POLITECNICO
MILANO 1863

Security

Alessandro Margara
alessandro.margara@polimi.it
https://margara.faculty.polimi.it

Why security in a DS course?

- Sharing of resources is the motivating factor for distributed systems
- Processes encapsulate resources and provide access to them
  - Interaction between processes
- We want this interaction to be "correct"
  - Protect resources against unauthorized accesses
  - Secure processes against unauthorized interactions
  - Secure communication (messages between processes)

Introduction

- We postulate an enemy (adversary) capable of
  - Sending any message to any process
  - Reading and copying any message between a pair of processes
- Potentially any computer connected to the network
  - In an authorized way
  - In an unauthorized way
    - Through a stolen account
- Security threats from the enemy
  - Interception (sniffing, dumping)
  - Interruption (disruption, denial of service)
  - Modification (tampering, website defacing)
  - Fabrication (injection, replay attacks)

Introduction

- Security to ensure that the overall system meets these requirements
  - Availability
    - We want the system ready for use as soon as we need it
  - Reliability
    - The system should be able to run continuously for long time
  - Safety
    - The system should cause nothing bad to happen
  - Maintainability
    - The system should be easy to fix
  - Confidentiality
    - Information should be disclosed to authorized party only
  - Integrity
    - Alteration can be made only in authorized ways
      - Data alteration but also hardware and software

Introduction

- We said correct interaction, authorized access …
- … but *correct* and *authorized* with respect to what?

- We need to specify a security policy
  - Defines precisely what is allowed and what's forbidden

Introduction

- A security policy can be enforced through

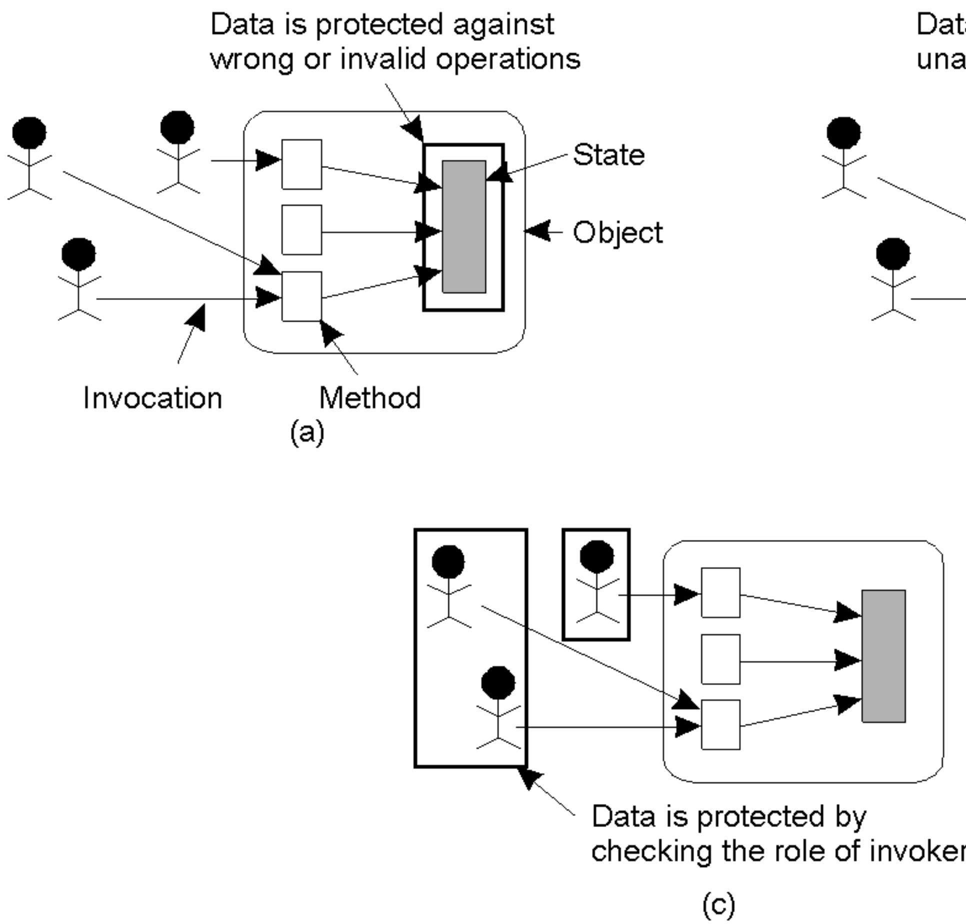
- Security mechanisms
  - Encryption (confidentiality, verify integrity)
  - Authentication (identification of parties)
  - Authorization (control information access)
  - Auditing (breach analysis, but also IDS)

Design issues

- When considering the protection of a (distributed) application, there are essentially three approaches that can be followed
  1. Protect the data, irrespective of the various operations that can possibly be performed on data items
  2. Focus on the operations that can be invoked on the data
  3. Focus on the users: only specific people have access to the application (role of the invoker)

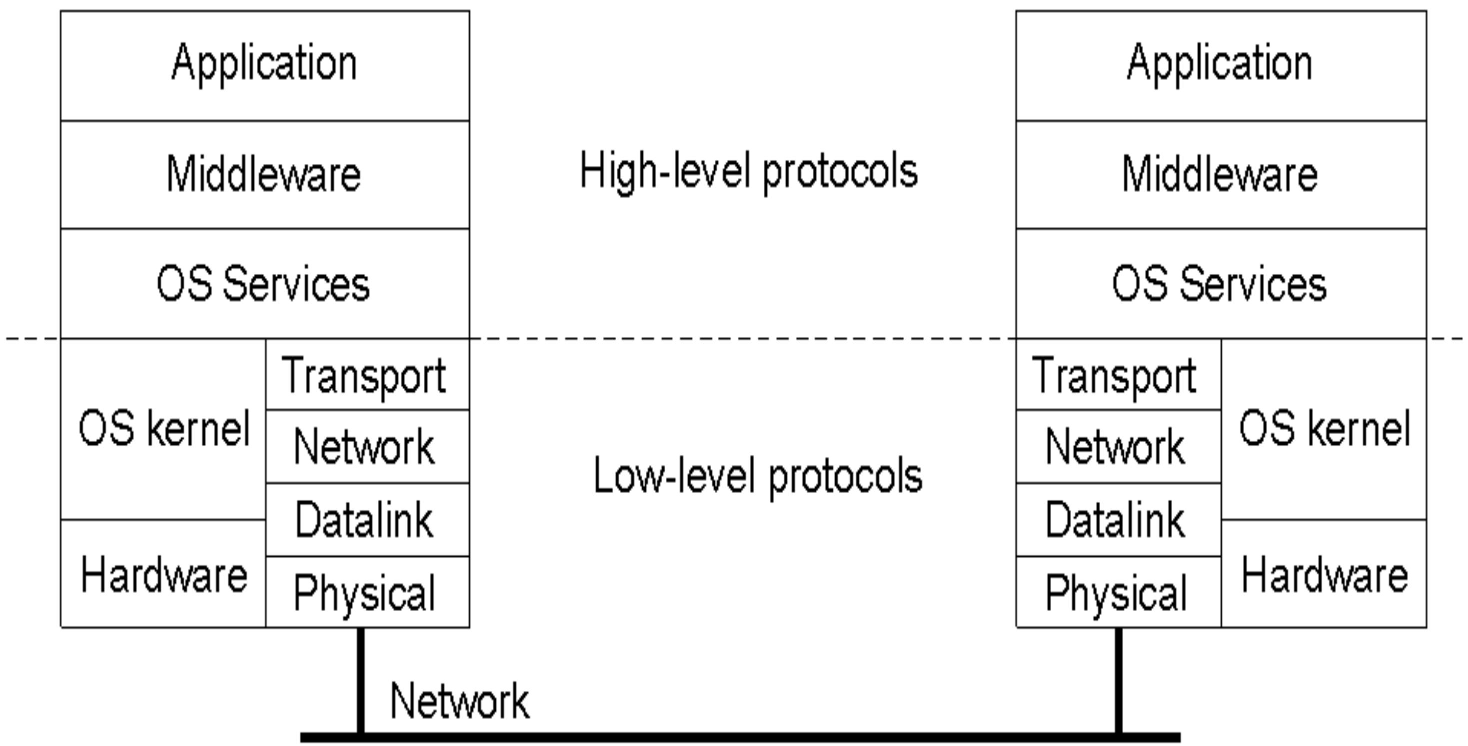
Design issues

- Focus of control



Design issues

- Layering of security mechanisms

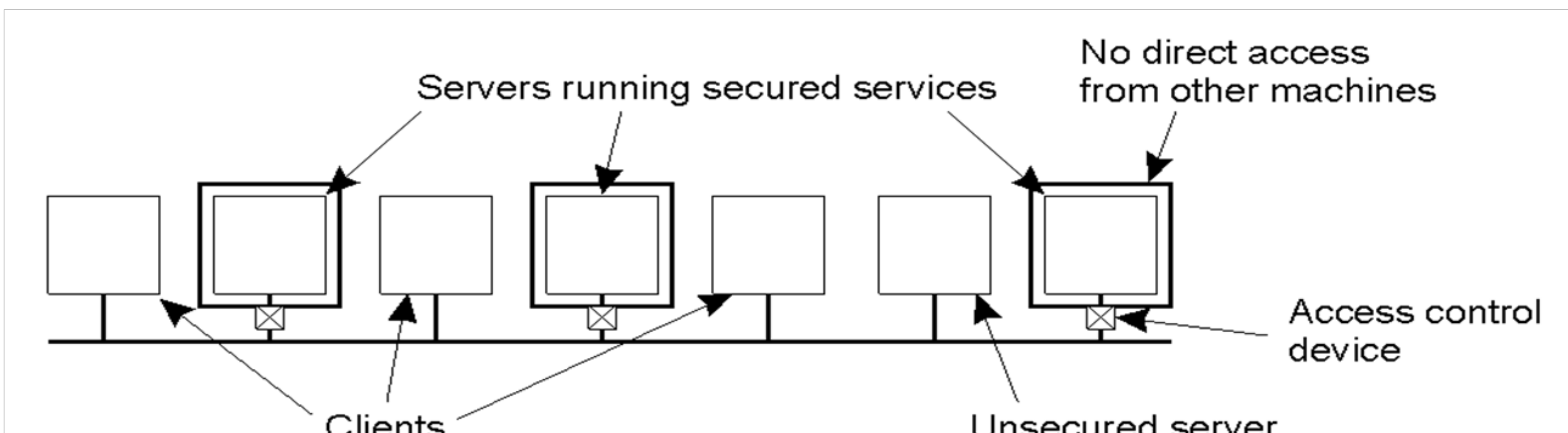


Layering of security mechanisms

- Where to put security mechanisms?
  - If you don't trust the security of a low level, you can build security mechanisms at a higher level
    - If you don't trust the transport and the lower levels, you can use SSL
- What to trust?
  - High level mechanisms might depend on lower level mechanisms
    - You need trust in local operating system (kernel at min)
  - The set of mechanisms you need to trust to enforce a given policy is called Trusted Computing Base (TCB)

