Distributed Systems Security

Alessandro Sivieri

Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy

sivieri@elet.polimi.it

http://corsi.dei.polimi.it/distsys

Slides based on previous works by Matteo Migliavacca and Alessandro Margara

Outline

- Introduction
- Cryptography
- Secure Channels
- Secure Group Communication
- Security Management
- Access Control
- Electronic Payment

Introduction

Introduction

- Why security in a distributed systems course?
- Sharing of resources is the motivating factor for distributed systems
- Processes encapsulate resources and provide access to them through interaction with other 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* wrt 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

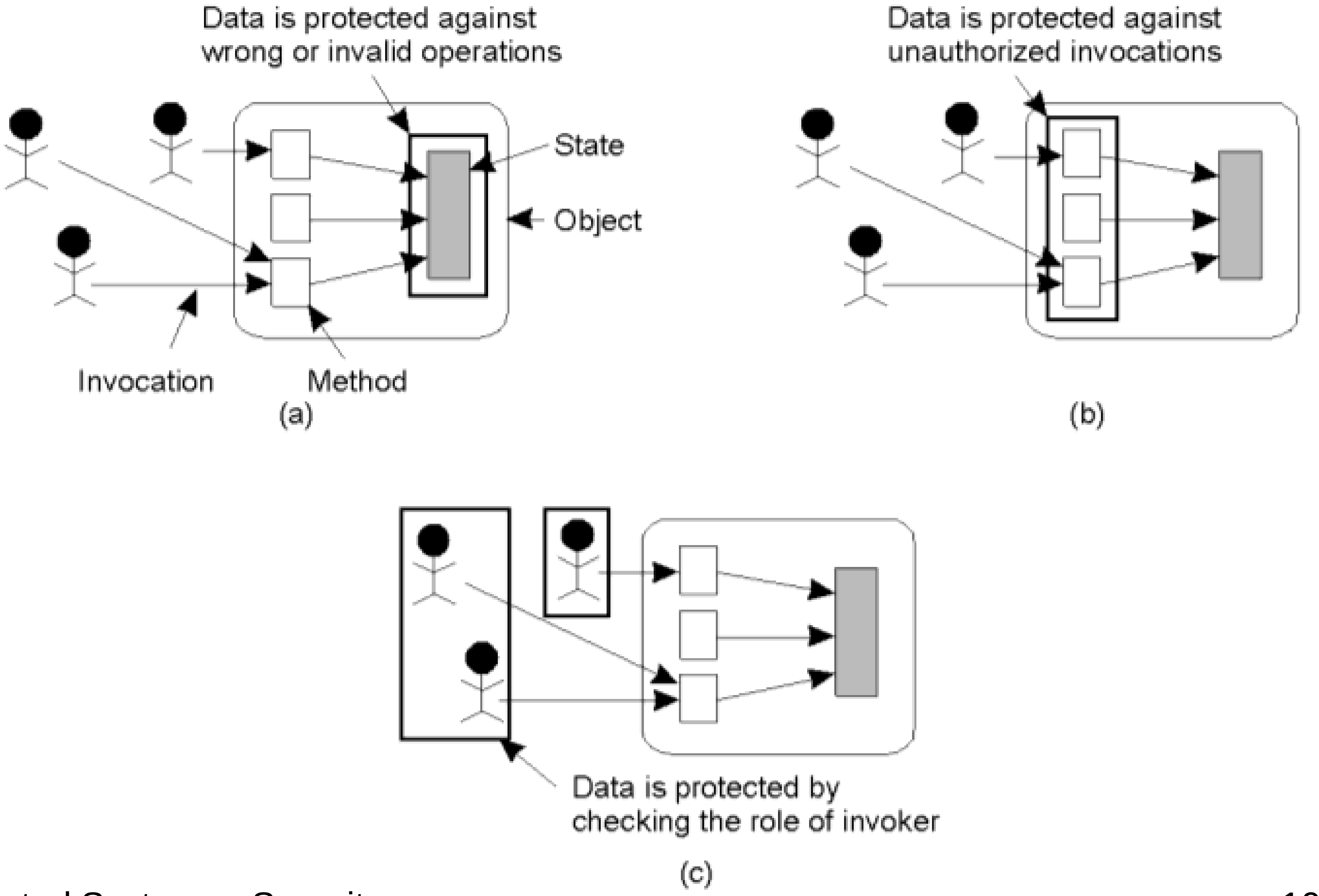
- Security Mechanisms
  - Encryption (confidentiality, verify integrity)
  - Authentication (identification of parties)
  - Authorization (control information access)
  - Auditing (breach analysis, but also IDS)

Design Issues

- When considering the protection of a (distributed) application, there are essentially three approaches that can be followed:
  - Concentrate directly on protection of the data, irrespective of the various operations that can possibly be performed on data items
  - Concentrate on protection by specifying exactly which operations may be invoked
  - Focus directly on users: only specific people have access to the application (role of the invoker)

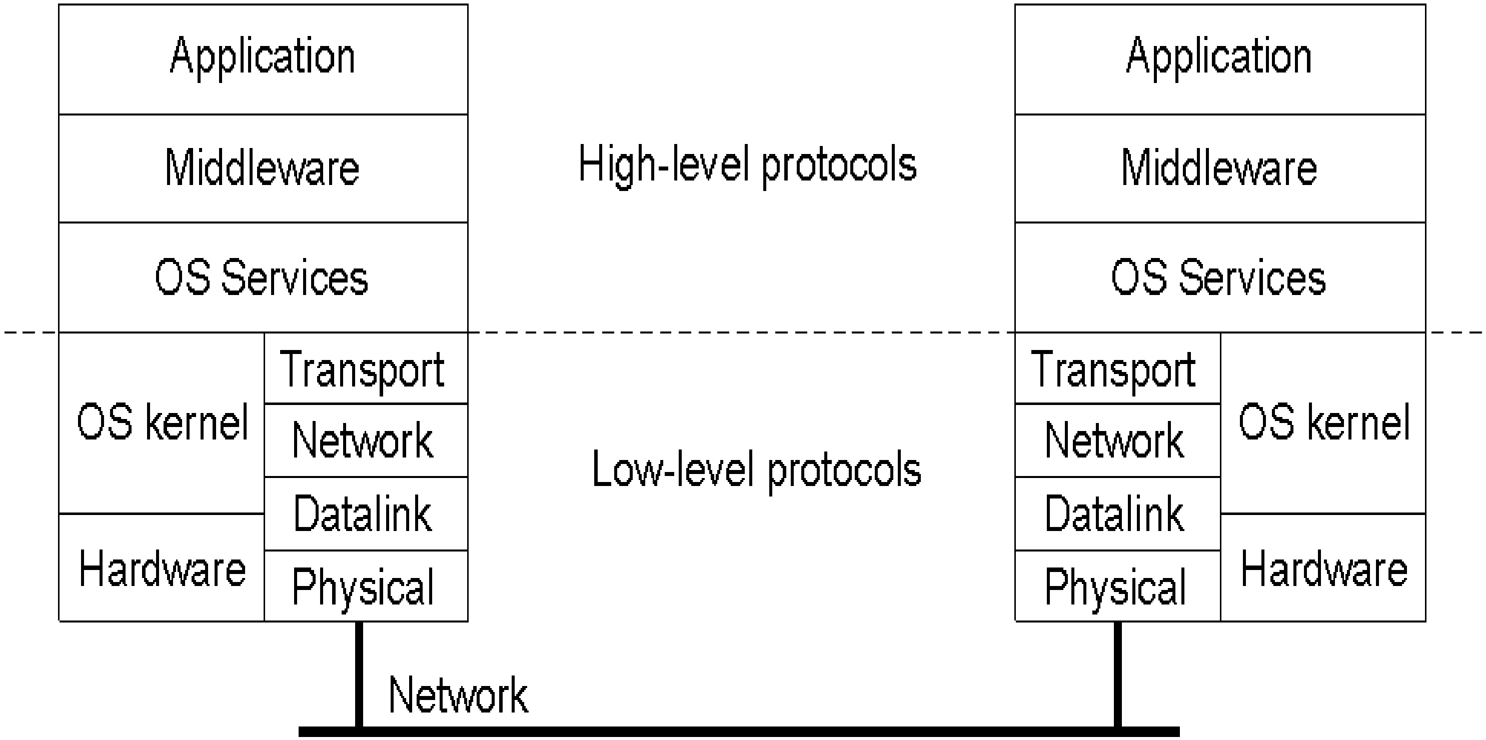
Design Issues

- Focus of control


Data is protected against wrong or invalid operations
State
Object
Invocation
Method
(a)
Data is protected against unauthorized invocations
(b)
Data is protected by checking the role of invoker
(c)

Design Issues

- Layering of security mechanisms



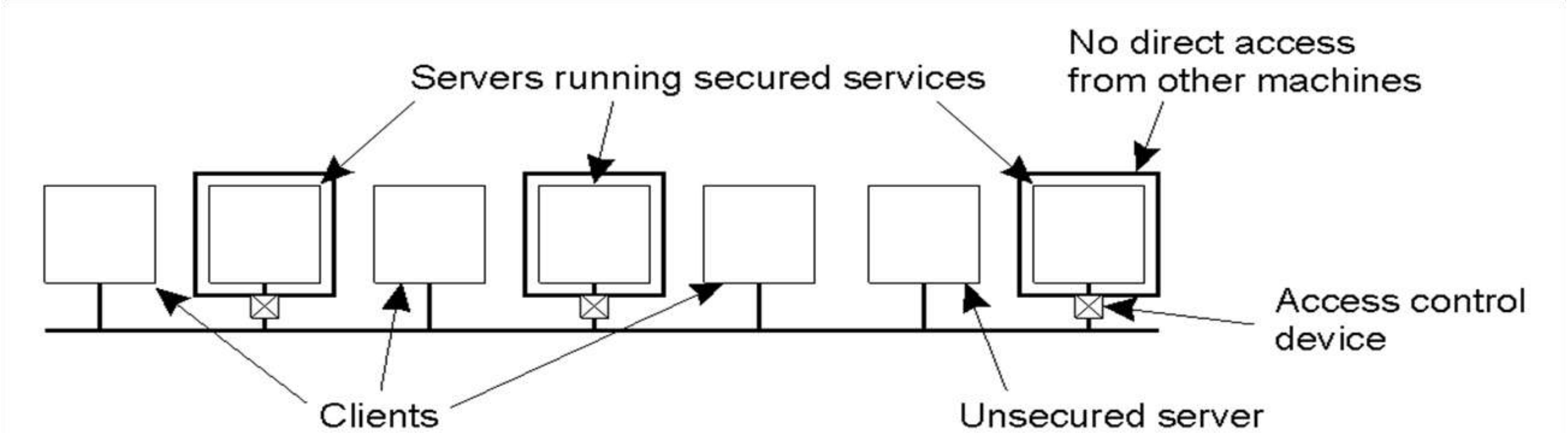
Layering of security mechanisms

- Where to put security mechanisms?
  - If you don’t trust security at low level you can build security mechanisms at higher level
    - If you don't trust transport and lower levels you can use SSL
- What to trust?
  - High level mechanisms depend on lower level mechanisms
    - Netscape’s SSL impl was built on top of 40 bit RC4, and hence was easily breakable
    - 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)

Politecnico
di Milano

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


Servers running secured services
No direct access from other machines
Access control device
Clients
Unsecured server

Design Issues

- Simplicity
  - As usual simple mechanisms lead to fewer errors (at design and implementation level)
  - But unfortunately (as we will see in the following) simple mechanisms are often not sufficient 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 it worse

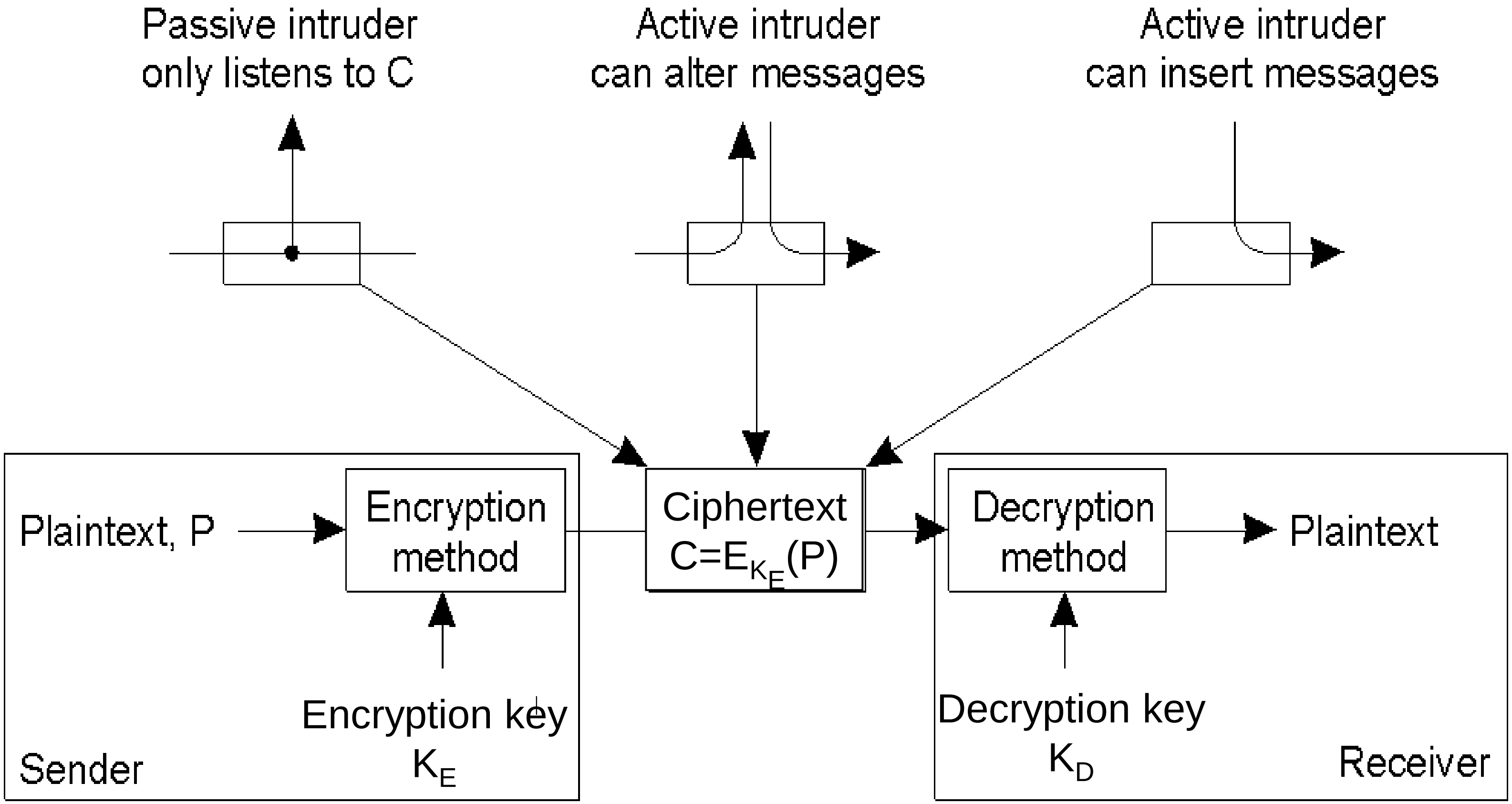
Cryptography

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)*, but this proved to be dangerous
  - Function can't 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
Encryption key
K_E
Sender
Ciphertext
$C=E_{K_E}(P)$
Decryption method
Plaintext
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

