  - Hashing system: Only encrypt data (no decryption) & produce a fixed-length digest; only comparison is possible
- Basic idea for applying these techniques



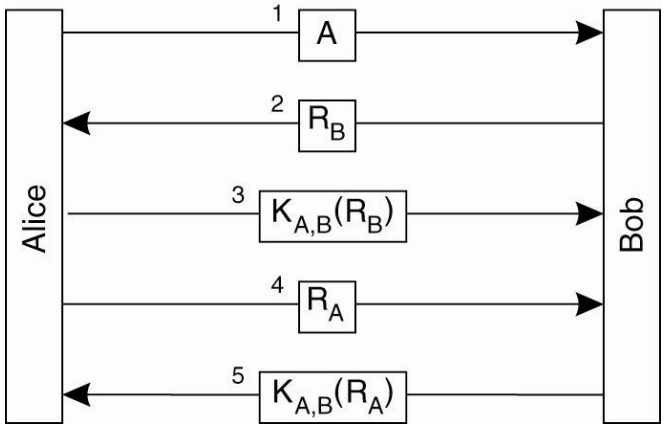
Cryptographic functions

- Hash functions used in cryptographic systems have a set of essential properties
  - One-way function: It's computationally infeasible to find the input m that corresponds to a known output h
  - Weak collision resistance: Given a pair (*m,h=H(m)),* it is computationally infeasible to find another *m* != m* such that *H(m*) = H(m)*
  - Strong collision resistance: Given only H, it is computationally infeasible to find any two different inputs *m and m** such that *H(m) = H(m*)*
- Similar properties apply to encryption functions and the keys used, also
  - It should be computationally infeasible to find the key K when given the plaintext m and associated ciphertext $C = E_K(m)$
  - Analogous to collision resistance, it is computationally infeasible to find any two different keys *K and K** such that for all *m:* $E_K(m) = E_{K^*}(m)$

Secure channels

- Making a distributed system secure
  - Authorization – controlling access to resources
  - Secure communication between processes
    - Authentication – who is on the other side
    - Message integrity – messages cannot be tampered with; authentication and message integrity rely on each other (a false message from the right person or the right message from the wrong person)
    - Message confidentiality – nor leak away
  - … setting up a secure channel

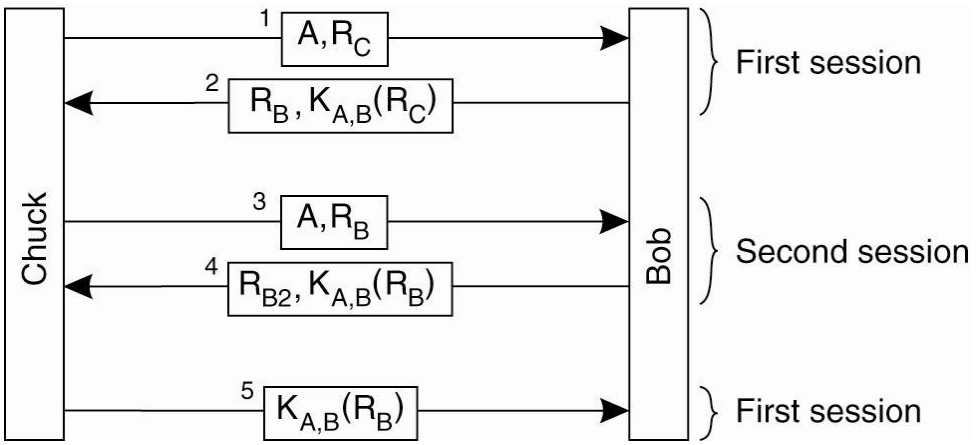
Authentication: Secret keys



1. Alice sends ID to Bob
2. Bob sends challenge R_B *(i.e.* a random number*)* to Alice
3. Alice encrypts R_B with shared key $K_{A,B}$. *Now Bob* knows he's talking to Alice
4. Alice send challenge R_A to Bob
5. Bob encrypts R_A with $K_{A,B}$. Now Alice knows she's talking to Bob