TCB

- Usually TCB is composed of many systems
- A way to reduce the TCB is to separate trusted and untrusted services, and granting access to trusted services by a minimal reduced secure interface
- RISSC: Reduced Interfaces for Secure System Components



Design issues

- Simplicity
  - Simple mechanisms lead to fewer design and implementation errors
  - Unfortunately, simple mechanisms are often insufficient to build secure systems
    - Examples of this complexity also emerge at the user level
      - Firewall-related problems
      - Certificate verification
      - …
  - Often applications are complex themselves and security makes them worse

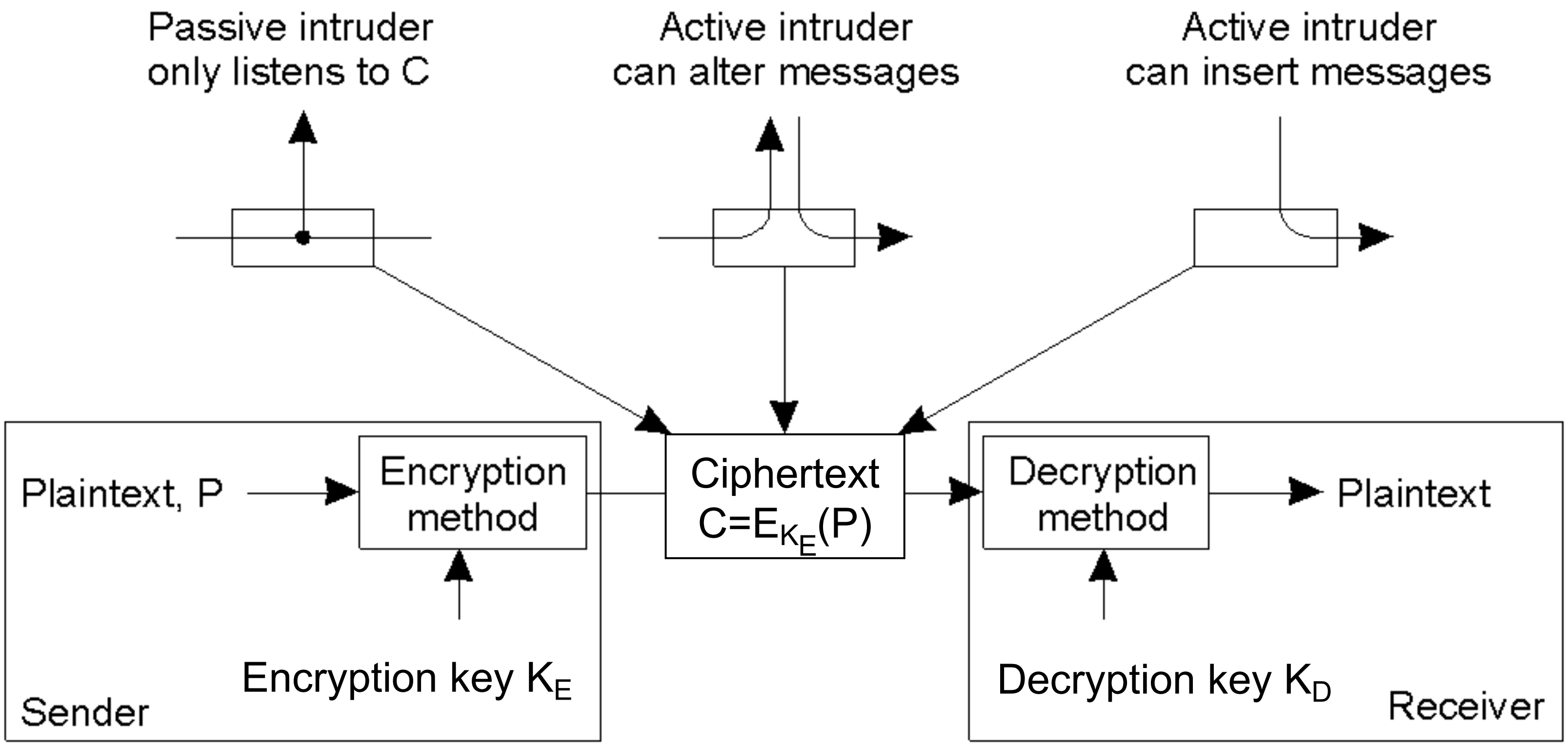
CRYPTOGRAPHY

Encryption

- Message P – encryption function→ Message C
  - P is called Plaintext, C is called Ciphertext
  - The encryption function must be invertible (decryption)
- Encryption function used to be secret
  - Security through obscurity
  - This proved to be dangerous
  - Function cannot be subject to public review
    - You must trust the inventor of the function
- Parametric encryption functions
  - From secret functions to known functions with secret parameters (keys)

$C=E_K(P)$ $P=D_K(C)$ $D_K=E_K^{-1}$

Encryption



Passive intruder only listens to C
Active intruder can alter messages
Active intruder can insert messages
Plaintext, P
Encryption method
Ciphertext $C=E_{K_E}(P)$
Decryption method
Plaintext
Encryption key K_E
Sender
Decryption key K_D
Receiver

Symmetric encryption

- If $K_E = K_D$
- We use the symbol $K_{A,B}$ to denote the key shared by A and B
- To achieve communication between N parties we need $O(N^2)$ different keys

Asymmetric encryption

- If $K_E \neq K_D$

- K_E and K_D are uniquely tied to each other (form a pair)
  - K_D can decrypt only from K_E
  - K_E can encrypt only for K_D
  - Computing K_E from K_D or vice-versa must be computationally infeasible
  - We can distribute one key without danger for the other

- We can safely distribute K_E to everyone who is interested in sending messages to me (while keeping K_D private)

Asymmetric encryption

- We can also reverse things: distribute K_D and keep K_E private
  - This is how e-signing is done

- We call the two keys
  - K^+_A (A's public key)
  - K^-_A (A's private key)

Hash

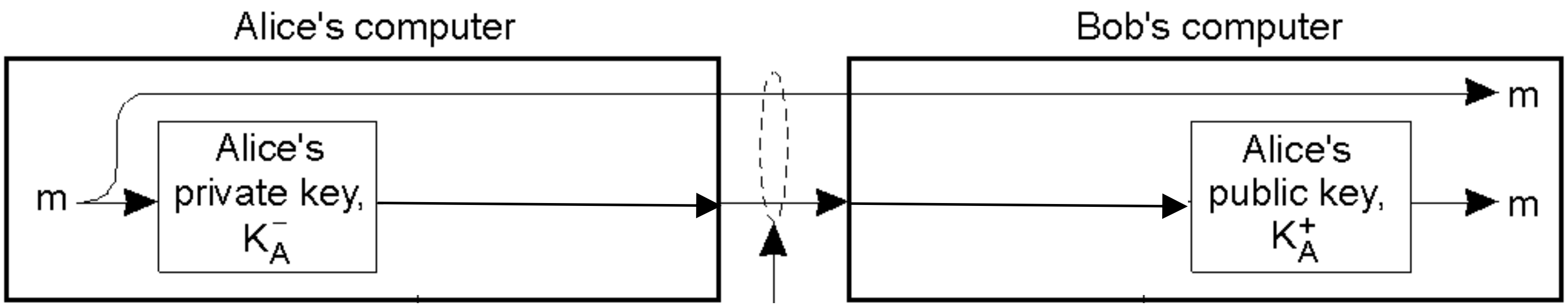
- Message m → hash function → Hash h
  - $h = H(m)$
- H function co-domain is smaller than the domain
  - H is not injective
  - We do not have a different hash for every message
- Collisions: $m \neq m'$ but $H(m)=H(m')$
  - The old MD5 used 128 bit (16 byte)
  - Messages can be of any length
  - For 17 bytes we have 256 collision, for 1Mbyte we have 250 Millions collision, …

Hash

- The desired properties of h depend on the intended use of hashing (error codes, hash tables, …)
  - One way
    - Given h such that h=H(m) …
    - … it should be hard to find m
  - Weak collision resistance
    - Given h and m such that h=H(m) …
    - … it should be hard to find m'≠m such that h=H(m')
  - Strong collision resistance
    - Given H() …
    - … it should be hard to find m'≠m such that H(m)=H(m')