- $K_E \neq K_D$
- 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
- So we can safely distribute K_E to everyone who is interested in sending messages to me (while keeping K_D private)

Asymmetric Encryption (cont'd)

- We can also reverse things: distribute K_D and keep private K_E
  - This is how e-signing is done
- So there's no point in calling the two keys of a pair K_E and K_D.
- We call them K^+_A (A's public key) and K^-_A (A's private key)

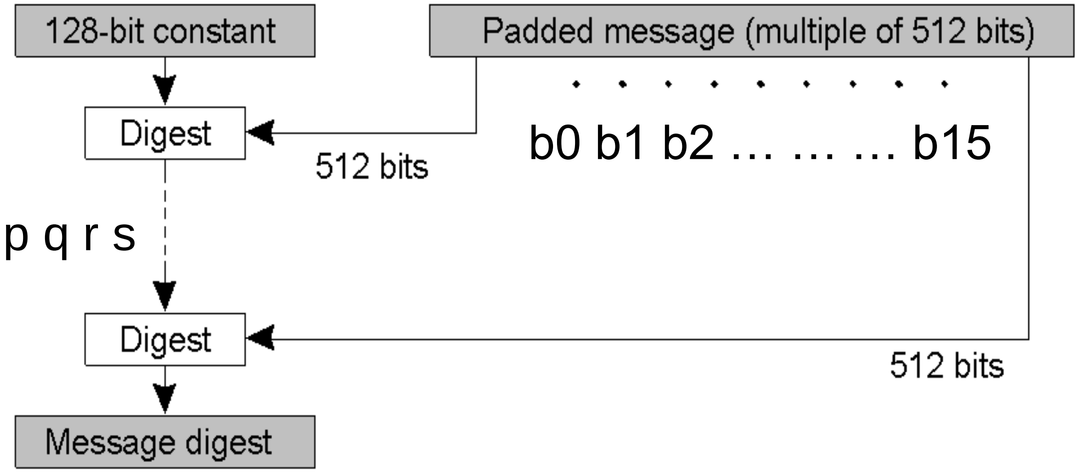
Hash

- Message m – hash function→ Hash h
  - h = H(m)
- A note about hashing (from a math point of view)
  - Since (usually) H function co-domain (range) is smaller than the domain, H cannot be injective
  - Collisions: m≠m' but H(m)=H(m')
    - MD5 uses 128 bit (16 byte). Messages can be of any length => if we try every 16 byte long message we exhausted the available space so for 17 bytes we have 256 collision, for 1Mbyte files we have 250 Millions collision
  - It's NOT true that for every message there's a different hash!!
  - So why we use hash for data integrity and digital signature?

Hash (cont'd)

- Message m – hash function→ Hash h
  - h = H(m)
- Properties
  - Depends on hashing intended use (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')

MD5 (Message Digest / Hash)



128-bit constant

Padded message (multiple of 512 bits)

Digest

512 bits

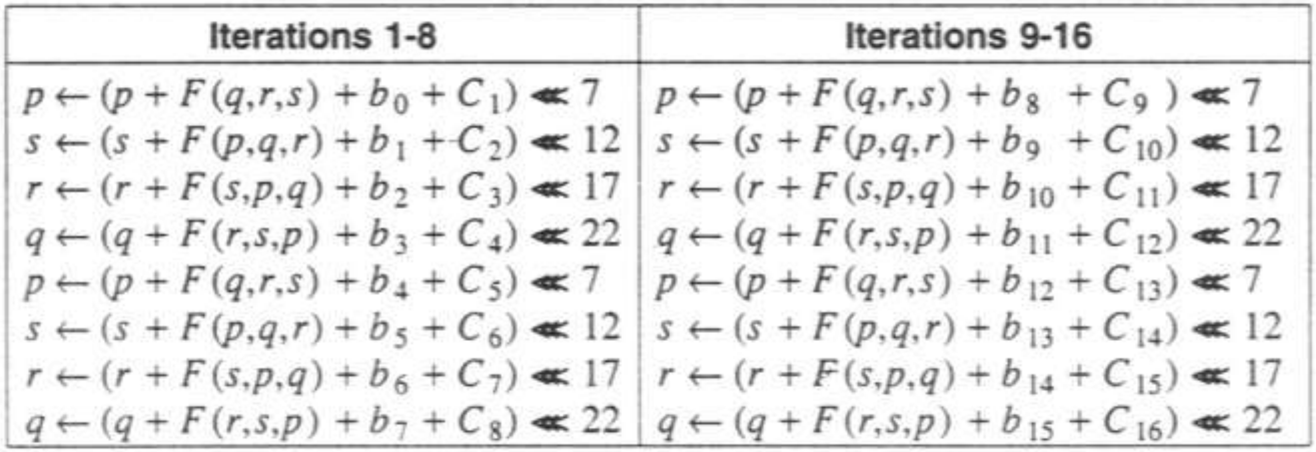
b0 b1 b2 … … … b15

p q r s

Digest

512 bits

Message digest



Iterations 1-8	Iterations 9-16
$p \leftarrow (p + F(q,r,s) + b_0 + C_1) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_8 + C_9) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_1 + C_2) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_9 + C_{10}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_2 + C_3) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{10} + C_{11}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_3 + C_4) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{11} + C_{12}) \lll 22$
$p \leftarrow (p + F(q,r,s) + b_4 + C_5) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_{12} + C_{13}) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_5 + C_6) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_{13} + C_{14}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_6 + C_7) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{14} + C_{15}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_7 + C_8) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{15} + C_{16}) \lll 22$

Total of 64 iterations for each message part
Uses 4 functions (F, G, H, I)

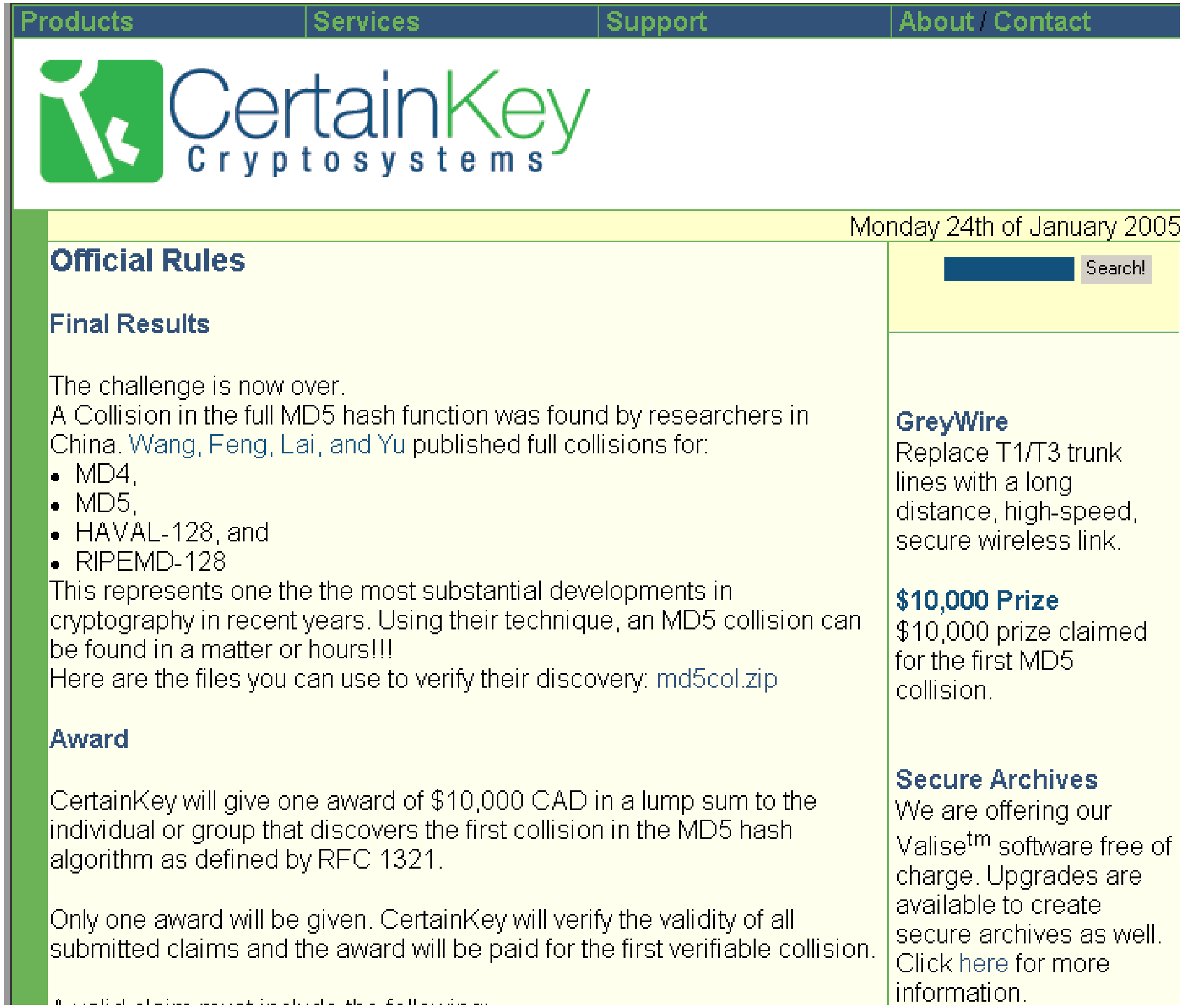
MD5 strong collision resistance

Jun 15, 2004 - Obsolete and Popular Banking Algorithm Gains New Foe

Ottawa - CertainKey, Inc., as a supporter and implementer of strong encryption, is proud to announce that it will award $10,000 to the first person or group to find a collision* in MD5, a 128-bit message digest (security algorithm) used by companies such as Bank of America, Citibank, Fleet Bank, eTrade.com USA, and eBay.com.

According to cryptography industry experts, the algorithm became obsolete in 1998, yet remains in use and leaves secure systems vulnerable to attack. With this award, CertainKey hopes to accelerate the adoption of more cryptographically secure algorithms to replace MD5 in all industry applications that relate to cryptography, and encourage public awareness of data security in general.

MD5 strong collision resistance



Monday 24th of January 2005

Official Rules

Final Results

The challenge is now over.
A Collision in the full MD5 hash function was found by researchers in China. Wang, Feng, Lai, and Yu published full collisions for:

- MD4,
- MD5,
- HAVAL-128, and
- RIPEMD-128

This represents one the the most substantial developments in cryptography in recent years. Using their technique, an MD5 collision can be found in a matter or hours!!!
Here are the files you can use to verify their discovery: md5col.zip

Award

CertainKey will give one award of $10,000 CAD in a lump sum to the individual or group that discovers the first collision in the MD5 hash algorithm as defined by RFC 1321.

Only one award will be given. CertainKey will verify the validity of all submitted claims and the award will be paid for the first verifiable collision.