Trying to improve its performance leads to incorrect protocols (given room to reflection attacks, for example)

Secret keys 'Reflection attack'

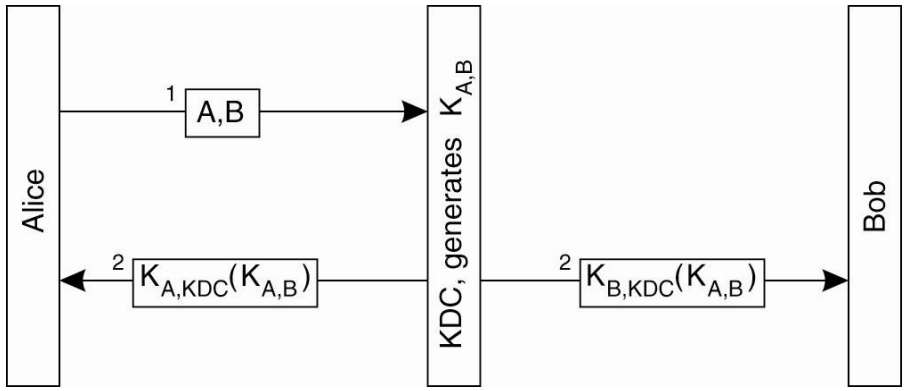


1. Chuck claims he's Alice, and sends challenge R_C
2. Bob returns a challenge R_B and the encrypted R_C
3. Chuck starts a second session, claiming he is Alice, but uses challenge R_B
4. Bob sends back a challenge, plus $K_{A,B}(R_B)$
5. Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice

Authentication: Key Distribution Center

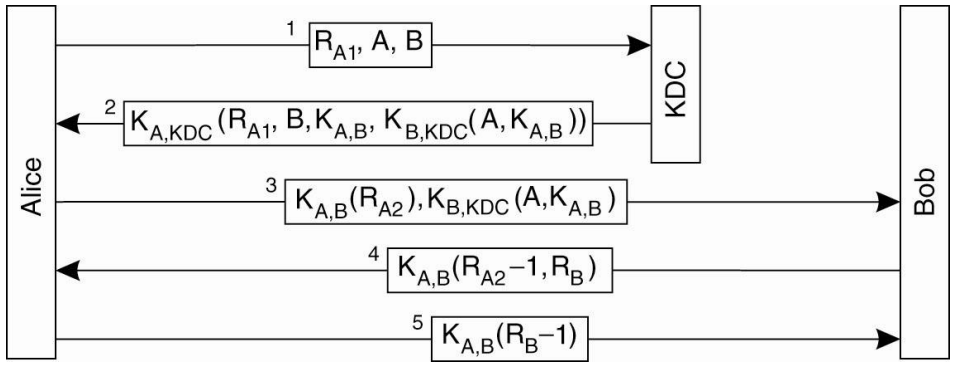
- With N subjects, we need to manage N(N - 1)/2 keys, each subject knowing N -1 keys
- Alternative – use a trusted Key Distribution Center (KDC) that generates keys when necessary
  - The KDC shared keys with each host, but no pair of hosts needs to have a share key as well
- For Alice to set a secure channel with Bob

1. Alice contact KDC stating what it wants
2. KDC returns shared key $K_{A,B}$ encrypted with key it shares with Alice …
3. sends Bob a similar message