Digital signatures

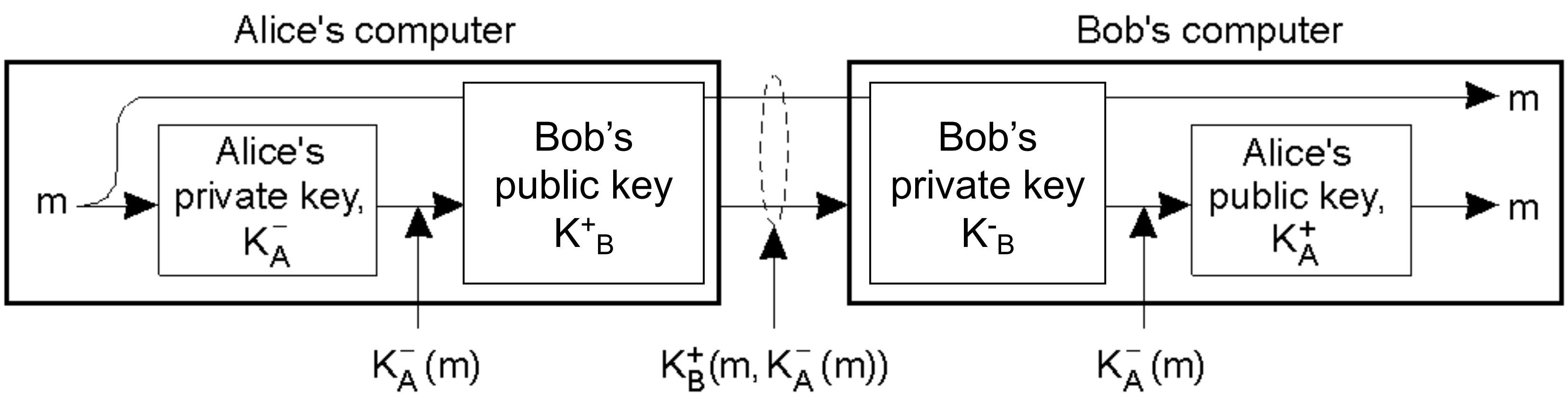
Signing



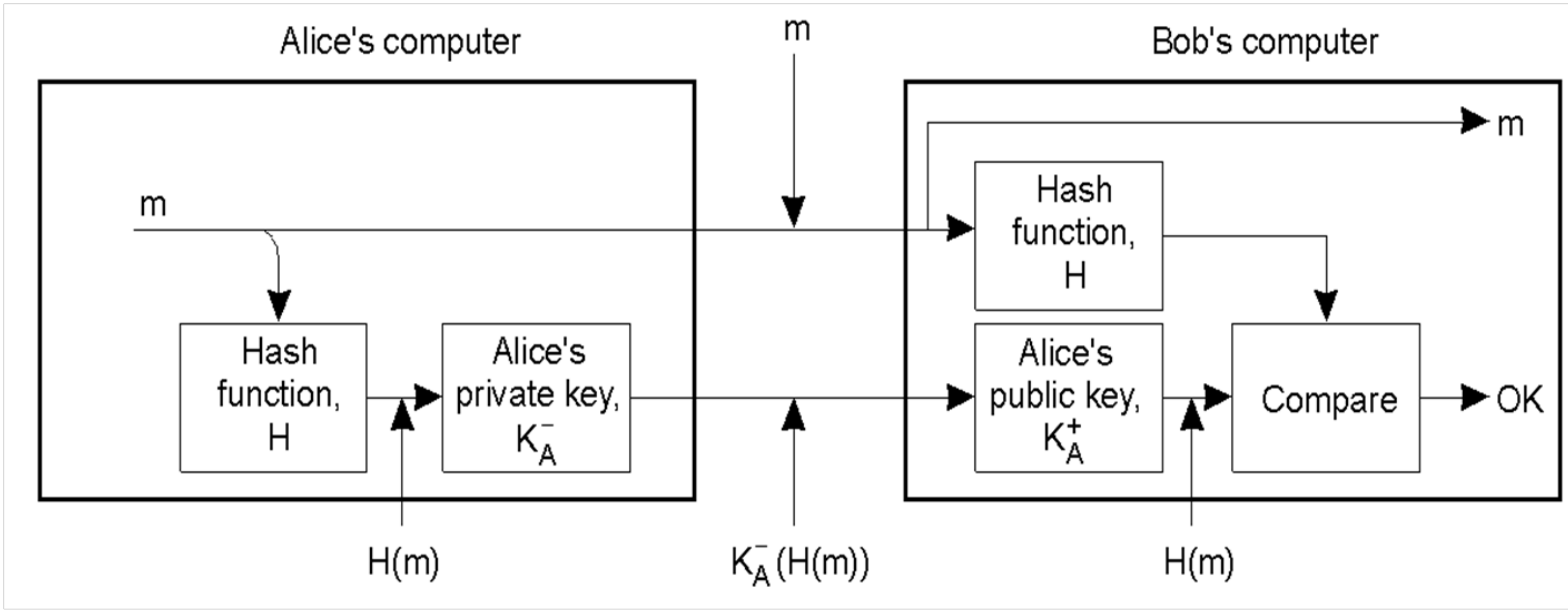
m,K^-_A(m)

Digital signatures

Signing and encrypting



Digital signatures



- Digitally signing a message using a message digest

Certification authorities

- How can we trust the association public key – physical person?

- Through public-key certificates
  - A tuple
    - Identity
    - Public key
    - Signed by a Certification Authority (CA)
  - The public key of the CA is assumed to be well-known
    - The basic idea: pervasive information is hard to alter
    - If the public key of the CA is everywhere, it is hard to alter every copy without being noticed
  - Yet the CA needs to authenticate public keys before issuing a certificate

Certification authorities

- We can have several trust models
  - Hierarchical
    - Root CA belongs to central authority (possibly governs)
    - There is a hierarchy of CAs that certificate each other
    - Leaf CAs certificate users
  - PGP's web of trust
    - Users can authenticate other users by signing their public key with their own
    - Users can define who they trust to authenticate others
    - The two are orthogonal: I can be sure that K^{+}_{A} is Alice's key, but I may not trust her diligence in signing others' key

Certificates lifetime

- A certificate associates a public key to the owner of the private key
- What if the private key is compromised?
- Solution: revocation
  - Certificate revocation list (CRL) published periodically by the CA
  - Every time a certificate is verified the current CRL should be consulted
  - This means that a client should contact CA every time a new CRL is published

SECURE CHANNELS

Secure channels

- Secure channels provide secure communication in distributed systems

- Secure channels provide protection against
  - Interception (through encryption)
  - Modification and fabrication (through authentication and message integrity)
  - Do not protect against interruption

- We assume that processes are secure, whereas every message can be intercepted, modified and forged by an attacker

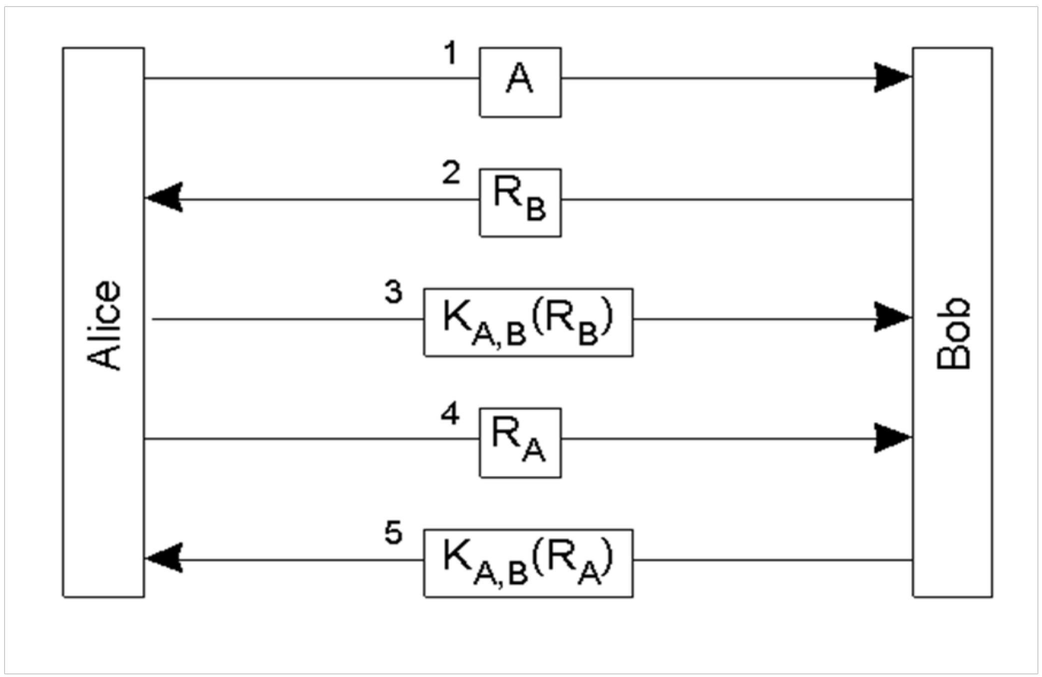
Secure channels

- Authentication and message integrity should go together (sender and content are together)
  - If a message is modified, knowing the sender of the original message is no longer useful
  - An unmodified message is not very useful if I do not know its source

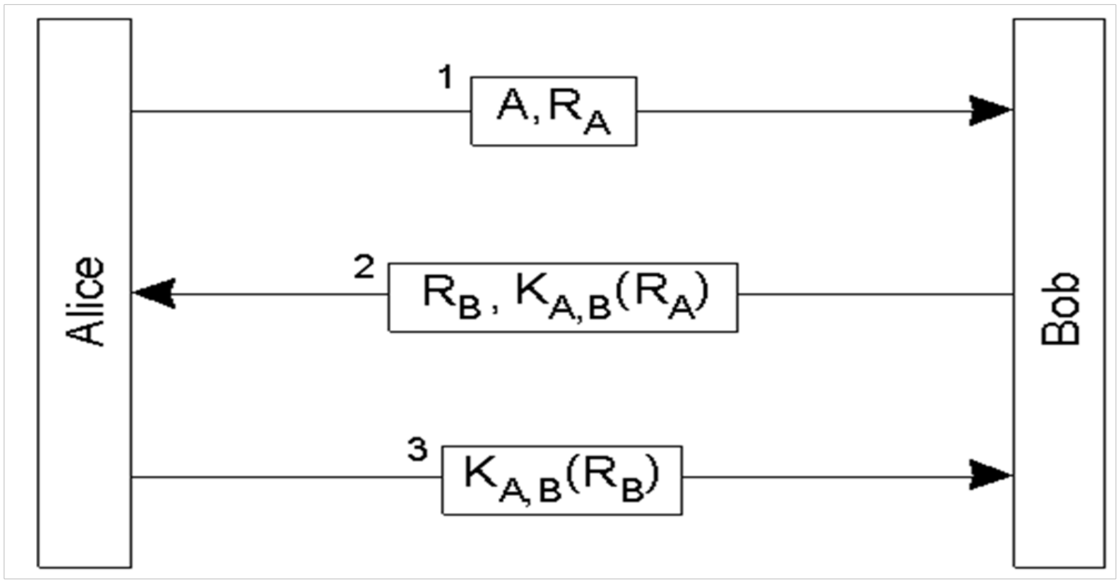
- Authentication needs shared information between the authenticator and the party
  - The very concept of authentication requires that (not the implementation in a specific protocol)
  - In the following protocols this information is the authorization key (either symmetric or asymmetric) exchanged beforehand
  - How this authorization key is exchanged? Hard: we'll try to answer later
  - Authentication protocols verify this common information without disclosing it on the channel

Secure channels

- Authentication with shared secret key
  - Challenge-response

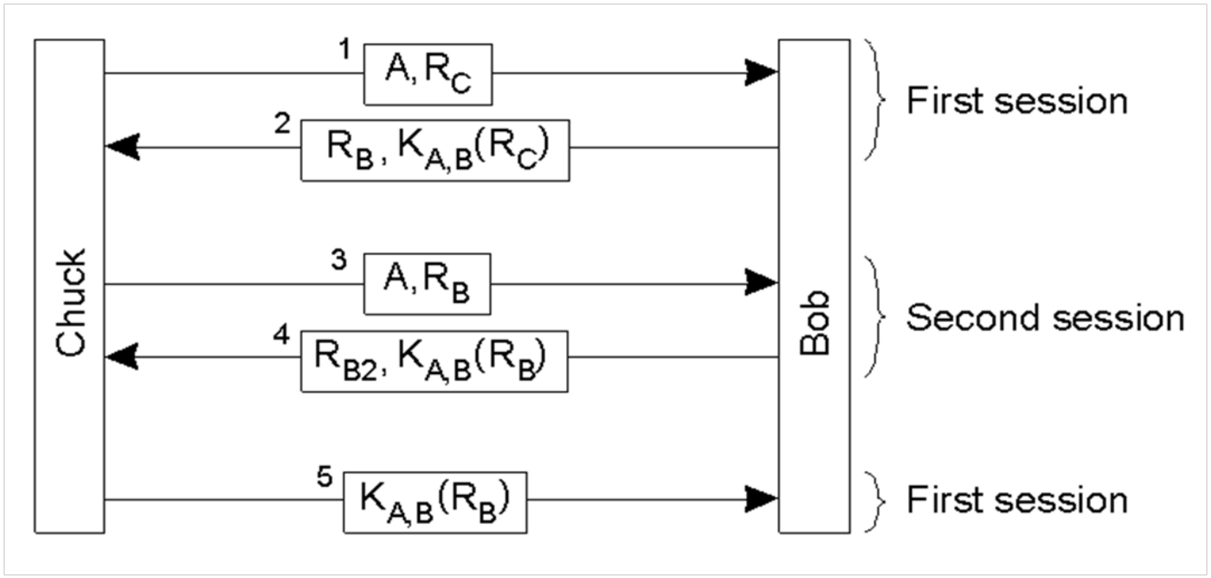


Secure channels



- It seems a very long exchange… let's try to piggyback some message
- Since A is going to authenticate B, let's challenge B in the first round
  - It does not work!