A valid claim must include the following:

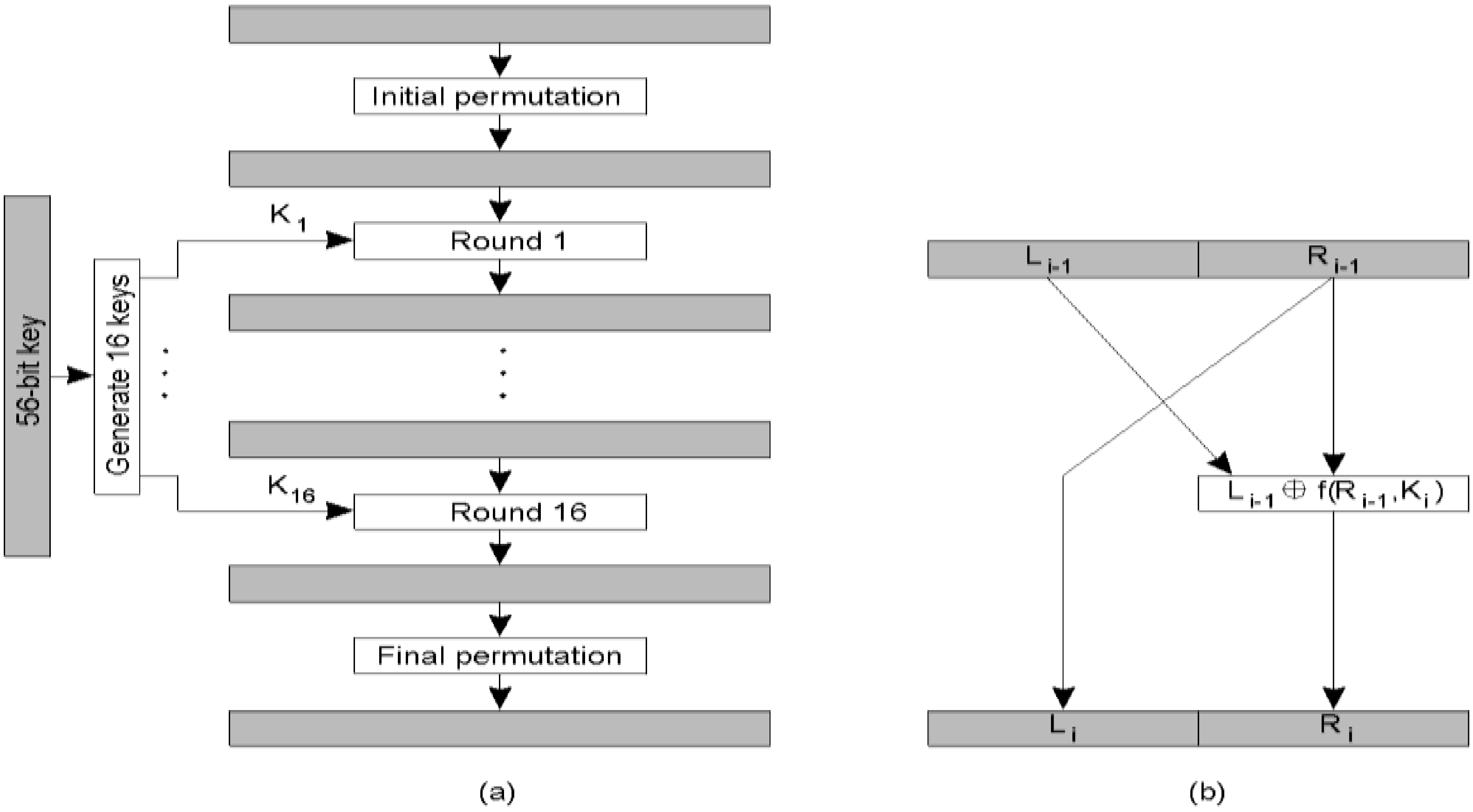
GreyWire
Replace T1/T3 trunk lines with a long distance, high-speed, secure wireless link.

$10,000 Prize
$10,000 prize claimed for the first MD5 collision.

Secure Archives
We are offering our Valise^{tm} software free of charge. Upgrades are available to create secure archives as well. Click here for more information.

DES (Symmetric key)

Data Encryption Standard. Standard from 1977 to 2001
Replaced by Advanced Encryption Standard (AES)



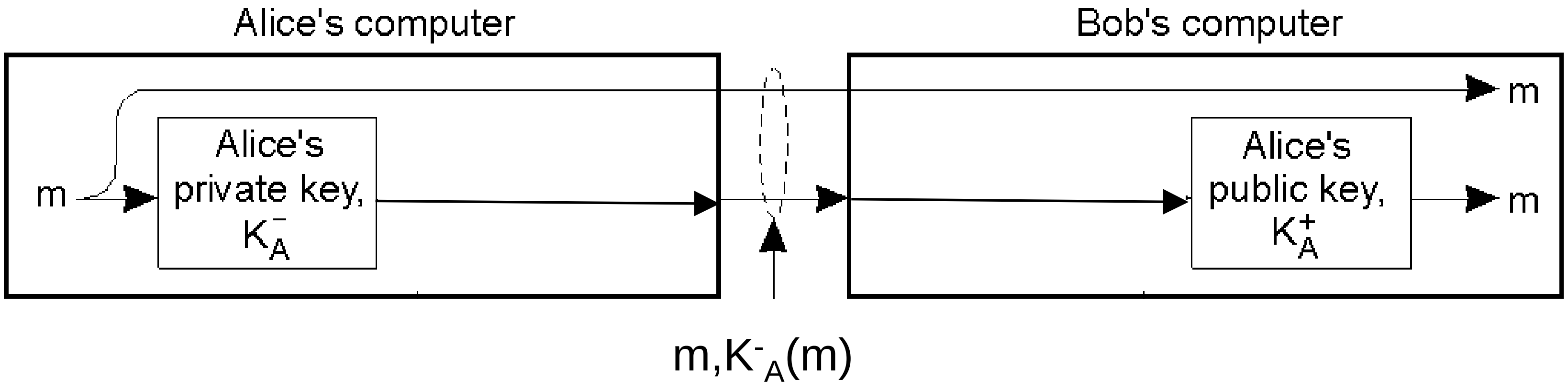
RSA (Public key)

- Based on the difficulty of factorization of large numbers
- Key generation
  - Choose two very large prime numbers, p and q
  - Compute n = p x q and z = (p – 1) x (q – 1)
  - Choose a number d that is relatively prime to z
  - Compute the number e such that e x d = 1 mod z (easy)
- K^{-}_{B}= (d,n) and K^{+}_{B}=(e,n)
- Alice encrypts m: c = m^{e}(mod n) and transmit c
- Bob decripts c: m = c^{d}(mod n)
  - It can be proven that m=m^{de}(mod n)
- Symmetric key algorithm are faster: RSA 100-1000 times slower than DES

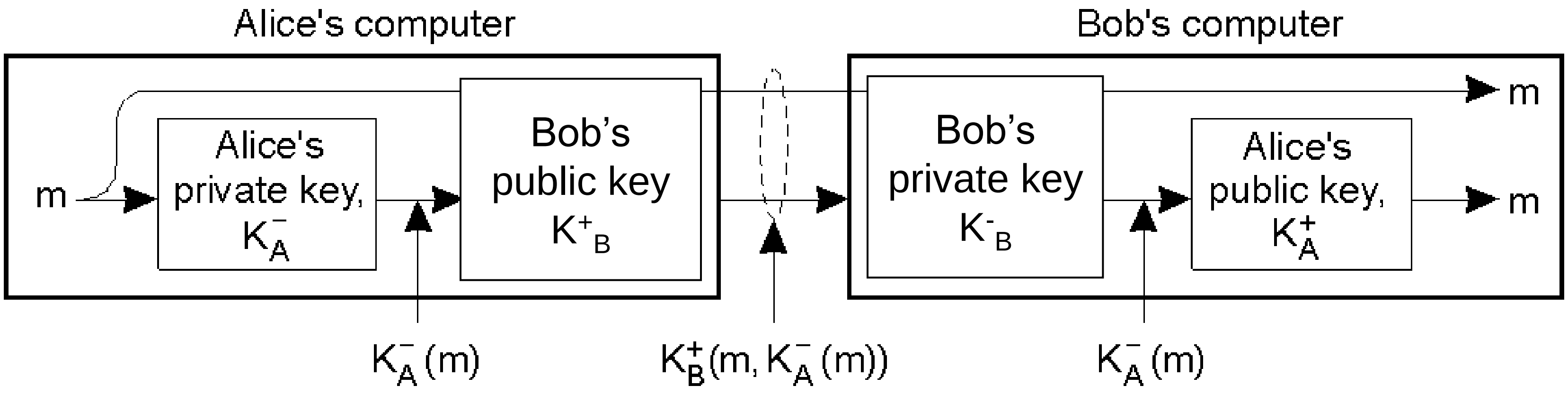
RSA (Public key) - Example

- Simple example with small prime numbers
  - $p = 11$
  - $q = 5$
  - $n = p \times q = 55$
  - $z = (p-1) \times (q-1) = 40$
  - Choose a number d that is relatively prime to z: $d = 13$
  - Compute the number e such that $(e \times d) = 1 \bmod z$: $e = 37$
- Alice wants to encrypt message $m = 16$ to send it to Bob
  - Alice's part of the secret is $(e, n) = (37, 55)$
  - Alice encrypts m : $c = m^e (\mathrm{mod}\ n) = 36$
  - Alice sends $c = 36$ to Bob
  - Bob's part of the secret is $(d, n) = (13, 55)$
  - Bob decrypts m : $m = c^d (\mathrm{mod}\ n) = 16$

Digital Signatures

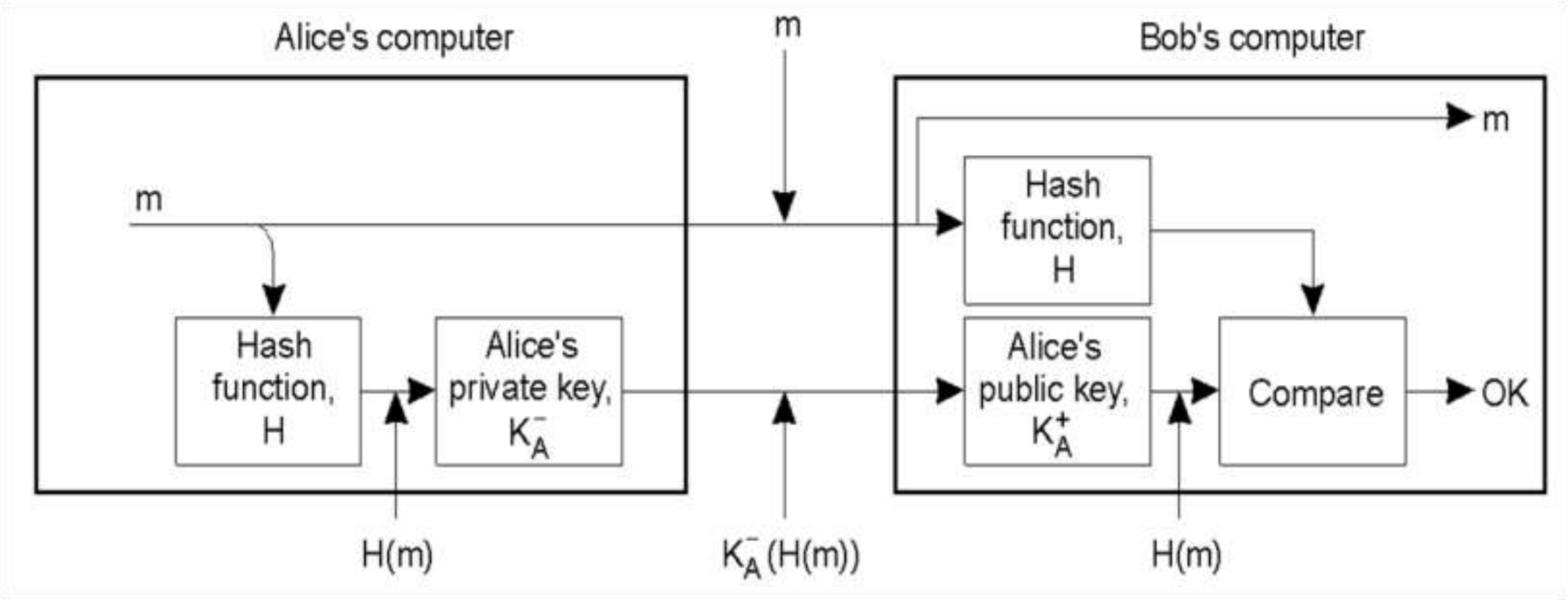


Signing



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*?
- Authentication of public keys
  - This is done by means of public-key certificates
  - A public key certificate is a tuple
    - Identity
    - Public key
    - Signed by a Certification Authority (CA)
  - The public key of CA is assumed to be well-known
    - The basic idea: pervasive information is hard to alter, if a CA public key is everywhere it is really hard to alter every copy without being noticed
  - Yet 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

Certification Authorities

- Certificates lifetimes
  - Protect from compromise
    - A certificate associates public-key and the owner
- Secret key is compromised. Now what?
  - 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

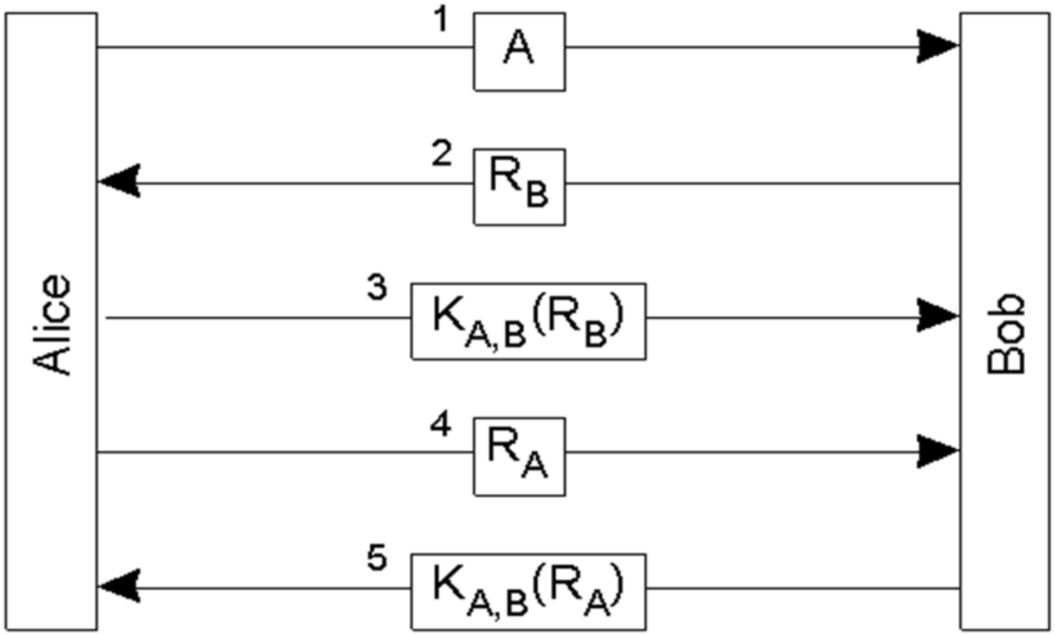
Secure channels

- For secure communication in a distributed systems we can exchange information through secure channels
- 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
  - If a message is never modified but I do not know the source it's not very useful
- 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
  - Auth protocols tackle how to verify this common information without disclosing it on the channel