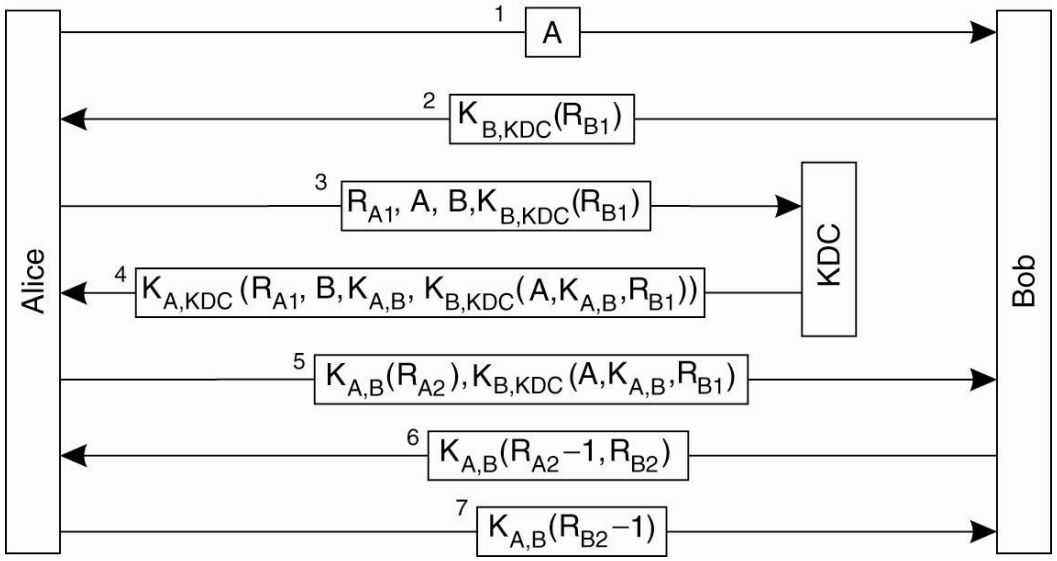
Needham-Schroeder

- Need to ensure Bob knows about $K_{A,B}$ before is reached by Alice
- Let Alice do the work and pass her a ticket to set up a secure channel with Bob – the Needham-Schroeder authentication protocol
- Some subtle issues
  - R_{A1} is a nonce, a random # used only once to related msgs (1 & 2)
  - Why Bob into the reply? To ensure Alice the channel is being set with Bob (rather than Chuck, who may intercept msg 1)
  - Bob uses the ticket to find the shared key
  - By returning R_{A2}-1, Bob shows that not only knows the shared secret key, but has actually decrypted the challenge

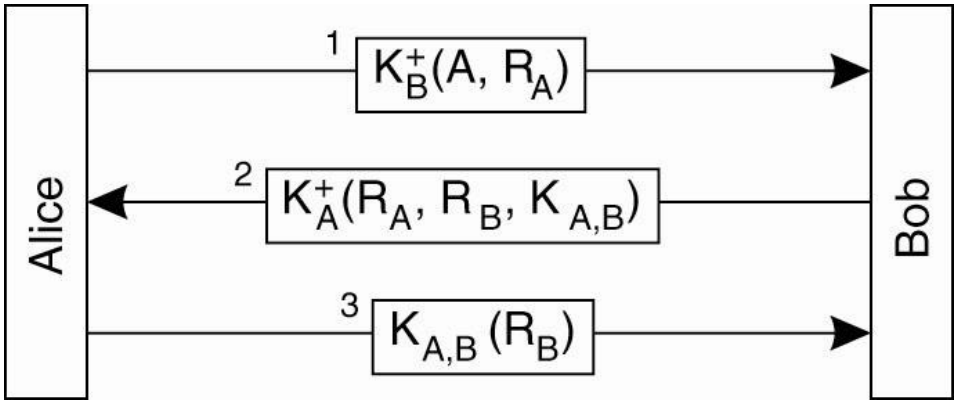


Needham-Schroeder improvement

- Security flaw – suppose Chuck finds out and old key, he could replay message 3 and have Bob set up a channel
- Need to relate message 1 to message 3 – make the key dependent on the initial request from Alice
- Solution – Alice get an encrypted number from Bob first, and put that number in the ticket provided by the KDC → Bob now knows the session key is tied to the original request to talk from Alice