Secure channels

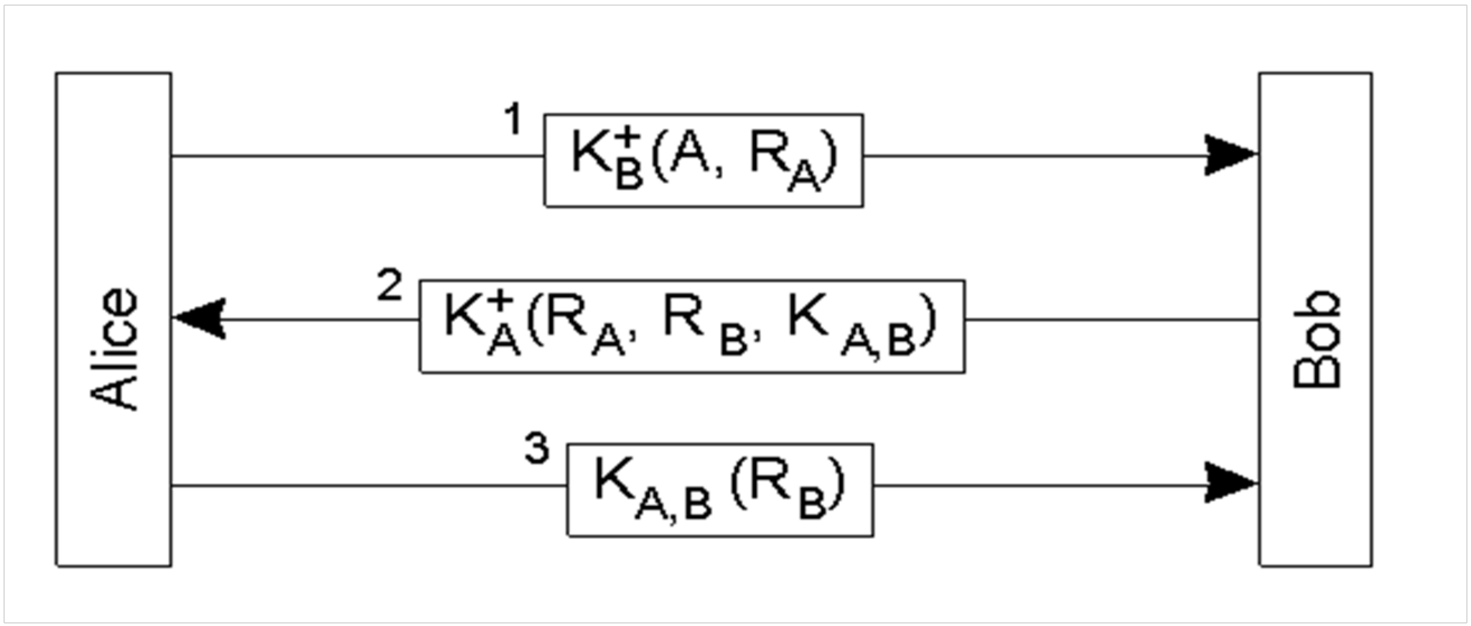


- Reflection attack
  - An attacker can request the response of the challenge!
    - Fix: use even for request, odd for response
- Key wearing
  - An active attacker can solicit use of the key (causing "key wearing")

Secure channels

- What happens in a secure channel set-up after authentication?
- Usually a session key (symmetric) is exchanged to provide integrity and possibly confidence of following messages
- Session keys are useful to limit the wearing of the main key (used for authentication)
- After the session is closed the session key must be destroyed

Secure channels



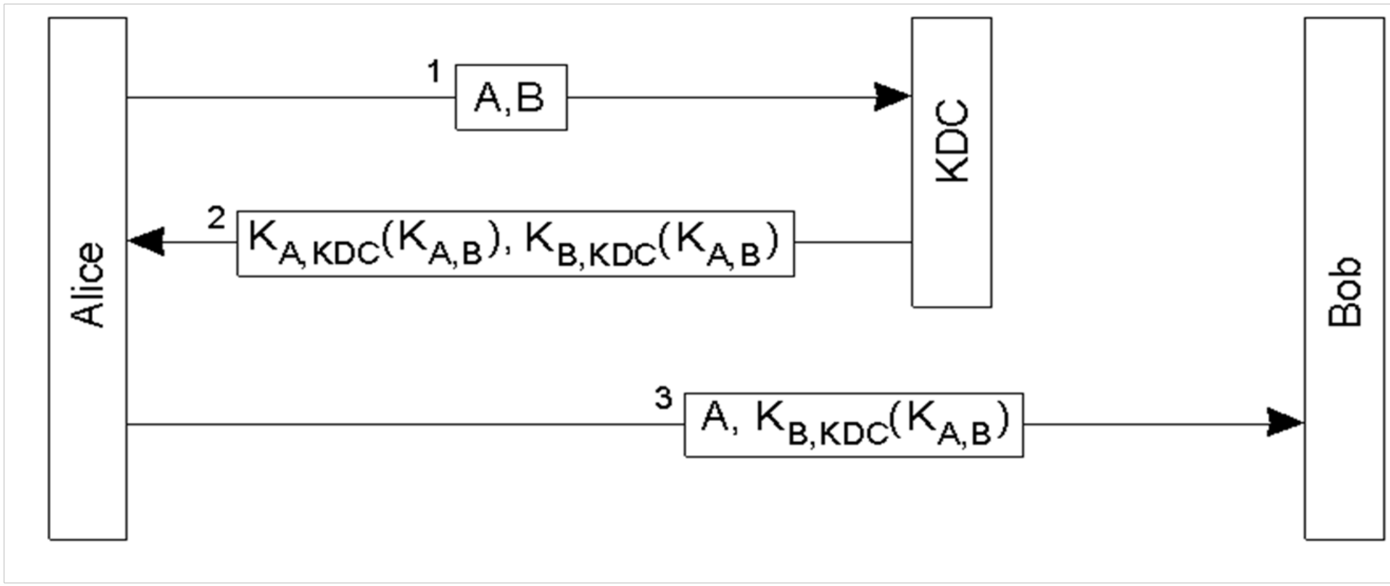
- Authentication using public-key cryptography

Secure channels

- One of the problems with using a shared secret key for authentication is scalability
  – If a distributed system contains N hosts, and each host is required to share a secret with each of the other N-1 hosts, we have N(N-1)/2 keys!
  – With public key cryptography, the keys are only 2N
    - But each node needs to know the public key of any other node

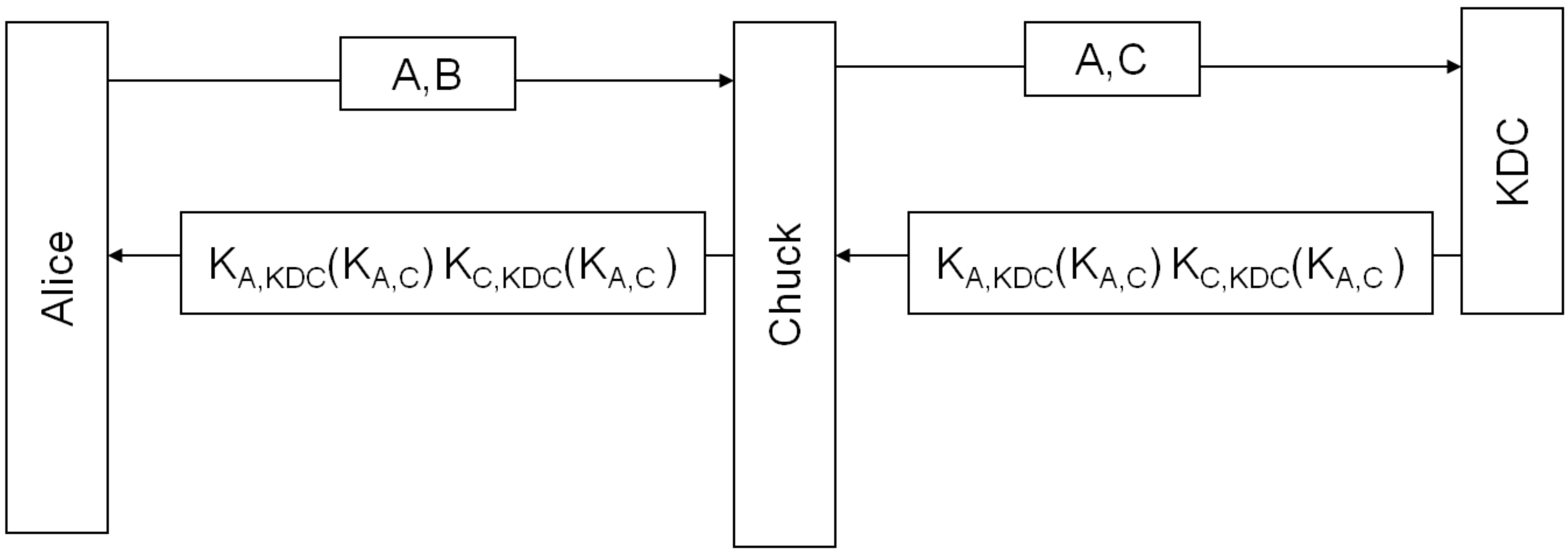
- An alternative is to use a centralized approach by means of a Key Distribution Center (KDC)
  – KDC shares a secret with each of the hosts (N keys) and generates tickets to allow communication between hosts