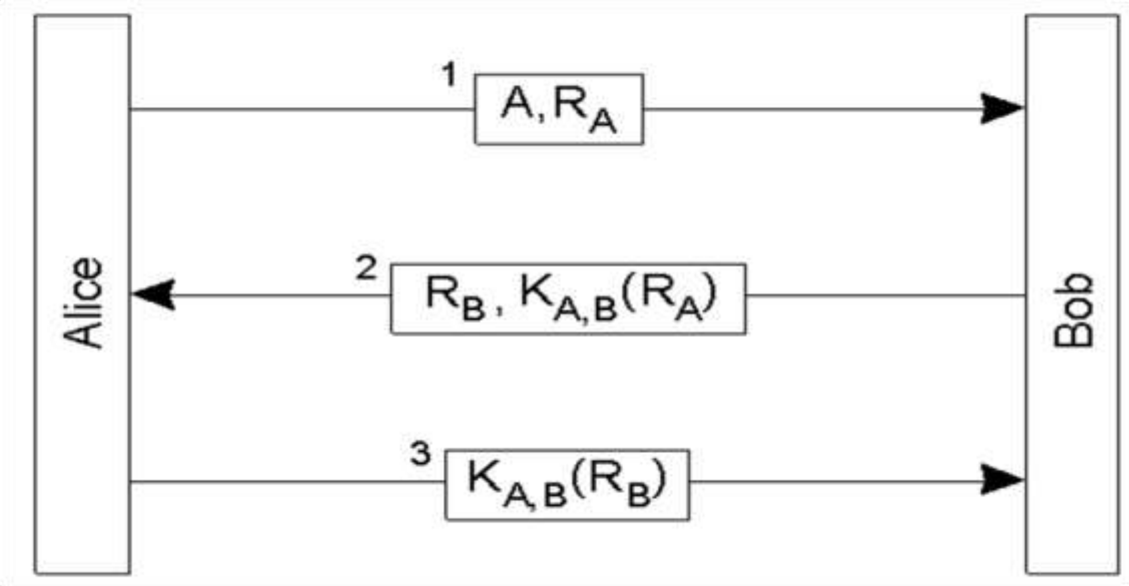
Secure channels



- Authentication with shared secret key
  - challenge-response

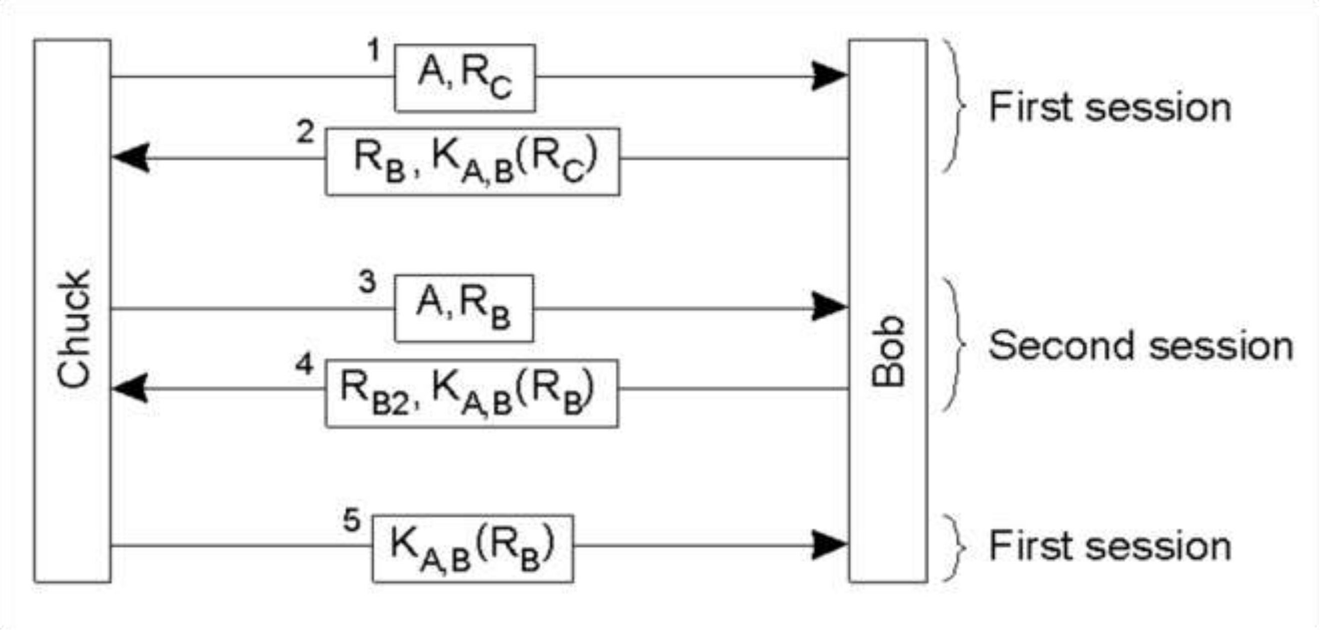
Politecnico
di Milano

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
- But 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”)

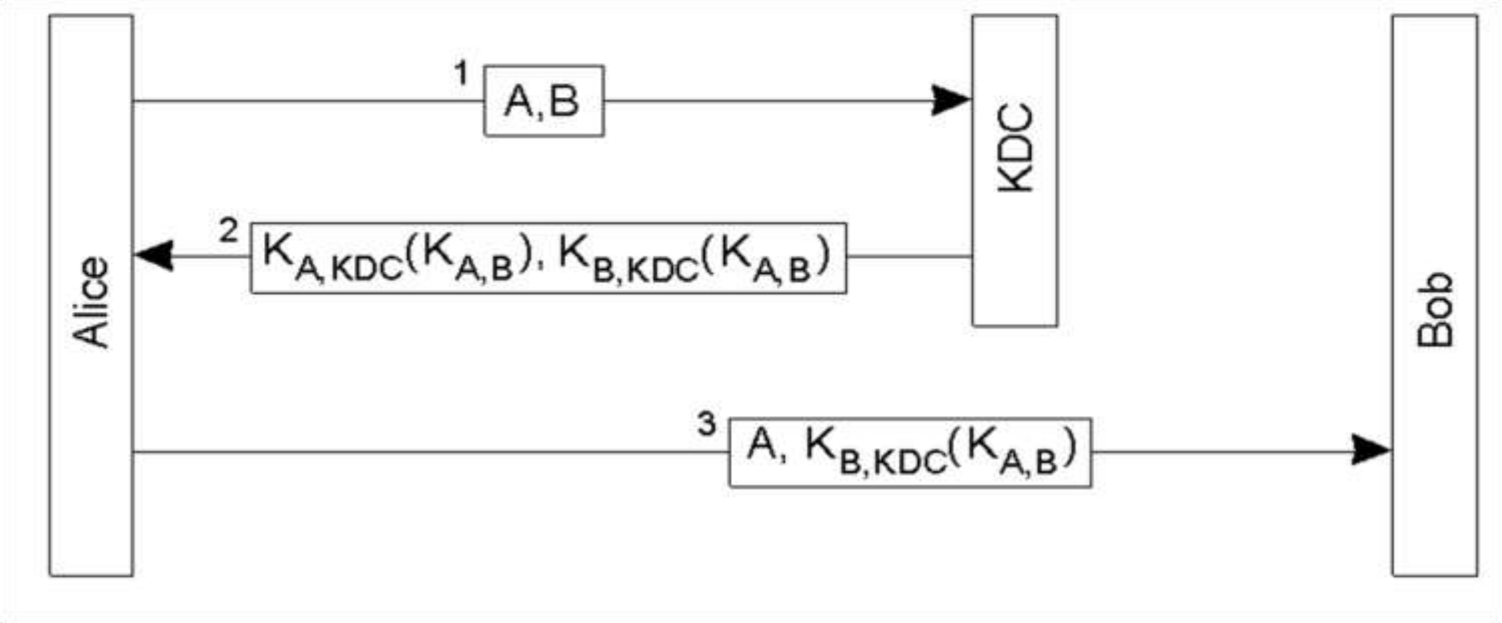
Secure channels

- 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 session is closed the session key must be destroyed

Secure channels

- One of the problems with using 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!
  - 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