Authentication: Public key



1. Alice sends a challenge R_A to Bob, encrypted with Bob's public key K^+_B .
2. Bob decrypts the message, generates a secret key $K_{A,B}$ (session key), proves he's Bob (by sending R_A back), and sends a challenge R_B to Alice. Everything's encrypted with Alice's public key K^+_A .
3. Alice proves she's Alice by sending back the decrypted challenge, encrypted with generated secret key $K_{A,B}$

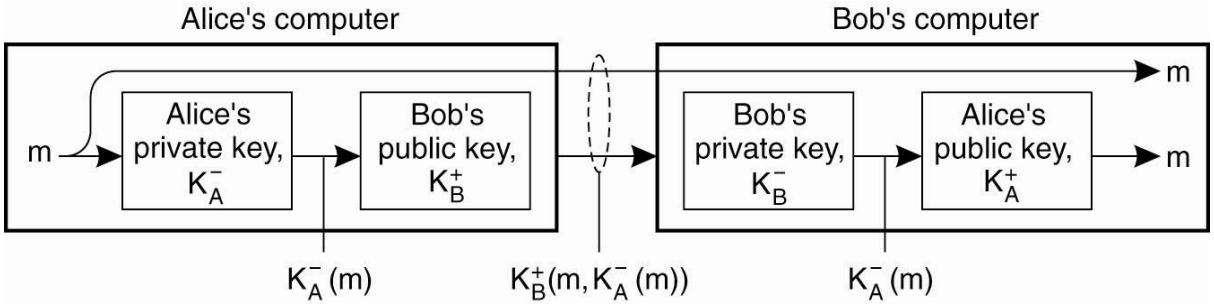
Confidentiality

- Rather than reusing keys for both authentication and message integrity and confidentiality
  - Expensive authentication keys for session establishment
  - Session keys for message integrity & confidentiality
- Why?
  - Keys wear out – the more you use it, the easier to break
  - Danger of replay – using the same key for different communication sessions, permits old messages to be inserted in the current session
  - Compromised keys – intruder can decrypt old messages
  - Temporary keys – some components you may trust once, but not long-term
    - Don't use valuable and expensive keys for all communication, but only for authentication purposes
    - Introduce a “cheap” session key is used only during one single conversation

Digital signatures

- Protecting messages integrity often goes beyond the transfer through a secure channel
  - Authentication – receiver can verify the claimed identity of the sender
  - Non-repudiation – sender can't later deny that he/she sent the message
  - Integrity – message cannot be maliciously altered during, or after receipt
- Solution: Let a sender sign all transmitted messages, in such a way that (1) the signature can be verified and (2) message and signature are uniquely associated

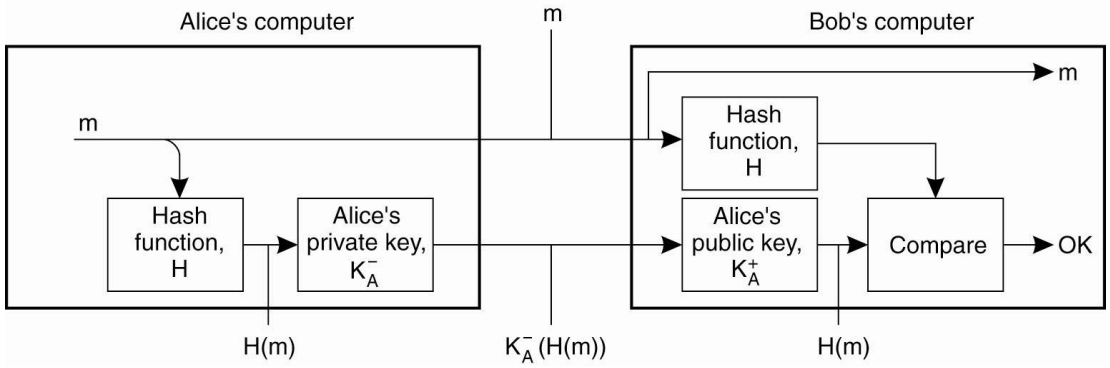
Public key signatures



1. Alice encrypts her message m with her private key
2. She then encrypts m' with Bob's public key, along with the original message and sends that
3. Bob decrypts the incoming message with his private. We know no one else has been able to read m, nor m' during their transmission.
4. Bob decrypts m' with Alice's public key to make sure it comes form Alice