Secure channels



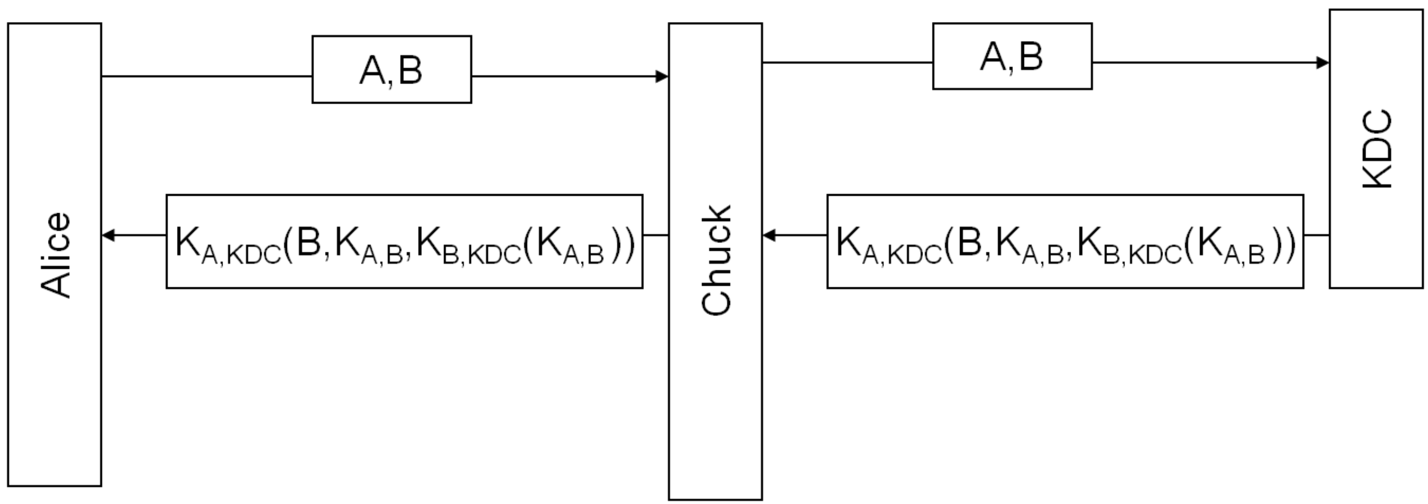
- Authentication with trusted KDC
  - $K_{B,KDC}$ ($K_{A,B}$) is called ticket
  - This protocol is not safe from many attacks

Secure channels



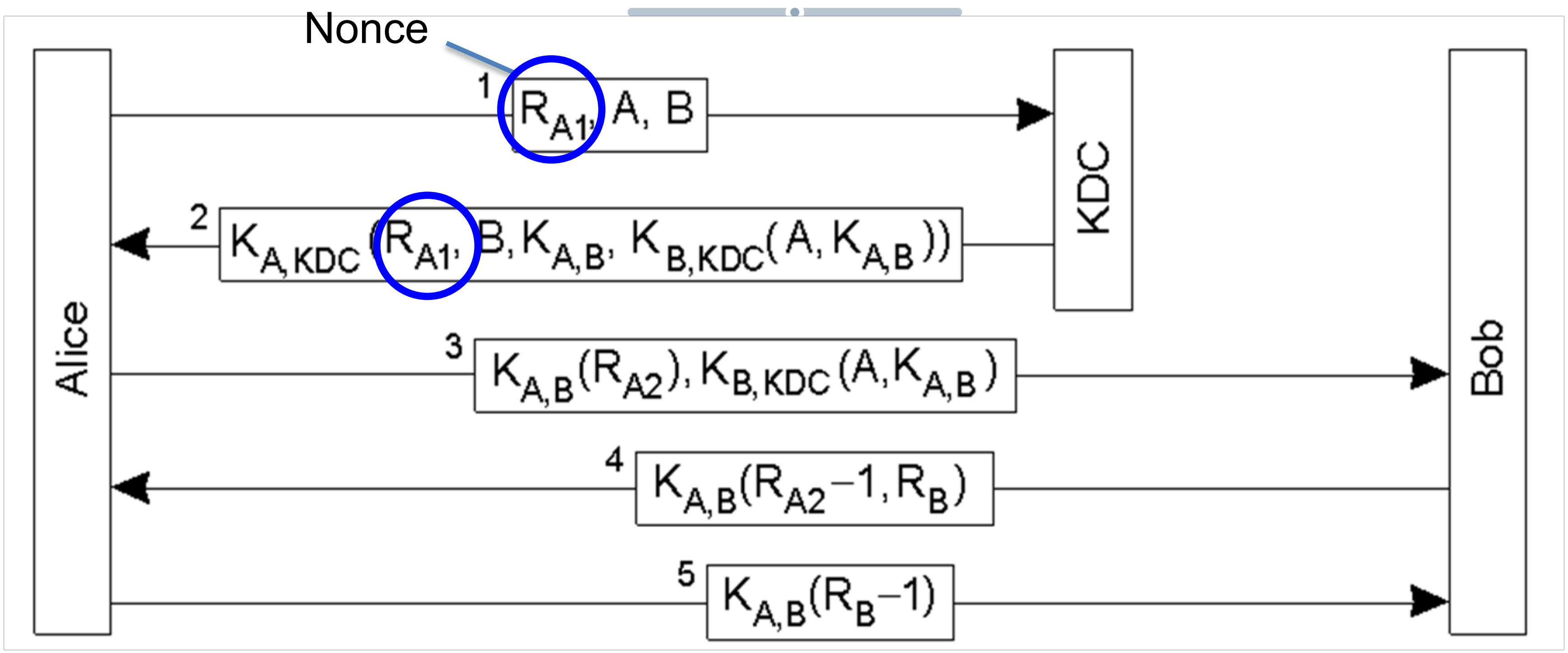
- Modification of the first message
  - Solution: we could encrypt A,B → $K_{A,KDC}$(A,B)
  - Unfortunately Chuck can replay an old $K_{A,KDC}$(A,C)

Secure channels



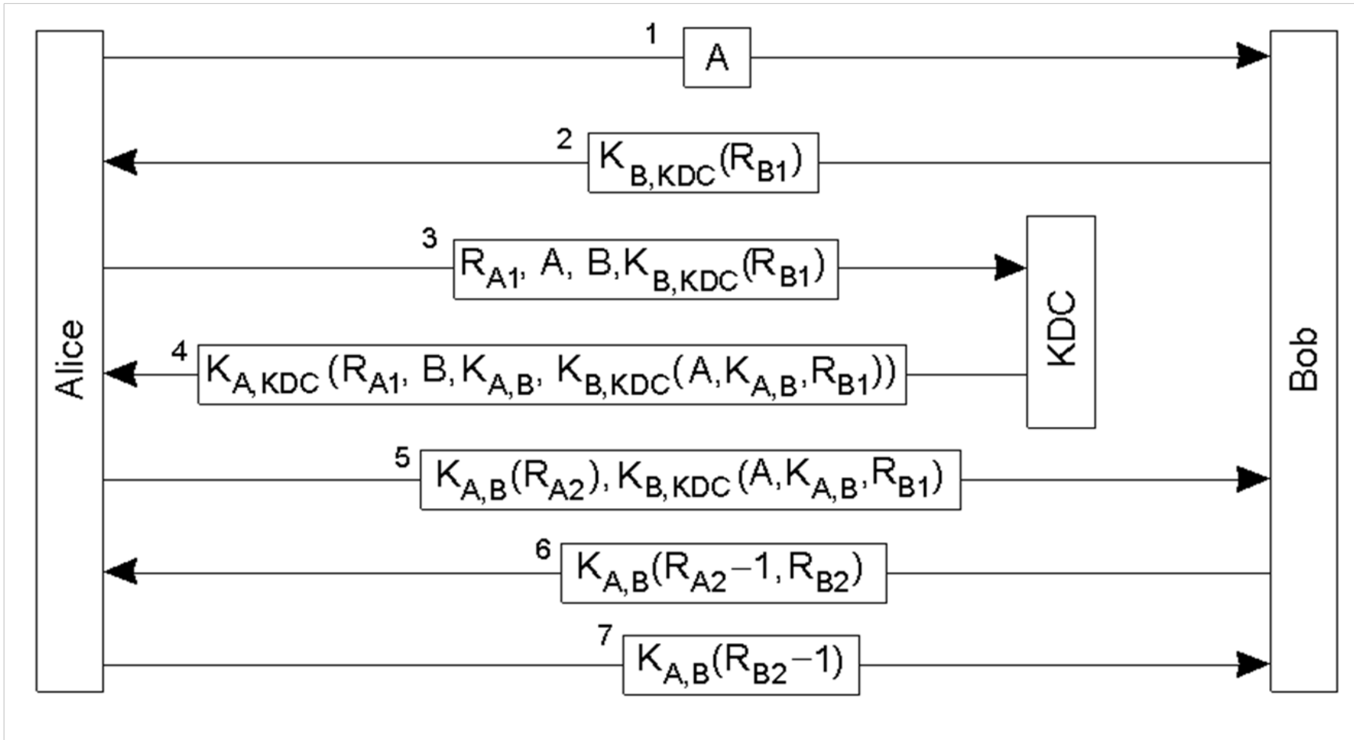
- Solution: put B in the response from KDC and protect it with $K_{A,KDC}$
  - Chuck cannot modify A's request
  - But after stealing $K_{B,KDC}$ he can serve an old KDC reply forever
    - Even after a fresh negotiation of $K_{B,KDC}$ between Bob and KDC

Secure channels



- Needham-Schroeder protocol solves these issues
- A last possible attack is replay msg 3 when $K_{A,B}$ is compromised

Secure channels



- Protection against malicious reuse of a previously generated session key in the Needham-Schroeder protocol

SECURITY MANAGEMENT

Security management

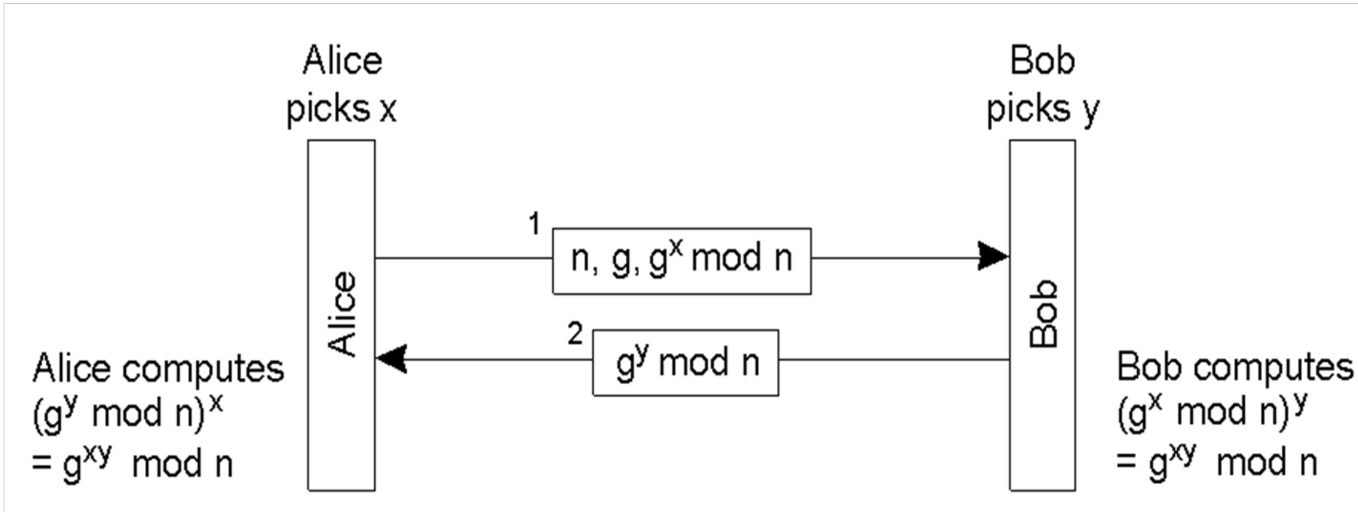
- We consider 3 different issues
  1. General management of cryptographic keys
     - How are public keys distributed?
  2. Secure management of a group of servers
     - How to add or remove servers from a group?
  3. Authorization management
     - How can a process securely delegate some of its rights to others?
     - Capabilities, certificates

Initial key distribution

- We said that for authentication protocols we need
  - Shared keys between each pair of nodes, or
  - Shared keys between each node and a KDC, or
  - Public key of each node

- Now we want to investigate how these initial keys can be distributed in a secure way
  - Diffie-Hellman algorithm to exchange symmetric keys on an insecure channel

Diffie-Hellman algorithm



- n and g publicly known numbers
  - With some good math property
- Is this a solution?