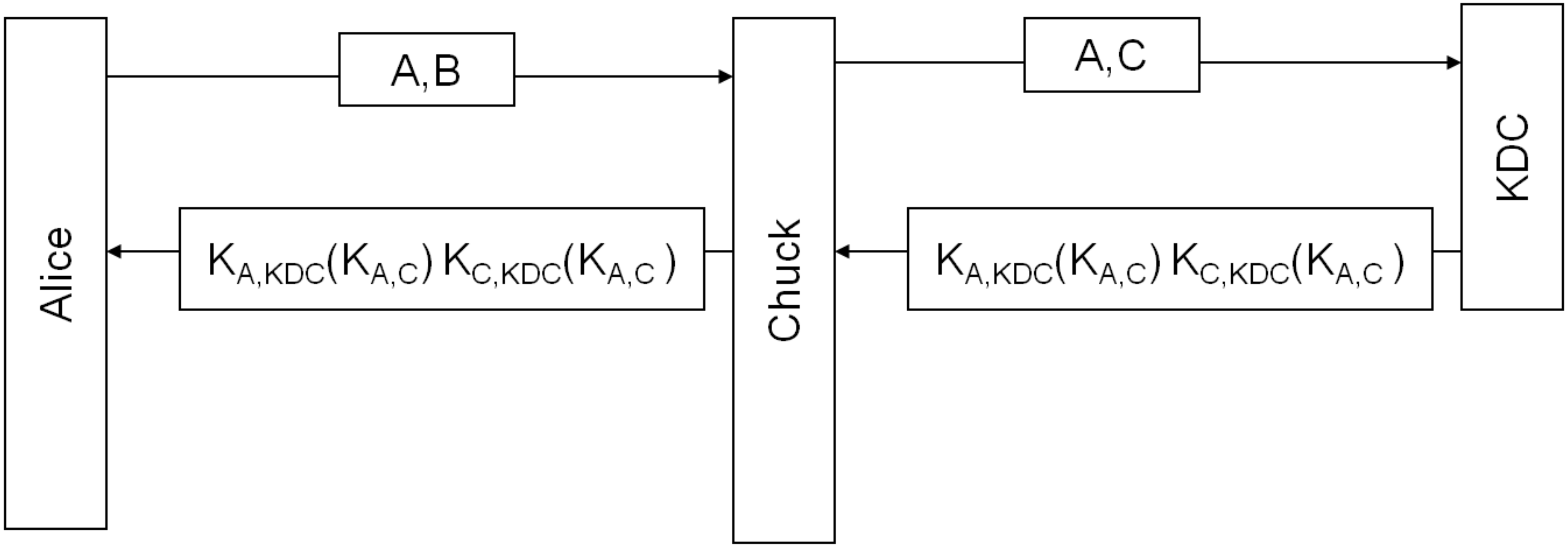
Secure channels



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

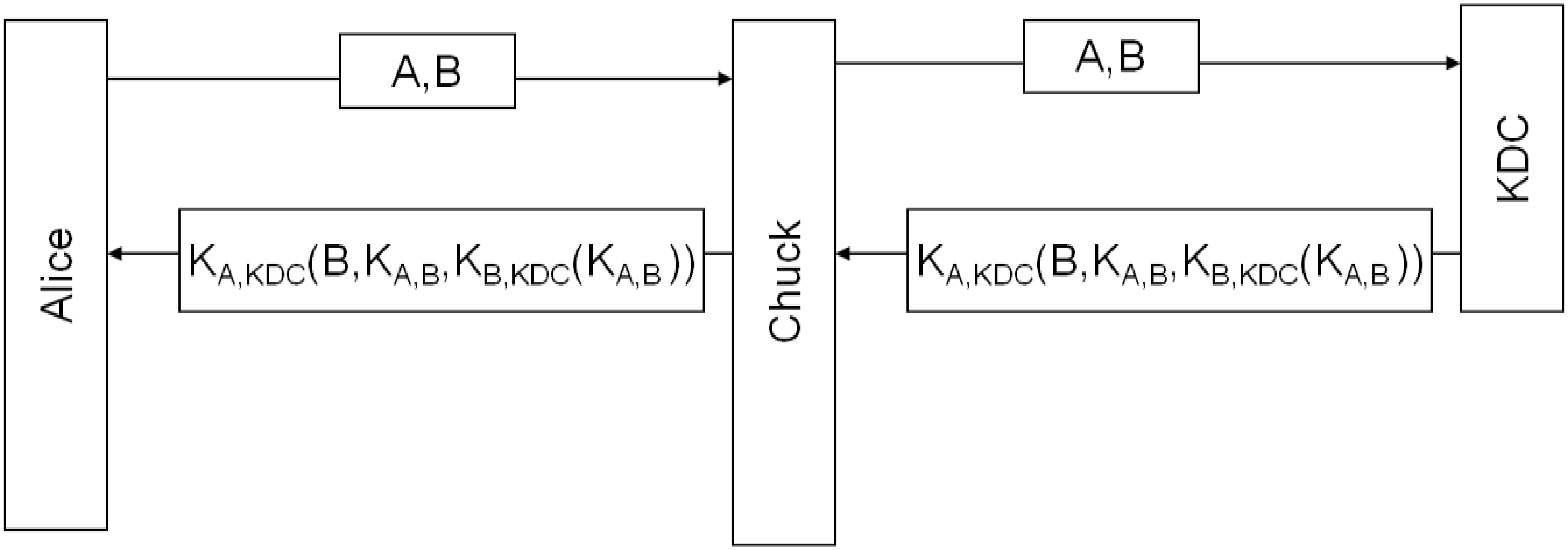
Politecnico di Milano

Secure channels


Alice
A,B
Chuck
A,C
KDC
$K_{A,KDC}(K_{A,C})\ K_{C,KDC}(K_{A,C})$
$K_{A,KDC}(K_{A,C})\ K_{C,KDC}(K_{A,C})$

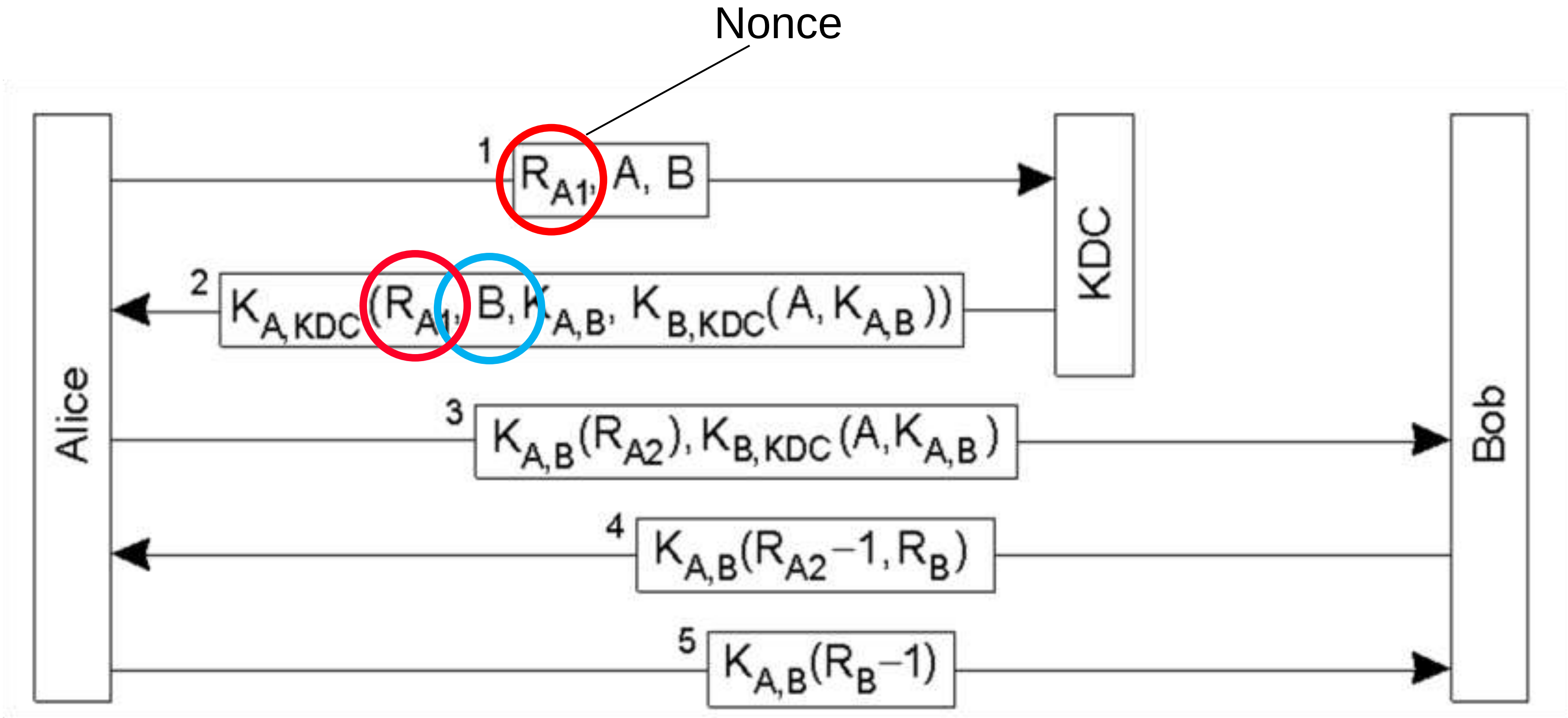
- Modification of the first message
- We could encrypt A,B: $K_{A,KDC}(A,B)$
- Unfortunately Chuck can replay an old $K_{A,KDC}(A,C)$

Secure channels


Alice
A,B
Chuck
A,B
KDC
$K_{A,KDC}(B,K_{A,B},K_{B,KDC}(K_{A,B}))$
$K_{A,KDC}(B,K_{A,B},K_{B,KDC}(K_{A,B}))$

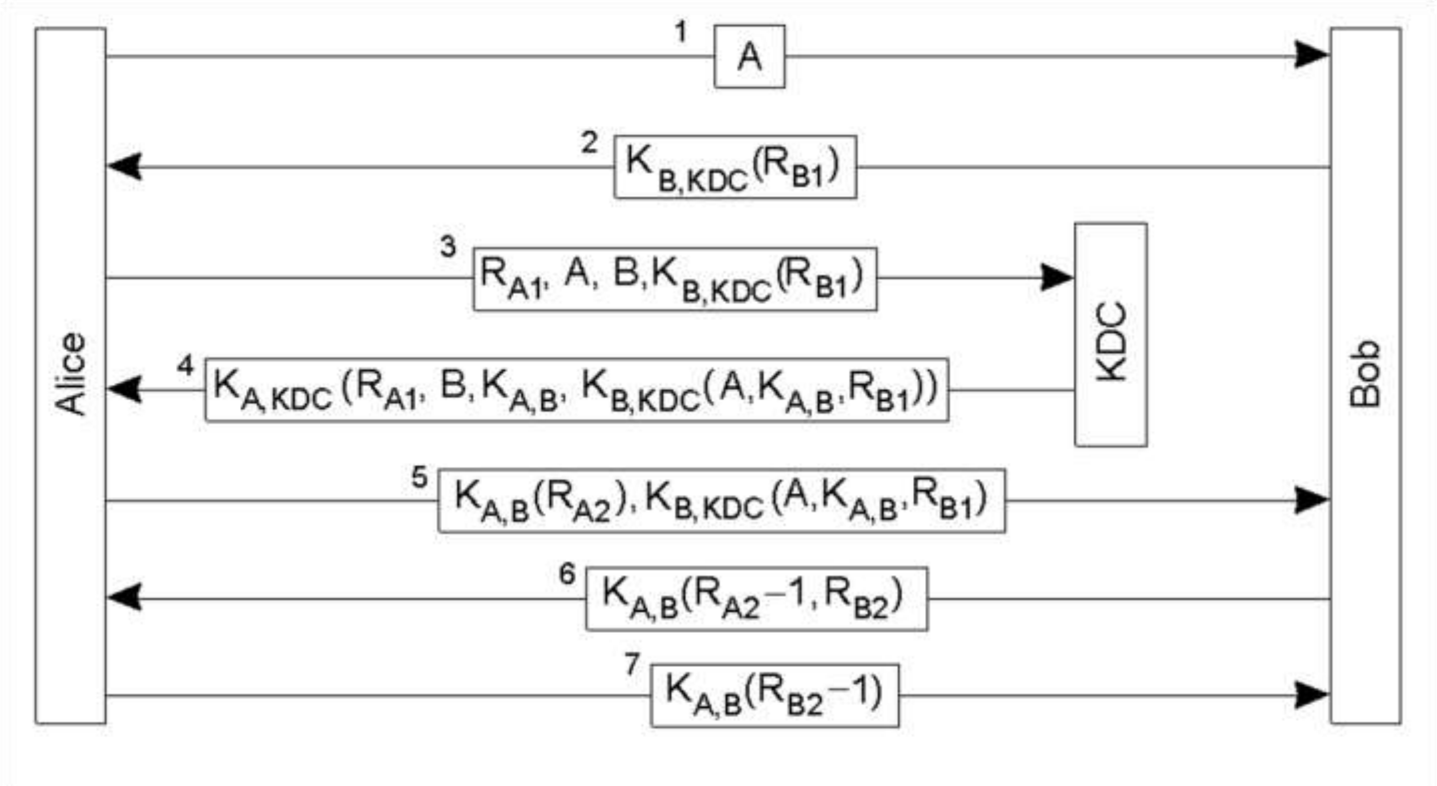
- Solution: put B in the response from KDC and protect it with $K_{A,KDC}$
- Chuck cannot modify A's request.
- But… after stealing $K_{B,KDC}$ he can serve an old KDC reply forever (even after a fresh negotiation of $K_{B,KDC}$ between Bob and KDC)

Secure channels


Nonce
Alice
KDC
Bob
1 R_{A1}, A, B
2 $K_{A,KDC}(R_{A1}, B, K_{A,B}, K_{B,KDC}(A, K_{A,B}))$
3 $K_{A,B}(R_{A2}), K_{B,KDC}(A, K_{A,B})$
4 $K_{A,B}(R_{A2}-1, R_B)$
5 $K_{A,B}(R_B-1)$

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

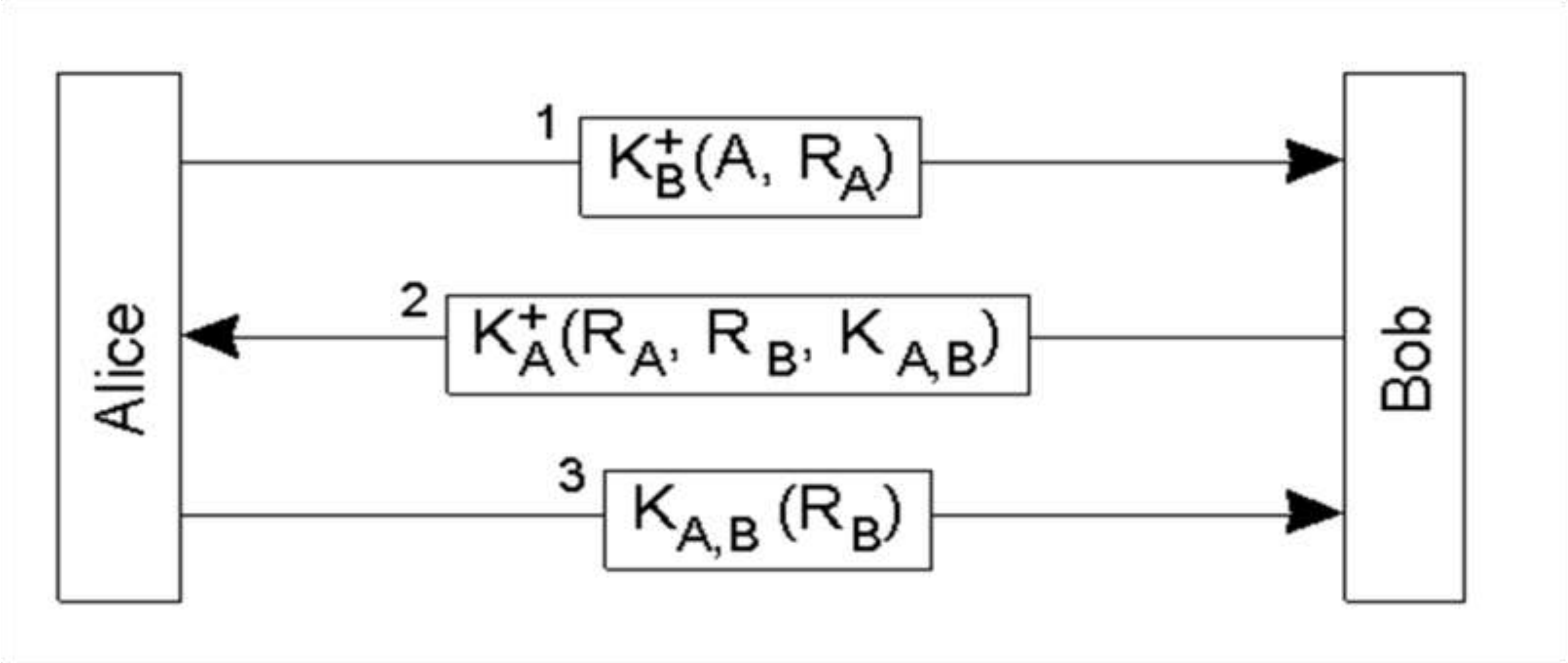
Secure channels


Alice
KDC
Bob
1
A
2
$K_{B,KDC}(R_{B1})$
3
R_{A1}, A, B, $K_{B,KDC}(R_{B1})$
4
$K_{A,KDC}(R_{A1}, B, K_{A,B}, K_{B,KDC}(A, K_{A,B}, R_{B1}))$
5
$K_{A,B}(R_{A2}), K_{B,KDC}(A, K_{A,B}, R_{B1})$
6
$K_{A,B}(R_{A2}-1, R_{B2})$
7
$K_{A,B}(R_{B2}-1)$