Message digests

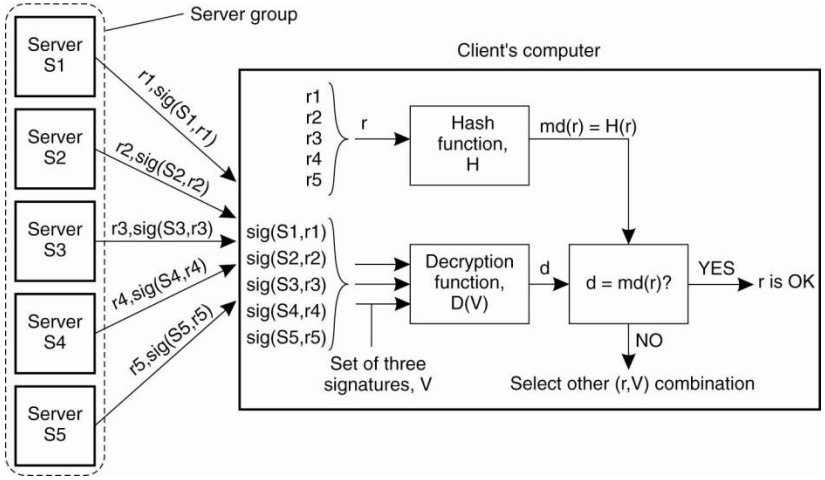
- Encrypting the whole message maybe costly and is unnecessary
- Separate authentication and secrecy, take a message digest, and sign that:



- Of course, you can still provide confidentiality by encrypting message with Bob's public key

Secure group communication

- How to share secret information between multiple members
- Confidential group communication
  - Sharing a key among all, too vulnerable to attacks
  - Use a separate key for each pair, hard to scale
  - Use public-key for communication between pairs
- Secure replicated servers
  - Client issues a request to a group of replicated servers
  - Rely on secret sharing – nobody knows the entire secret, you need c+1 signatures to create a valid signature for the response
  - At most *k* out of *N* processes can produce an incorrect answer, at most $c \leq k$ processes have been corrupted

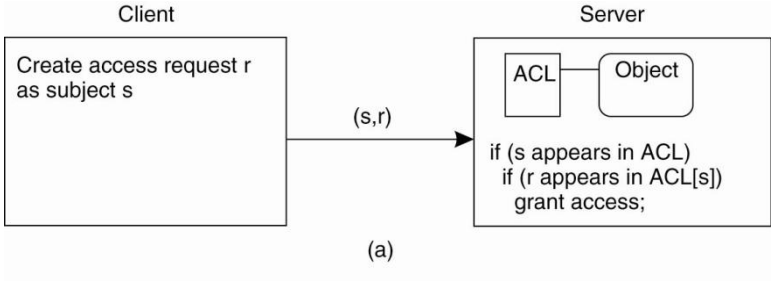


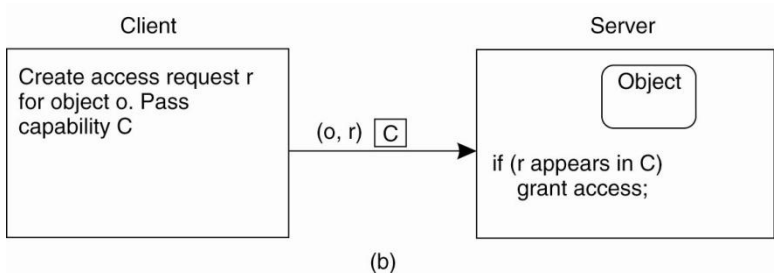
Access control

- Once there's a secure channel set up, client can issue requests to be carried out by the server
- … but on an object to which the client should have access rights
- Verifying access rights – access control
- Authentication and authorization
  - Authentication: Verify the claim that a subject says it is S: verifying the identity of a subject
  - Authorization: Determining whether a subject is permitted certain services from an object
- Authorization makes sense only if the requesting subject has been authenticated