Diffie-Hellman algorithm

- Diffie-Hellman works only against passive attacks
  - If active opponent: man in the middle attack

- We need authentication (and integrity) of both DH messages
  - These two messages can be seen as a public key exchange

- What's the usefulness of Diffie-Hellman key exchange mechanism?
  - It's a way to transform a secret key exchange in a public key exchange
  - That's good since the first requires both confidentiality and integrity, while the second only requires integrity (authentication)

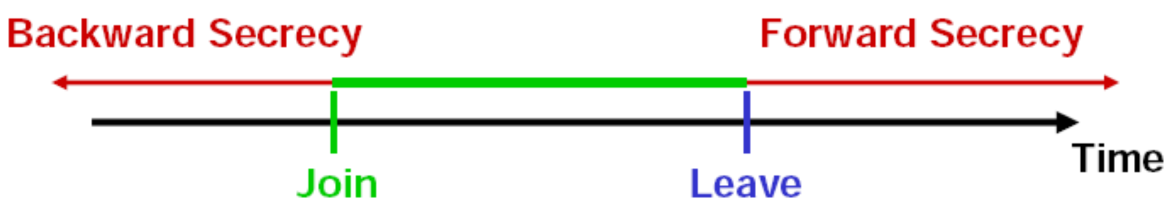
Secure group communication

- Suppose we want secure group communication
  - Symmetric encryption with a key for each pair of participants
    - Encryption O(n)
    - Requires O(n^2) keys
  - Public key encryption
    - Encryption O(n)
    - Requires O(n) keys
    - Computationally very expensive
  - Symmetric encryption with s single shared key
    - Encryption O(1)
    - Requires only one key

Cost	Symmetric, pair	Public	Symmetric, single
Processing	n	n	1
Keys in the system	n(n-1)/2	2n	1

Secure group communication

- Joining and leaving a group
  - Requirements
  - Backward secrecy: cannot decrypt messages before join
  - Forward secrecy: cannot decrypt messages after leave



- Solution: change the group key
  - Encryption of the new key
    - Join: with the old group key, with the joiner's key: O(1)
    - Leave: with every remaining member's key: O(n)
  - Leave is a costly operation!

These are personal keys not used for group communication

Secure group communication

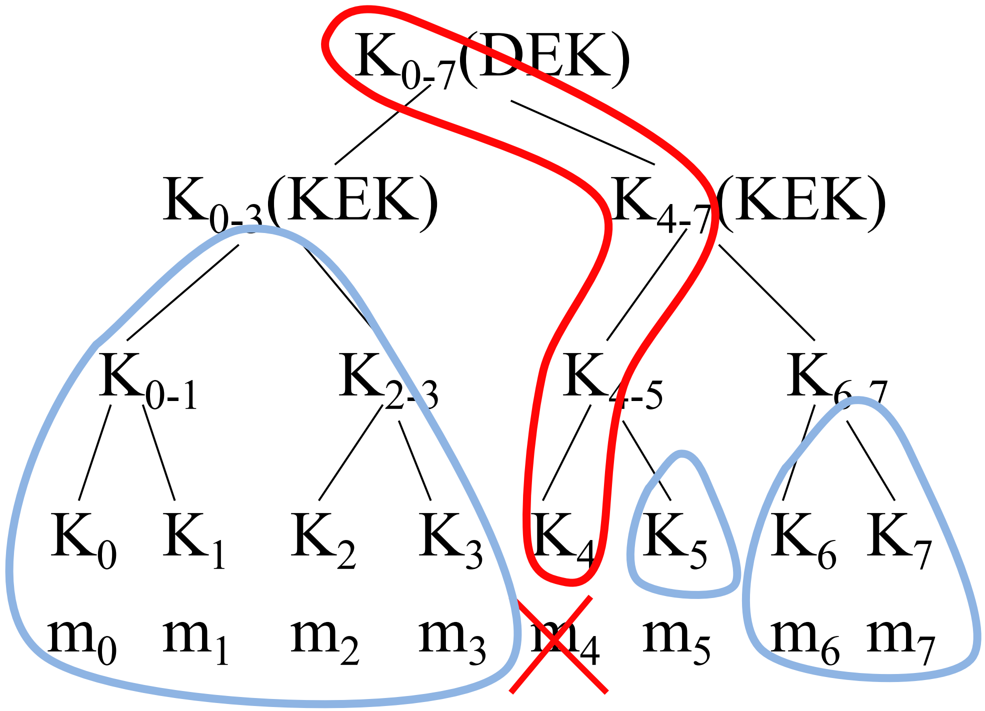
- Who will choose the new key?

- One server/leader
  - Key distribution problem
    - We will see two examples of efficient techniques for performing key distribution

- The participants
  - Key agreement problem
    - We have seen Diffie-Hellman for 2 parties
    - It can be generalized to more than two participants

Efficient key distribution

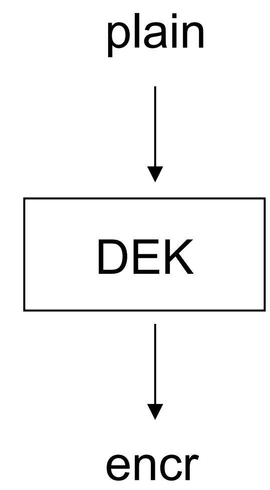
- Logical key hierarchy (tree)
  - Leaves: members with keys
  - Root: Data Encryption Key
  - Each member knows the Key Encryption Keys up to root
- Leave:
  - Change all the keys known by the leaving member
    - Data Encryption Key
    - Keys on the path to root
  - Diffuse efficiently the new keys
    - Encrypt each key with the children
    - Exploit stable subtrees for key distribution



$K_5(K_{4-5}') \rightarrow K_{4-5}'(K_{4-7}') \rightarrow K_{6-7}(K_{4-7}') \rightarrow K_{0-3}(K_{0-7}') \rightarrow K_{4-7}'(K_{0-7}')$

Efficient key distribution

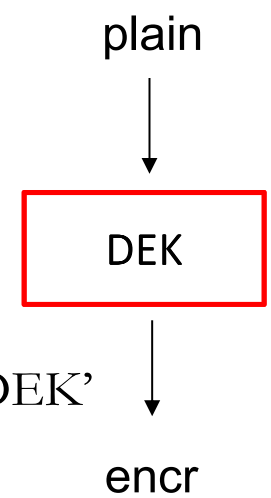
- Centralized flat table
  - 2 KEKs for each bit of member ID + 1 DEK
    - (= 2log(n)+1 vs 2n-1 in LKH)
  - Each node has a key for each bit in its ID
    - Node 9=(1001) has keys $K_{0,1}$ $K_{1,0}$ $K_{2,0}$ $K_{3,1}$ + DEK
    - If 9 leaves all these keys + DEK must be changed



	ID bit 0	ID bit 1	ID bit 2	ID bit 3
Bit = 0	$K_{0,0}$	$K_{1,0}$	$K_{2,0}$	$K_{3,0}$
Bit = 1	$K_{0,1}$	$K_{1,1}$	$K_{2,1}$	$K_{3,1}$

Efficient key distribution

- Centralized flat table
  - Encrypt DEK’ with every other (still valid) key
    - Each member except 9 can decrypt the new DEK’
  - Encrypt new KEKs with the old one and with DEK’
    - Send $K_{DEK'}(K_{i,j}(K_{i,j'}))$
    - Each member receives (and can read) only KEKs for its bits
    - Member 9 cannot access the new keys because it does not have DEK’
  - Collusion attack
    - Difficult to remove many participants



	ID bit 0	ID bit 1	ID bit 2	ID bit 3
Bit = 0	$K_{0,0}$	$K_{1,0}$	$K_{2,0}$	$K_{3,0}$
Bit = 1	$K_{0,1}$	$K_{1,1}$	$K_{2,1}$	$K_{3,1}$

Secure replicated servers

- A client issues a request to a (transparently) replicated server
  - We want to be able to filter c responses corrupted by an intruder

- First simple solution
  - Use 2c+1 replicated servers
  - Each server signs its own response
  - The client verifies the signature and decides on a majority
  - However this forces the client to know the identity (and the public key) of all the servers

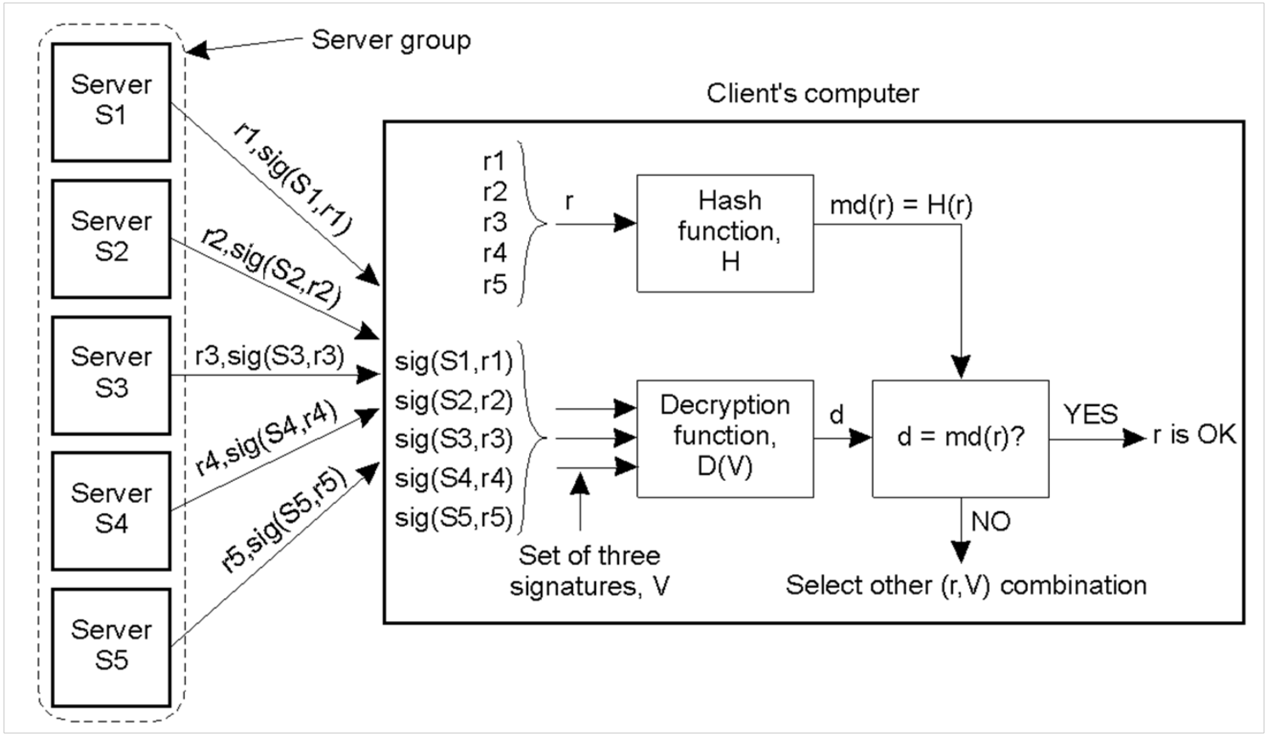
Secure replicated servers

- (n,m) Threshold schemes
  - Divide a secret into m pieces
  - n pieces are sufficient to reconstruct the secret (e.g. n degree polynomial and m evaluation)