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

Politecnico
di Milano

Secure channels

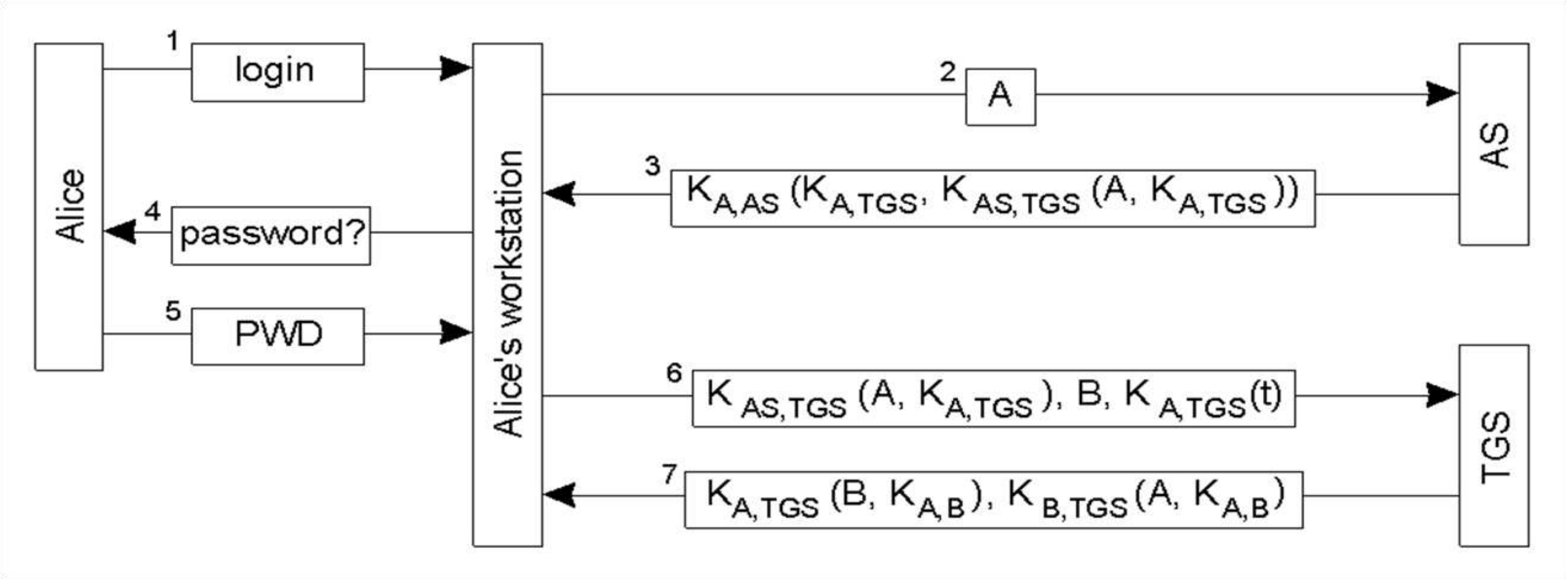


Authentication Using Public-Key Cryptography

Kerberos

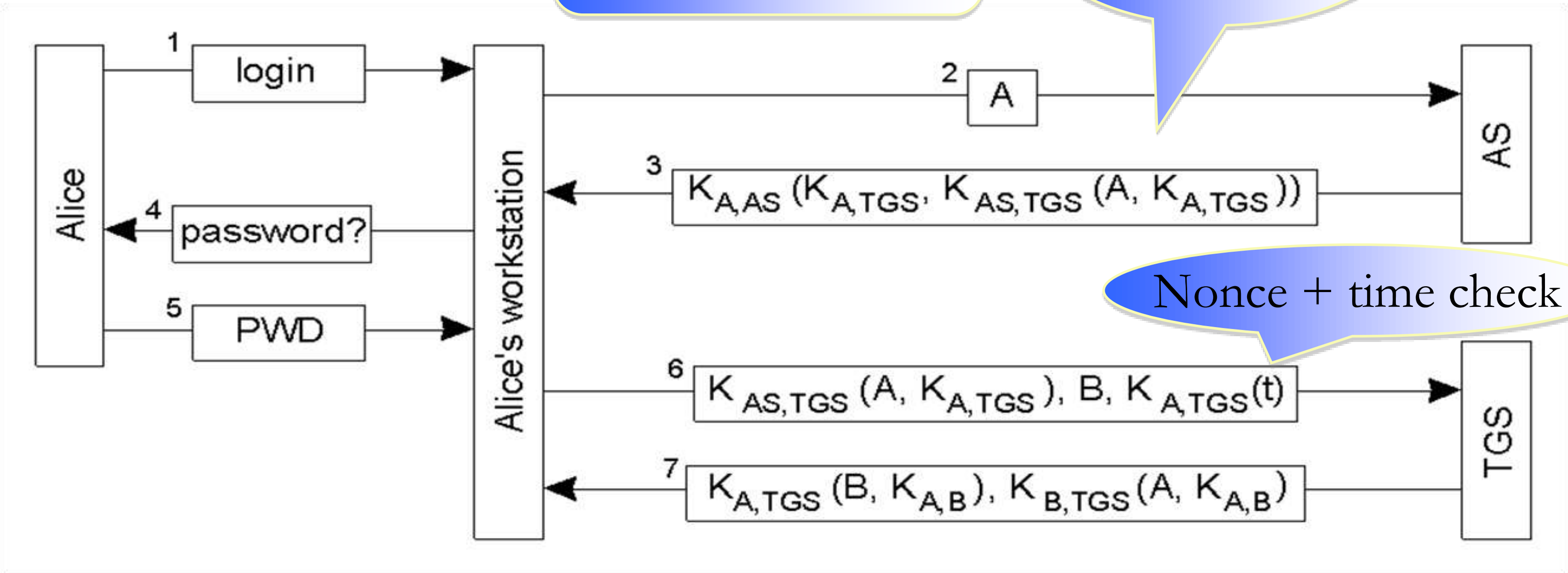
- Authentication and security facilities for intranets
  - Born in the 1980s @ MIT
  - Version 5 is a RFC
- Three security objects:
  - Ticket: to be presented to a server, granting user authentication to the system
  - Authentication: token proving user identity
  - Session key: randomly produced to be used by a client when communicating with a server (used only if needed)

Kerberos


Alice
1
login
Alice's workstation
2
A
AS
3
$K_{A,AS}(K_{A,TGS}, K_{AS,TGS}(A, K_{A,TGS}))$
4
password?
5
PWD
6
$K_{AS,TGS}(A, K_{A,TGS}), B, K_{A,TGS}(t)$
TGS
7
$K_{A,TGS}(B, K_{A,B}), K_{B,TGS}(A, K_{A,B})$

- Kerberos authentication uses Needham-Schroeder. KDC is split in two:
  - An Authentication Server (AS)
    - Communication with AS is simplified by the fact that the destination is always the TGS
  - A Ticket Granting Service (TGS)

Kerberos


1
login
2
A
3
$K_{A,AS}(K_{A,TGS}, K_{AS,TGS}(A, K_{A,TGS}))$
4
password?
5
PWD
6
$K_{AS,TGS}(A, K_{A,TGS}), B, K_{A,TGS}(t)$
7
$K_{A,TGS}(B, K_{A,B}), K_{B,TGS}(A, K_{A,B})$
Alice
Alice's workstation
AS
TGS

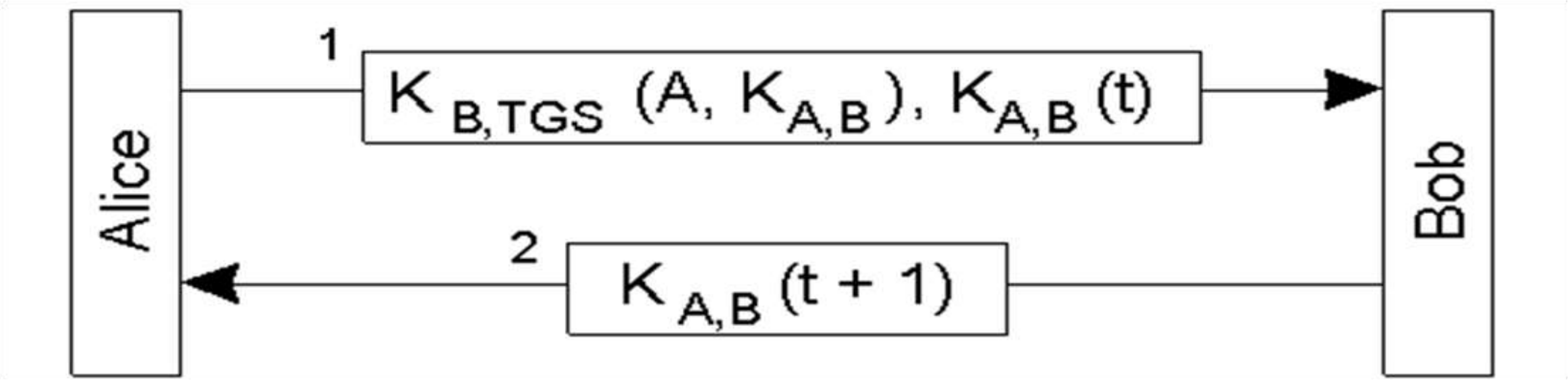
Time limit prevents from $K_{A,TGS}$ crack

Valid for a limited time

Nonce + time check

- The password is used to generate $K_{A,AS}$ but there's no need to send it to other hosts or even to temporary store it in workstation
- Needham-Schroeder simplified because destination is always TGS

Kerberos



- The scheme establishes what is known as single sign-on (within the time limit imposed by the ticket)
  - As long as Alice does not change workstation, there is no need for her to authenticate herself again to any other host of the distributed system
- Setting up a secure channel is easy:
  - Bob can be sure message 1 is coming from Alice because of $K_{B,TGS}$
  - By responding with message 2 Bob proves his own identity to Alice

Secure group communication

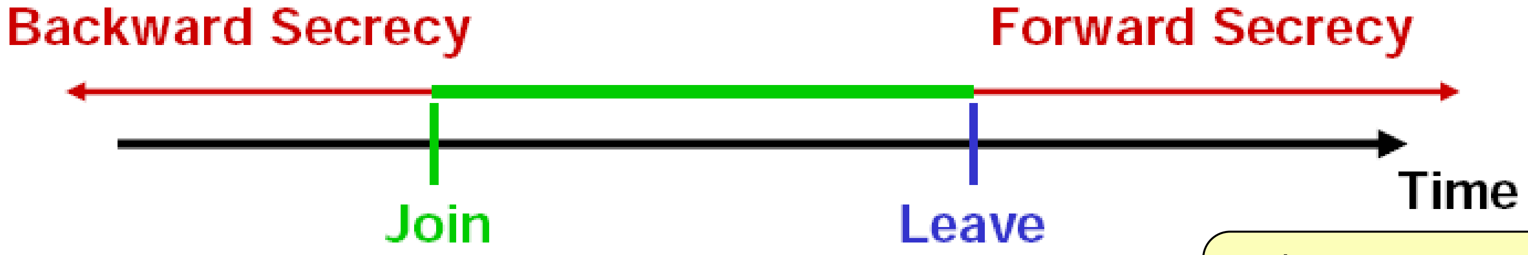
Secure Group Communication

- 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^2-1)/2	2n	1

Secure Group Communication

- Joining and leaving a group
  - **requirement**: *Backward* and *Forward* secrecy
  - *Backward*: cannot decrypt messages before join
  - *Forward*: 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

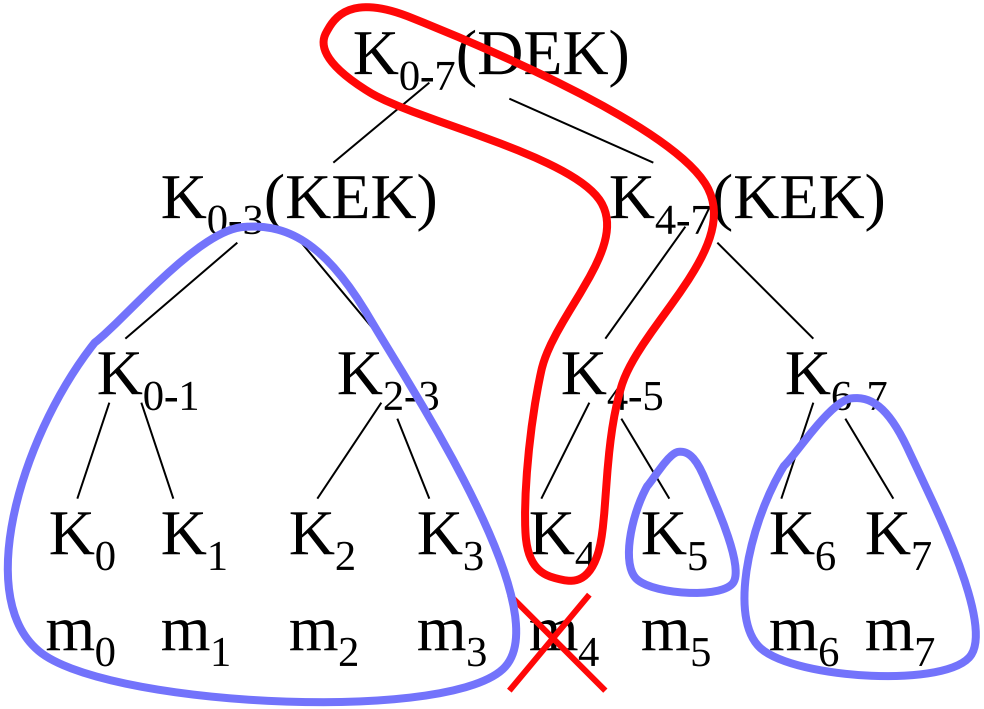
Secure Group Communication

- Who will choose the new key?
- The server
  - Key Distribution Problem
    - We will see two examples of efficient techniques for performing key distribution
- The participants
  - Key Agreement Problem
    - We will see Diffie-Hellman for 2 parties
    - It can be generalized to more than two participants

Politecnico di Milano

Efficient Key Distribution

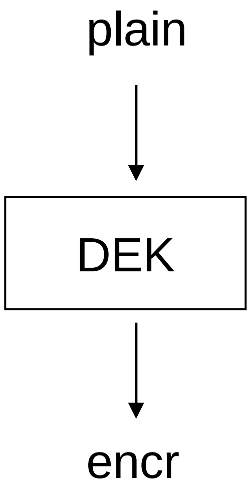
- Logical Key (Tree) Hierarchy
  - leaves: members with keys
  - root: (Data Encryption) Key
  - each member knows (Key Encryption) Keys up to root
- Leave:
  - change all keys known by the leaving member
    - Data Encryption Key
    - change keys on path to root
  - diffuse efficiently the new keys
    - encrypt each key with childs
    - exploit stable subtrees for key distribution


K_{0-7}(DEK)
K_{0-3}(KEK)
K_{4-7}(KEK)
K_{0-1}
K_{2-3}
K_{4-5}
K_{6-7}
K_0 K_1 K_2 K_3 K_4 K_5 K_6 K_7
m_0 m_1 m_2 m_3 m_4 m_5 m_6 m_7

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

Efficient Key Distribution

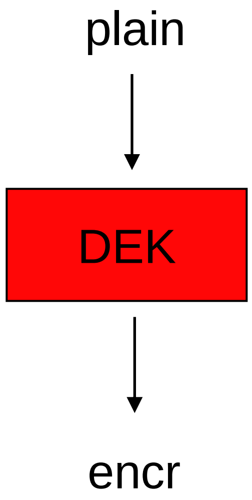
- Centralized Flat Table
  - 2 KEKs for each bit of member ID + 1 DEK
    - (= 2*log(n)+1 vs 2n-1 in LKH)
  - Each node has a key for each bit in its ID
    - Node 9=(1001) has keys $K_{0,1}$ $K_{1,0}$ $K_{2,0}$ $K_{3,1}$ + DEK
    - If 9 leaves all these keys + DEK must be changed