Access control matrix

- A common approach to model access rights of subjects wrt objects – access control matrix (ACM)
- Maintain an ACM in which entry ACM[S,O] contains the permissible operations that subject S can perform on object O





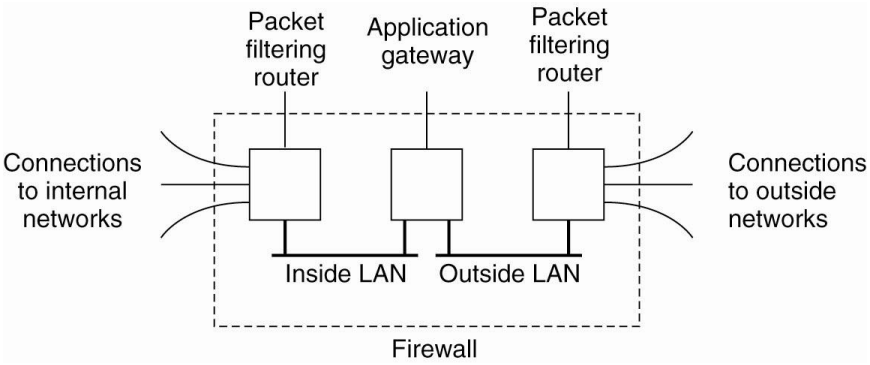
- ACM is a sparse matrix (many empty entries), to implement it
  - (a) Each object O maintains an access control list (ACL): ACM[*,O] describing the permissible operations per subject (or group of subjects)
  - (b) Each subject S has a capability: ACM[S,*] describing the permissible operations per object (or category of objects)

Protection domains

- ACLs or capability lists can be very large. Reduce information by means of protection domains:
  - Set of *(object, access rights)* pairs
  - Each pair is associated with a protection domain
  - For each incoming request the reference monitor first looks up the appropriate protection domain
- Common implementation of protection domains:
  - Groups: Users belong to a specific group; each group has associated access rights
  - Roles: Don't differentiate between users, but only the roles they can play. Your role is determined at login time. Role changes are allowed.

Firewalls

- When not all parties are playing with the same set of rules – firewalls
- A reference monitor checking on all communication
- Two basic flavors
  - Packet-filtering gateway – operates as a router, decisions made based on header info such as origin and destination ip
  - Application-level gateway – look at the payload, e.g. discard emails that are too large

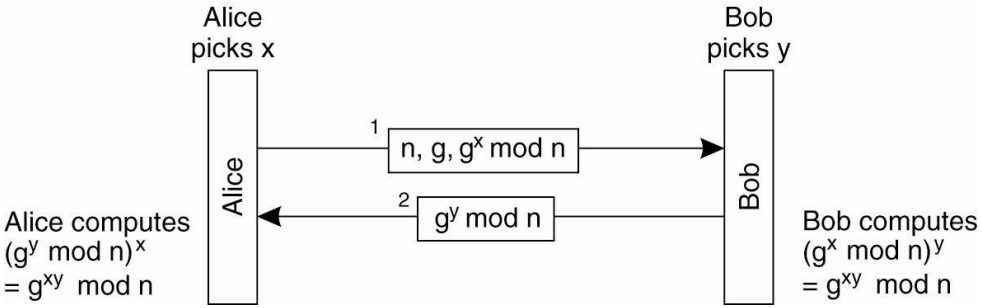


Secure mobile code

- Mobile code is great for balancing communication and computation, but
  - Hard to ensure nothing happen to the mobile code
  - The host can be sure is protected against malicious agents
- Rather than protect the client, detect changes
  - Read-only state signed by owner and easy to check
  - Append-only log that only the owner can deconstruct
  - Selective revealing to provide items to specific servers
- Protecting the host
  - Sandbox model: Remote code is allowed access to only a pre-defined collection of resources & services (by checking instructions for illegal memory access and service access)
  - Playground model: Same policy, but mechanism is to run code on separate “unprotected” machine