- Applied to signatures
  - Find a way such that c+1 correct server signatures are needed to build a valid signature of the response
  - r_i response from server i
  - $sig(S_i,r_i)$ signature from server i of r_i

Secure replicated servers

- Sharing a secret signature in a group of replicated servers
  - Transparent replication is possible by collecting signatures inside the group of servers and forwarding a single reply to the client



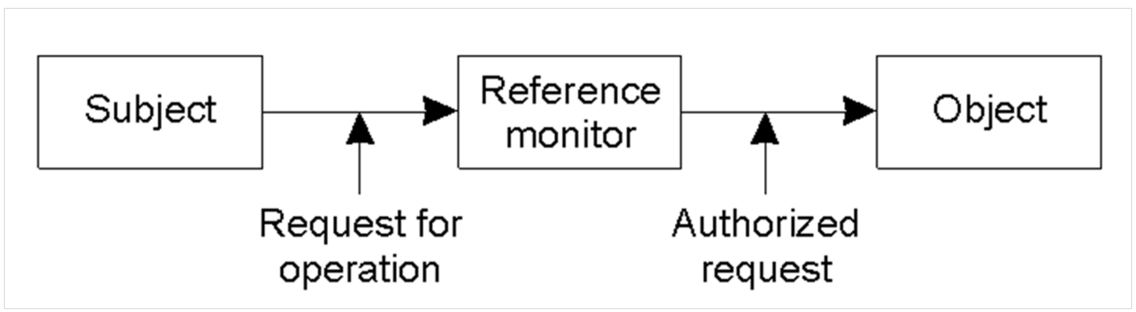
Access control

- In non distributed systems, managing access rights is relatively easy
  - Each user has rights to use resources

- In distributed systems account management is not trivial
  - We need to create an account for each user on each machine …
  - … or have a central server that manages authorization rights
  - A better approach is the use of capabilities

Access control

- Access control is done through a reference monitor which mediates requests from subjects to access objects



Access control

- Access control matrix
  - Conceptually, we have a matrix listing rights for every combination (resource, user)

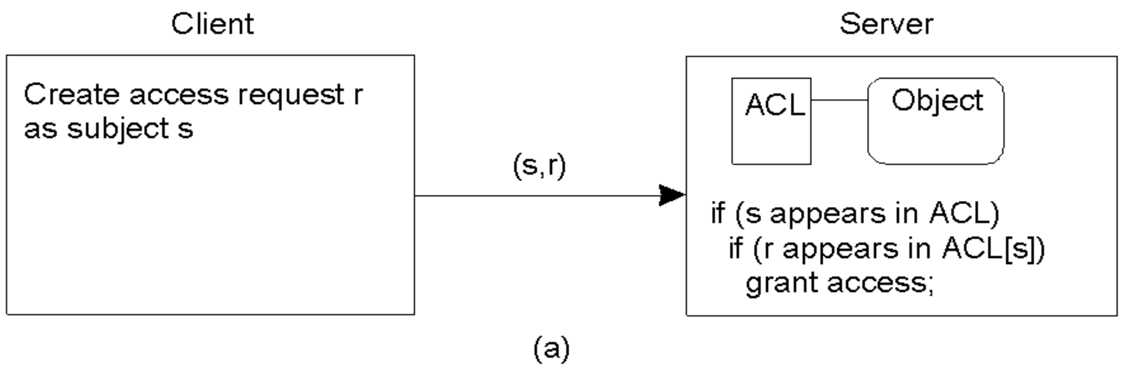
	Object 1	…	Object n
User A	R,W	…	R
…	…	…	…
User k	W	…	W

  - It's a sparse matrix (it is not efficient to implement it as a true matrix)
    - If we distribute it column-wise we have Access Control Lists (ACL): each object has its own ACL
    - If we distribute it row-wise we have capabilities lists: each capability is an entry in the access control matrix

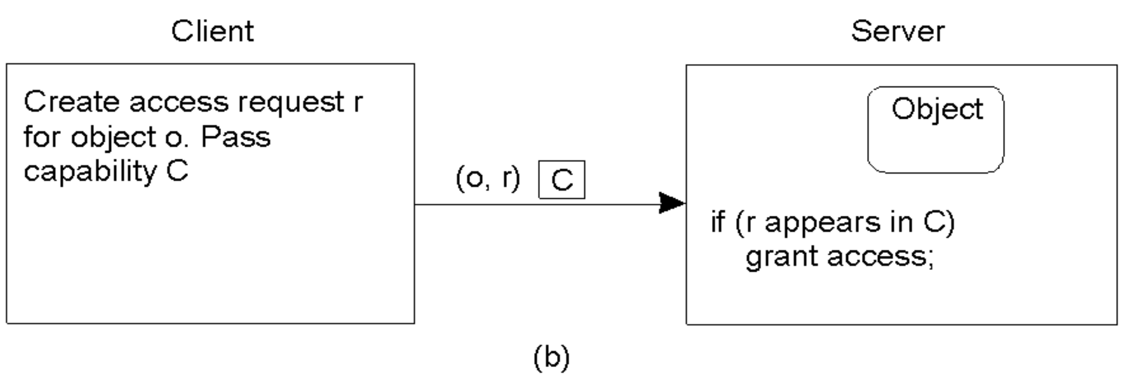
Access control

- Using ACL vs. capabilities lists affects the way interaction takes place with the Reference Monitor

a) ACL

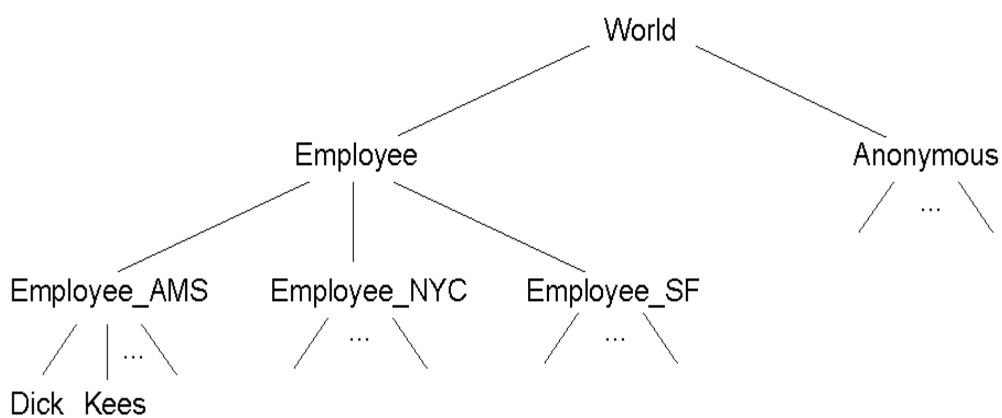


b) Capabilities



Access control

- ACL still occupy great memory. We can use groups to build a hierarchy in ACL
  - Management is easy and large groups can be constructed efficiently
  - Costly lookup procedure
  - We can simplify the lookup by having each subject to carry a certificate listing groups he belongs to (this is similar to capabilities)



Access control

- Another possibility is role-based access control
- Each user is associated to one or more *roles*
  - Often mapped to user's functions in an organization
- When a user logs in, she specifies one of her roles
  - Roles define what can be done on which resources
  - Analogous to groups
- The difference is that users can dynamically switch from one role to another one
  - This is difficult to implement in terms of groups access control

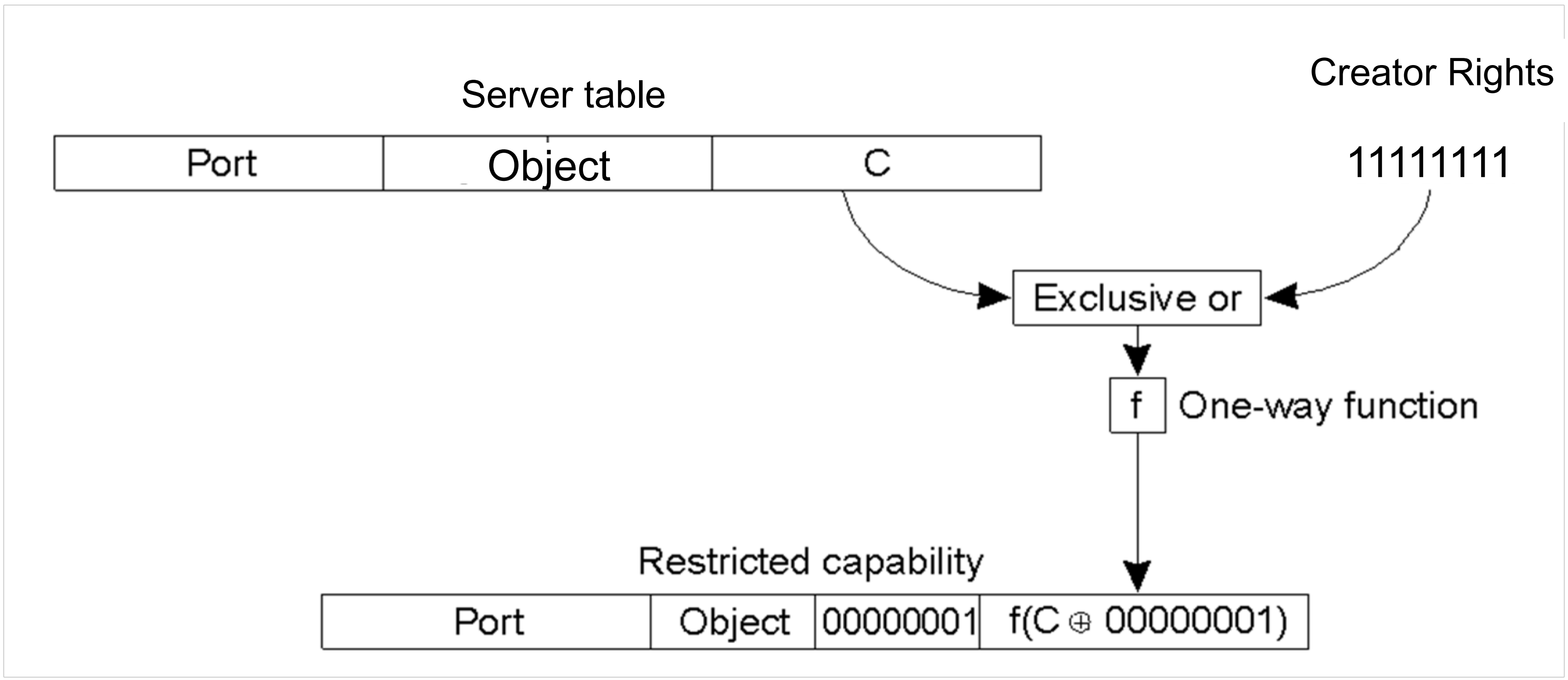
Capabilities

- An example of capabilities (Amoeba)
  - 128 bit
  - 72 (48+24) to identify an object
  - 8 for access rights
  - 48 to make it unforgeable