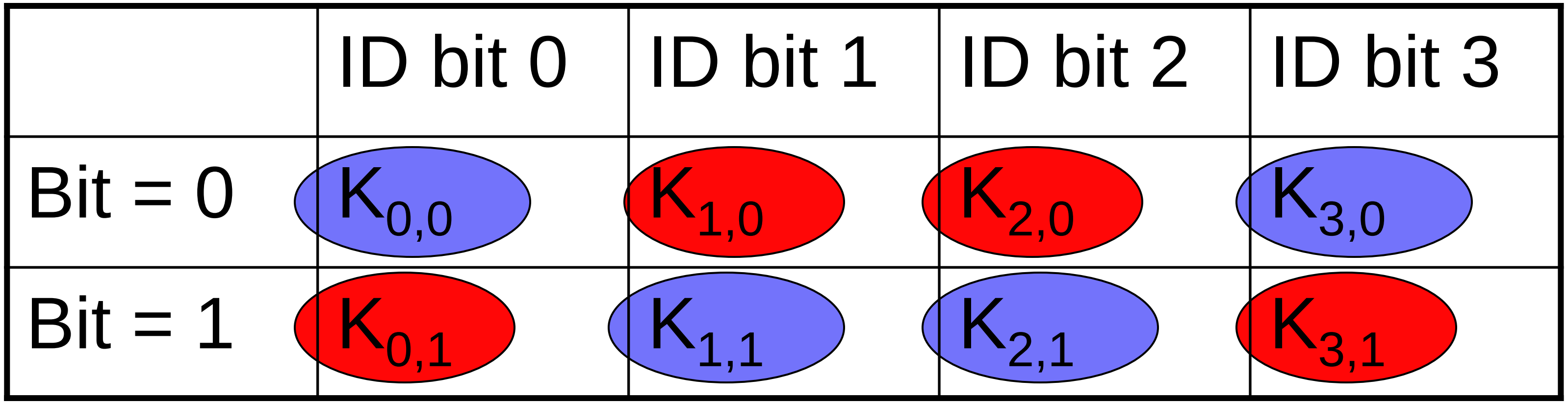


	ID bit 0	ID bit 1	ID bit 2	ID bit 3
Bit = 0	$K_{0,0}$	$K_{1,0}$	$K_{2,0}$	$K_{3,0}$
Bit = 1	$K_{0,1}$	$K_{1,1}$	$K_{2,1}$	$K_{3,1}$

Efficient Key Distribution

- Centralized Flat Table
  - Encrypt DEK’ with every other 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
  - Collusion attack
    - Difficult to remove many participants



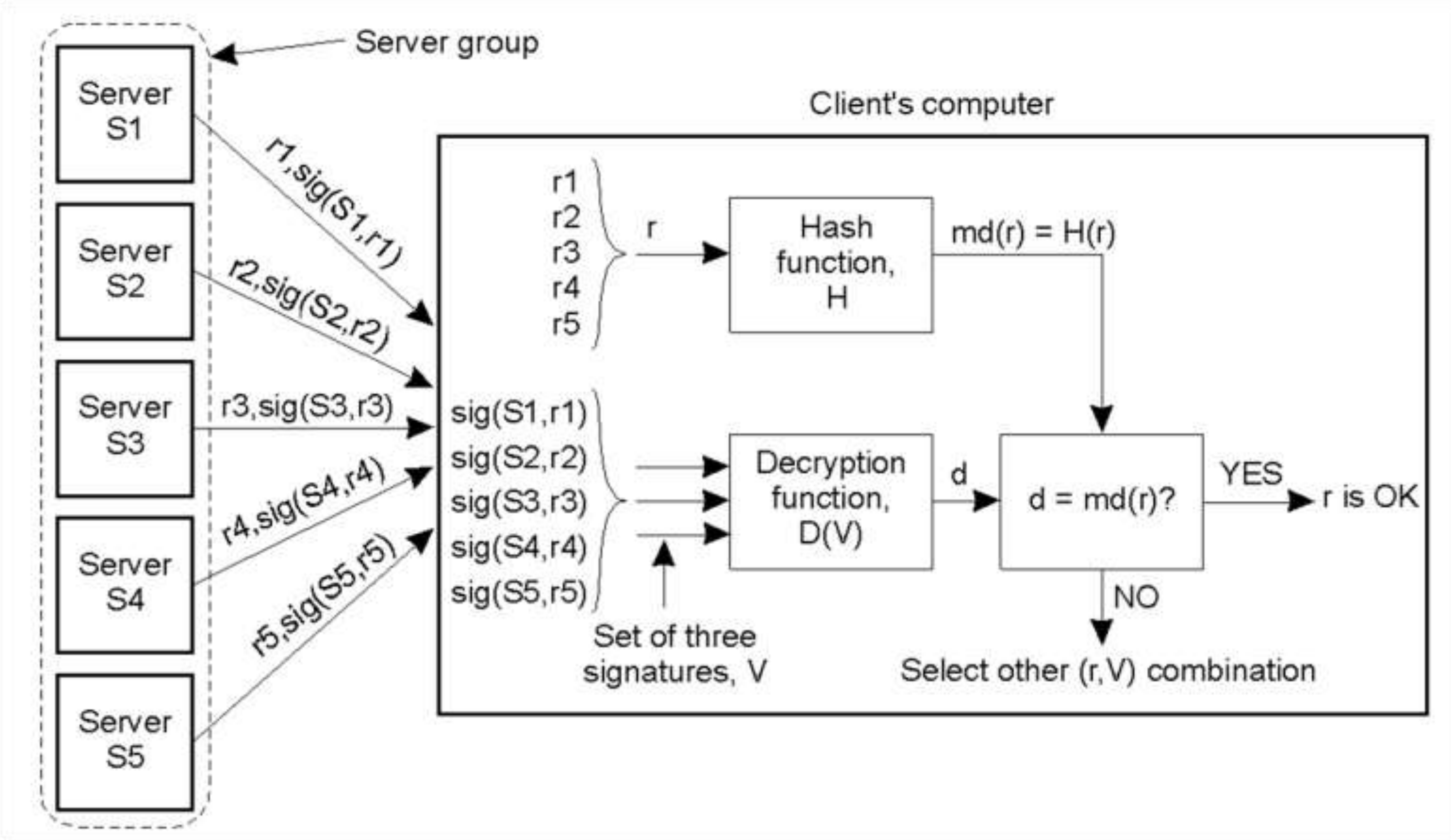


	ID bit 0	ID bit 1	ID bit 2	ID bit 3
Bit = 0	$K_{0,0}$	$K_{1,0}$	$K_{2,0}$	$K_{3,0}$
Bit = 1	$K_{0,1}$	$K_{1,1}$	$K_{2,1}$	$K_{3,1}$

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 decide on a majority. However this forces the client to know the identity (and the public key) of all the 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
  - ri response from server i
  - sig(Si,ri) signature from server i of ri

Secure replicated servers



Sharing a secret signature in a group of replicated servers

Security management

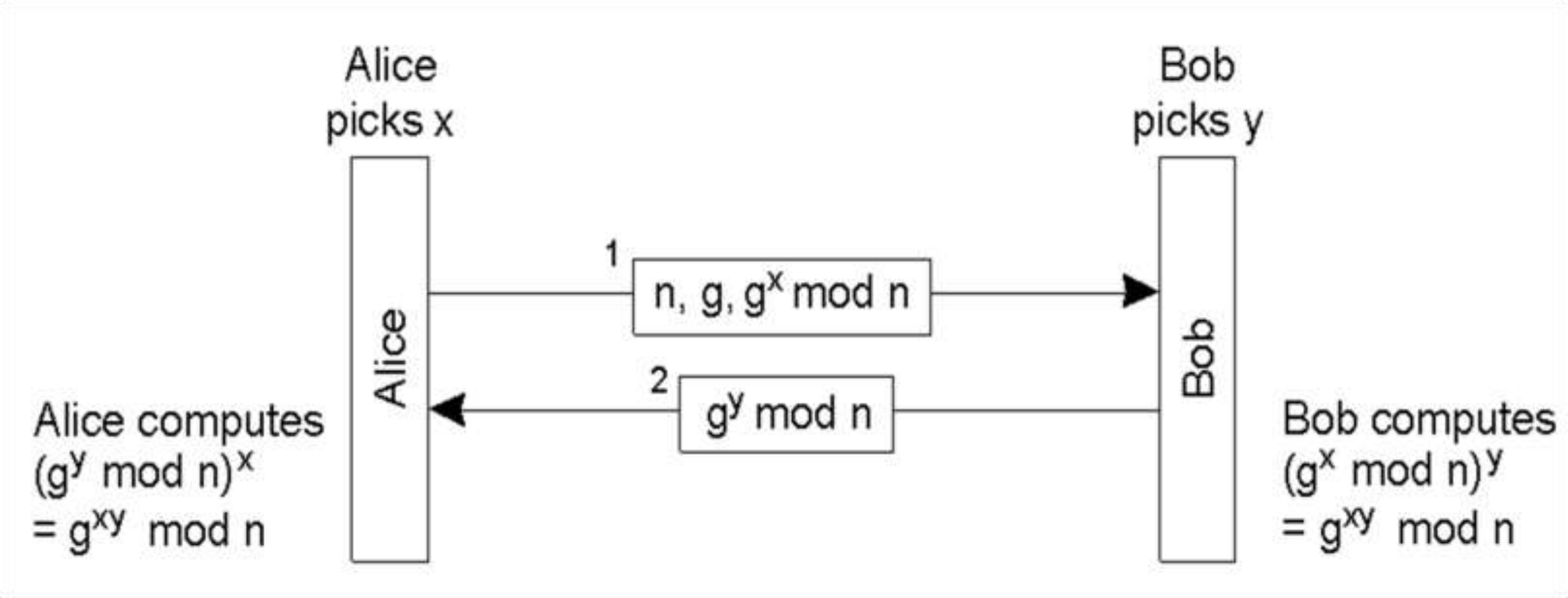
Security Management

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

Security Management

- Initial key distribution
  - We said that for authentication protocols we need one of
    - shared secret keys (between each party or with the KDC)
    - public key of the recipient of the message
  - Now we want to investigate how these initial keys can be distributed in a secure way
  - The first algorithm we consider is Diffie-Hellman key exchange
    - algorithm to exchange symmetric keys on an insecure channel

Security Management



- Diffie-Hellman key exchange
  - n and g publicly known numbers (with some good math property)
- Is this a solution?

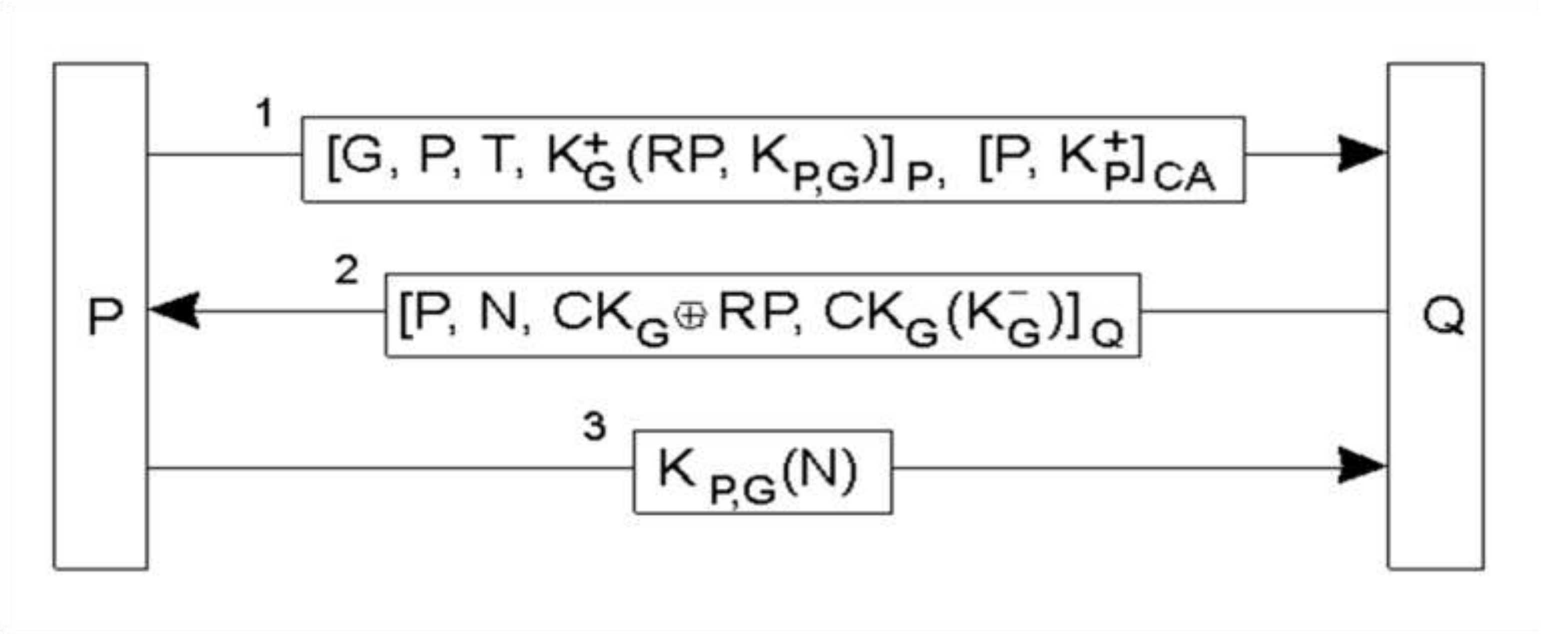
Security Management

- 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 (in particular authentication)