Management – Key establishment

- Diffie-Hellman – create secret keys in a safe way without having to trust a third party (i.e. a KDC):
  - Alice and Bob have to agree on two large numbers, *n* and *g.* Both numbers may be public.
  - Alice chooses large number *x,* and keeps it to herself. Bob does the same, say *y.*



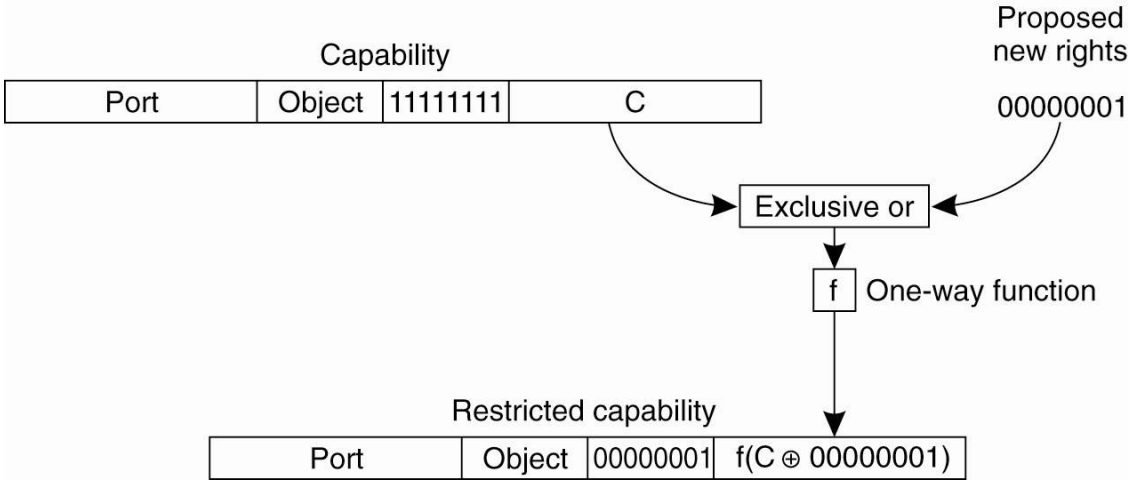
1. Alice sends (*n, g, g^x mod n) to Bob*
2. Bob sends (*g^y mod n) to Alice*
3. Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4. Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Management – Key distribution

- If authentication is based on cryptographic protocols, and we need session keys to establish secure channels, who's responsible for handing out keys?
- Secret keys: Alice and Bob will have to get a shared key. They can invent their own or trust a KDC.
- Public keys: Alice will need Bob's public key to decrypt (signed) messages from Bob, or to send private messages to Bob. But she'll have to be sure about actually having Bob's public key. Use a trusted certification authority (CA) to hand out public keys. A public key is put in a certificate, signed by a CA.

Management – Authorization management

- To avoid that each machine needs to know about all users, use capabilities and attribute certificates to express the access rights that the holder has
- In Amoeba, restricted access rights are encoded in a capability, along with data for an integrity check to protect against tampering
- Owner gets a owner capability and asks server for a restricted capability



Delegation

- A subject sometimes wants to delegate its privileges to an object O1, to allow that object to request services from another object O2
  - Have the print server to print a file w/o giving it the full copy
- Non-solution: Simply hand over your attribute certificate to a delegate (which may pass it on to the next one, etc.)
- To what extent can the object trust a certificate to have originated at the initiator of the service request, without forcing the initiator to sign every certificate?
  - Ensure that delegation proceeds through a secure channel, and let a delegate prove it got the certificate through such a path of channels originating at the initiator.

Summary

- A key principle in distributed systems and perhaps the most difficult to get right
- We have looked at basic ideas
  - Secure channels
  - Ensuring controlled access to resources
  - Security management
- Clearly, just a sample of what's out there