48 bits	24 bits	8 bits	48 bits
Server port	Object	Rights	Check

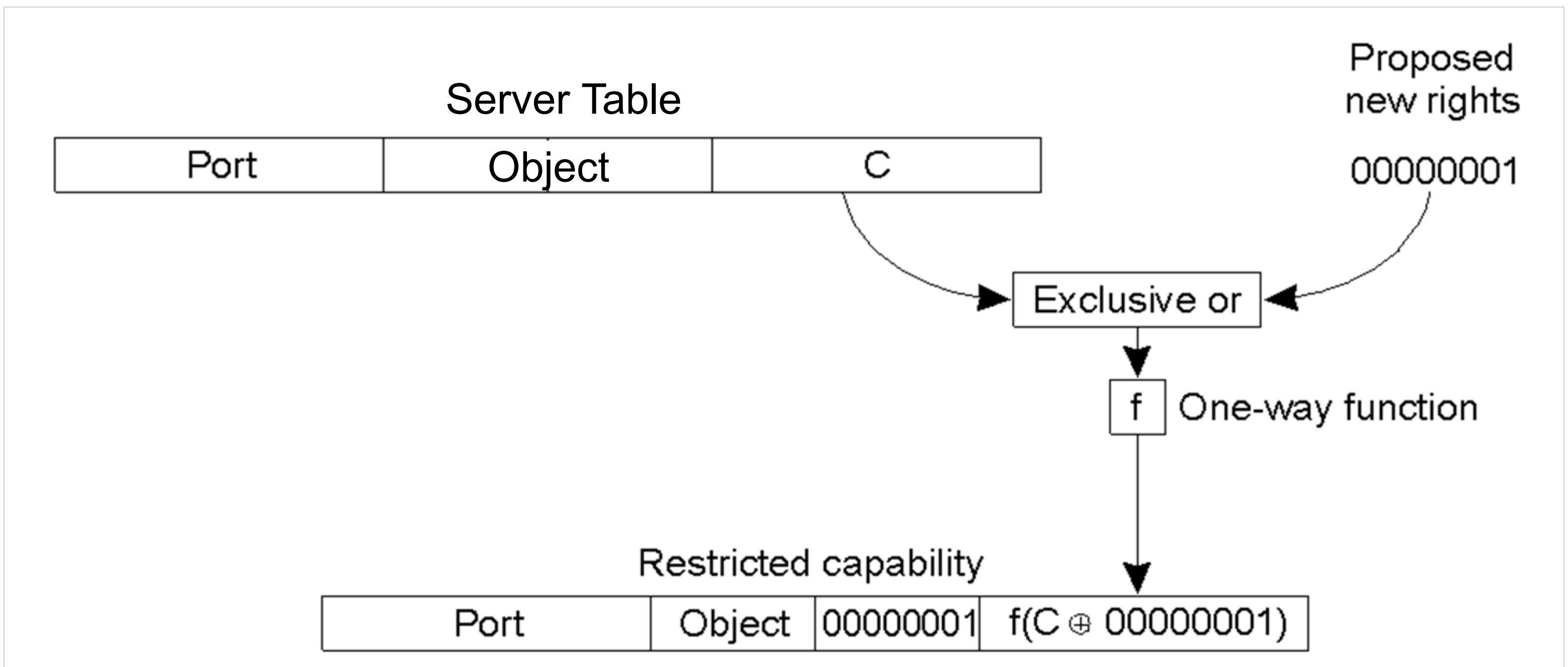
Capabilities

- On object creation, the server generates and stores C internally



Capabilities

- Changing rights is not possible for the owner of the capability
  - Only the server knows C

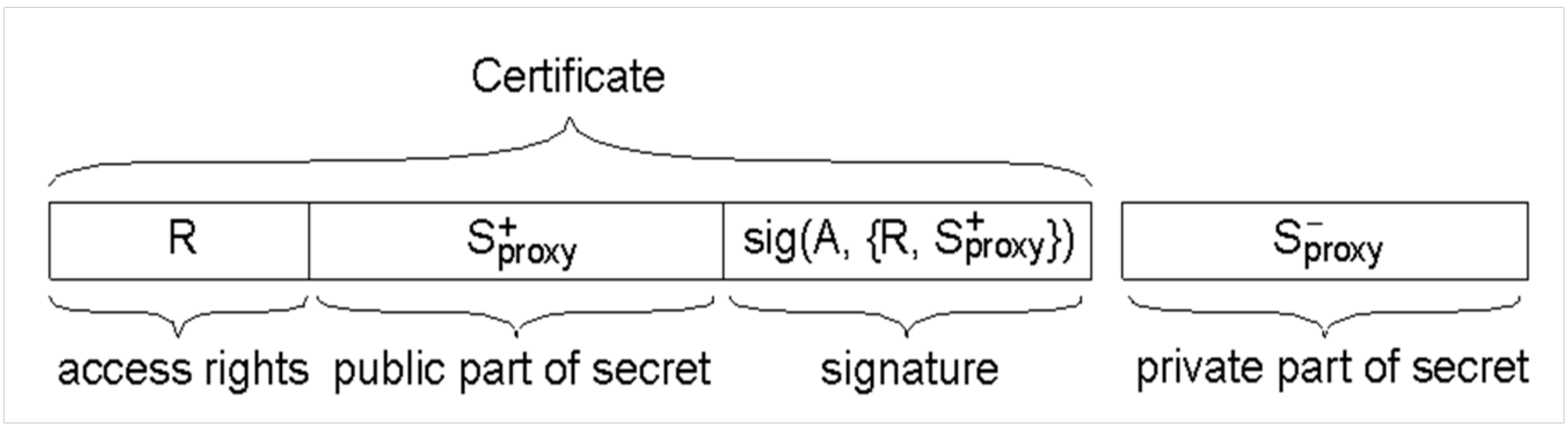


Capabilities delegation

- Delegation
  - A process may want to delegate some of its rights to other processes
    - Amoeba capabilities can only be passed as they are, it is not possible to further restrict rights unless we request a restricted capability to the server
    - A general scheme that supports delegation including rights restriction is based on proxies
  - A proxy is a token
    - Provides rights to the owner
    - A process can create proxies with at best the same rights of the proxy it owns

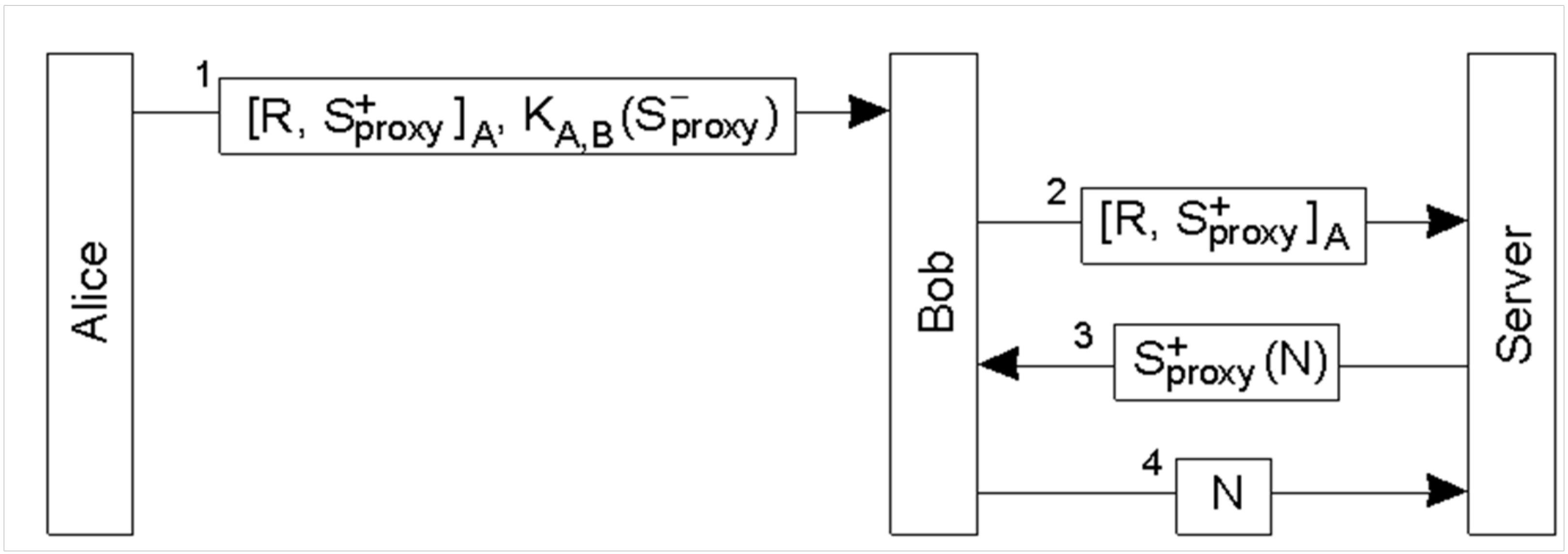
Proxy

- A proxy has two parts: a certificate and a key
  - The certificate proves that grantor A entitled R rights to some grantee
  - The key is the proof that a process is the grantee



Proxy

- Protocol for delegation and authorization



Restricting a proxy

- How can we restrict a proxy?
  - A → B → C → D (the server)
  - B receives $[R, S^{+}]_A$ S^{-}

- B places R2 restriction and sends the new proxy to C

  $[R, S^{+}]_A$ S^{-}

  $[R2, S2^{+}]$ $S2^{-}$

- C can't pretend that he is entitled with R because it doesn't know S^{-}

Restricting a proxy