Security Management

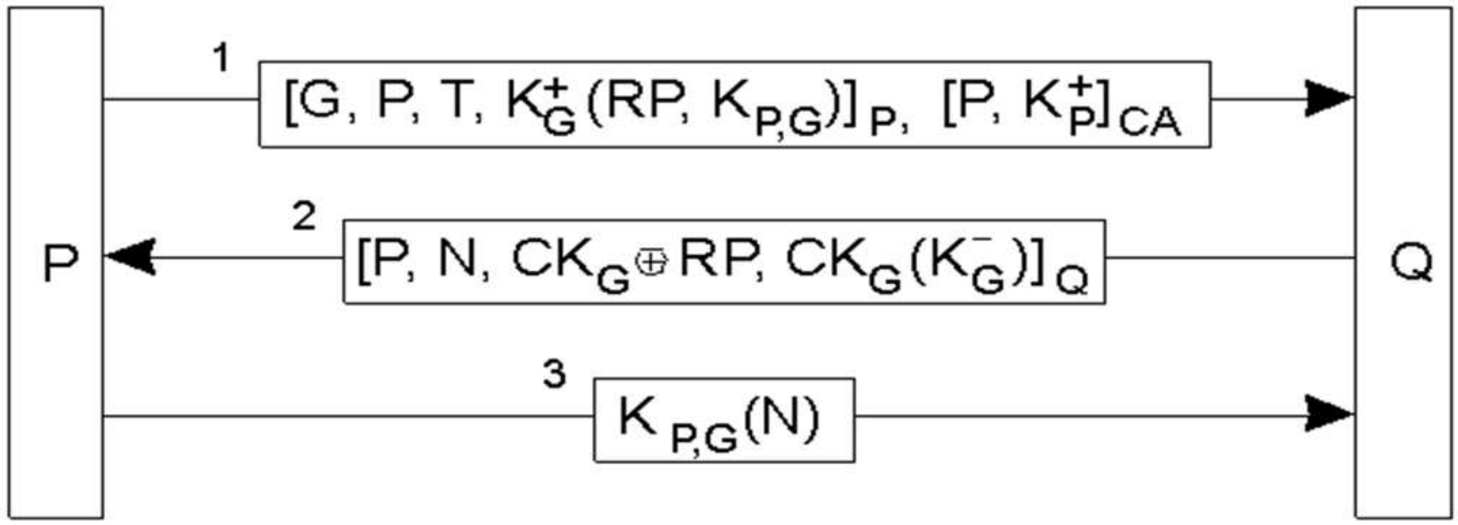
- Many systems make use of special services such as Key Distribution Center (KDC)
- These services demonstrate a difficult problem in distributed systems
  - First of all, they must be trusted
  - Second, they must offer high availability
  - Solution to high availability is replication …
  - … but replication makes a server more vulnerable to security attacks
- We have seen how secure group communication takes place, the problem that needs to be solved is how to manage a group of replicated servers
  - In particular, if a process joins or leaves the group, the integrity of the group should not be compromised

Security Management


P
Q
1
$[G, P, T, K_G^+(RP, K_{P,G})]_P, [P, K_P^+]_{CA}$
2
$[P, N, CK_G \oplus RP, CK_G(K_G^-)]_Q$
3
$K_{P,G}(N)$

- Secure group management
- P wants to join a group G by contacting a member Q
- RP is a *Reply Pad*, generated and used only ones
- Message 2 signed by Q with $K_{P,G}$
  - Only a member of G could read $K_{P,G}$ from message 1

Security Management



1 $[G, P, T, K_G^+(RP, K_{P,G})]_P, [P, K_P^+]_{CA}$

2 $[P, N, CK_G \oplus RP, CK_G(K_G^-)]_Q$

3 $K_{P,G}(N)$

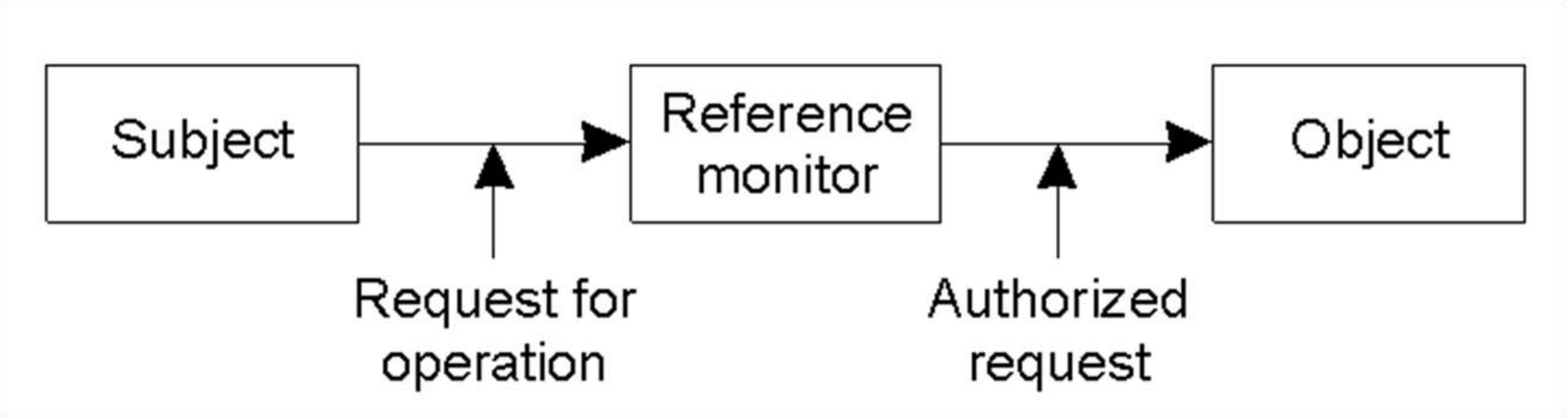
- Why using RP instead of K^+_P ?
- Because RP is used only once in the second message. Using K^+_P would be risky: if K^-_P were ever to reveal, it would become possible to also reveal CK_G which would compromise the secrecy of all group communication

Security Management

- Authorization Management
  - 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 to have a central server that manages authorization rights
    - A better approach is the use of capabilities

Access control

Access Control



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

Access Control

- Access Control Matrix
  - Ideally we have a matrix listing rights for every combination (resource, user)

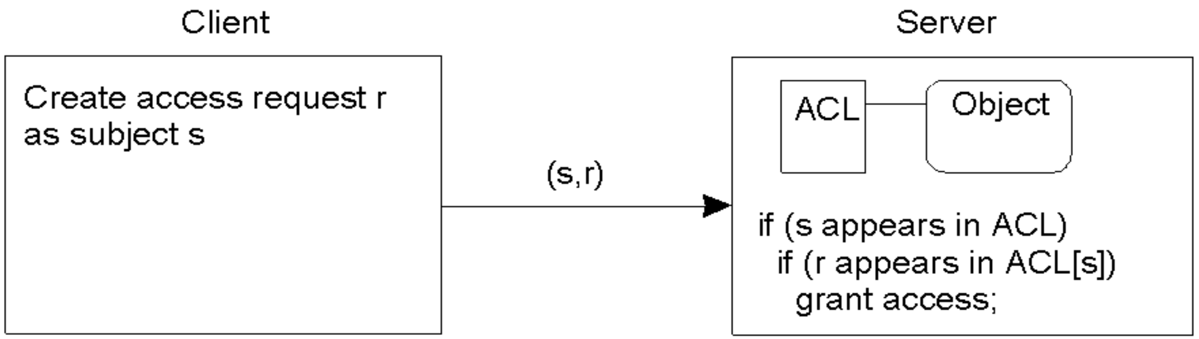
	Object 1	…	Object n
User A	R,W	…	R
…	…	…	…
User k	W	…	W

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

Access Control

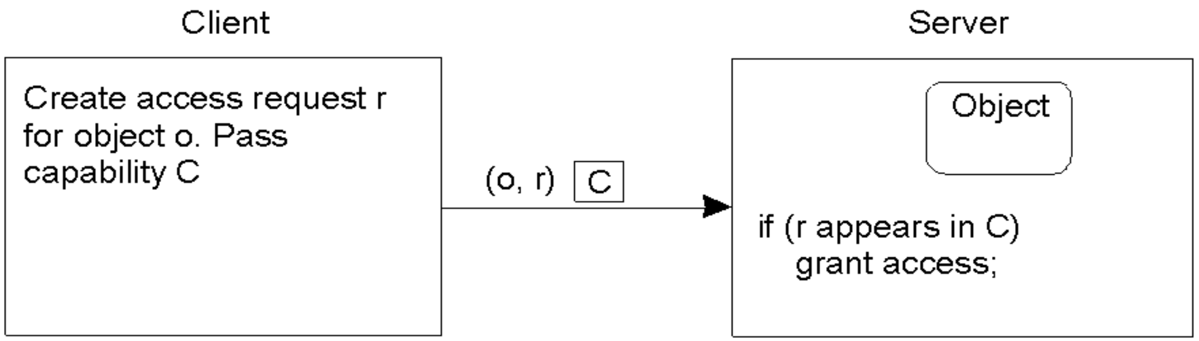
- Using ACL vs. *capabilities lists* affects the way interaction takes place with the Reference Monitor

a) ACL



(a)

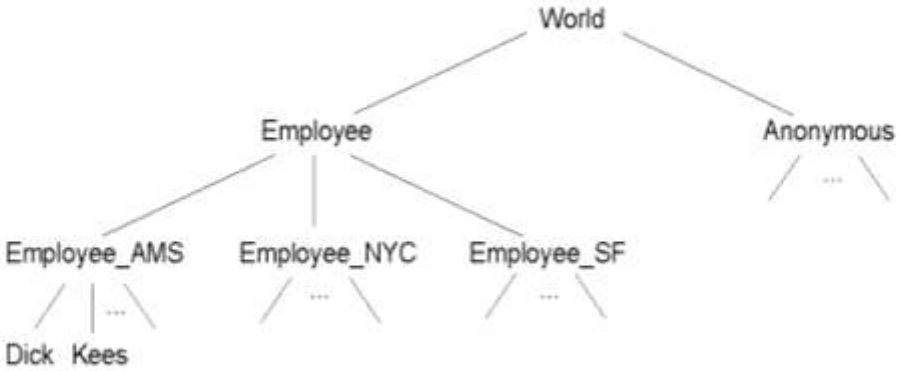
b) capabilities



(b)

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 ore more *roles*
  - often mapped to user's functions in an organization
- When a user logs in, he specifies one of its 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

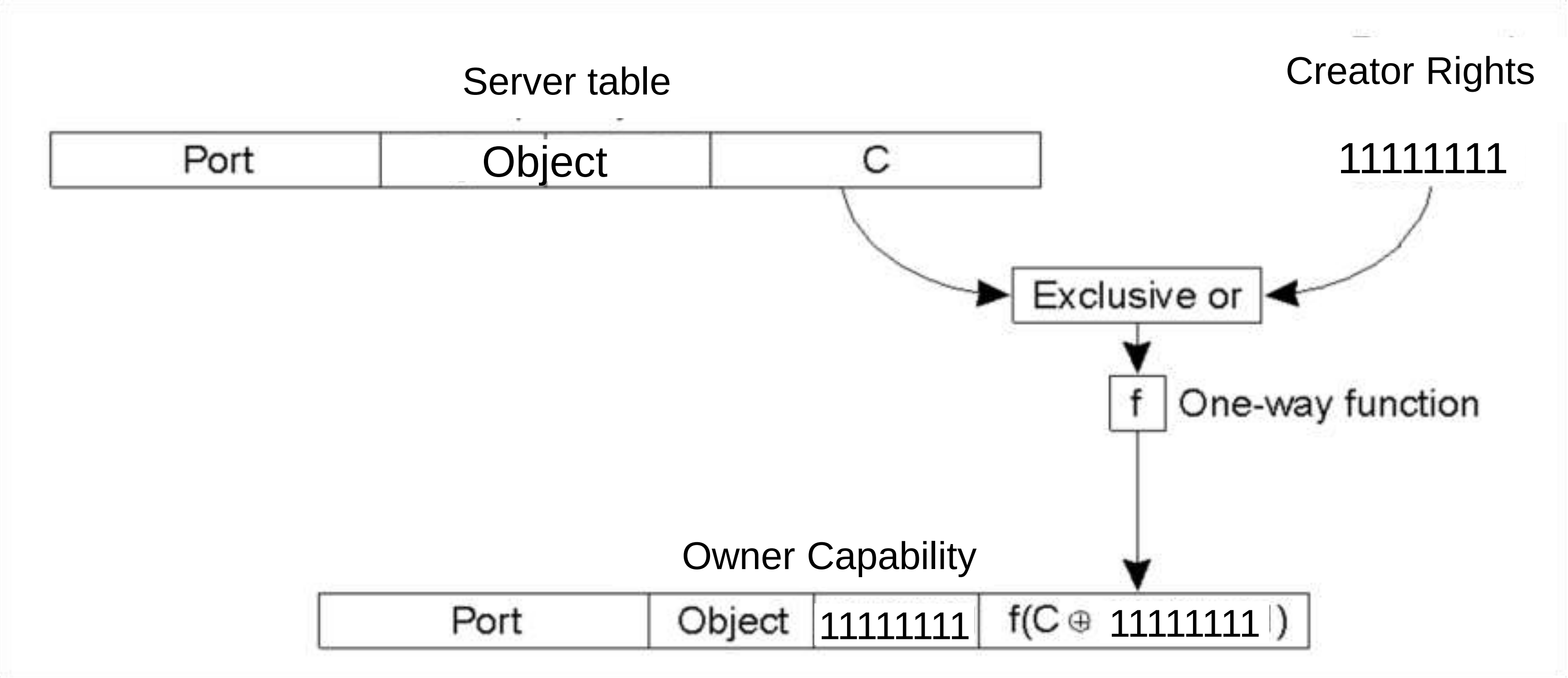
AC - Capabilities

- An example of capabilities (Amoeba)
  - 128 bit

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

  - 72 (48+24) to identify an object
  - 8 for access rights
  - 48 to make it unforgeable

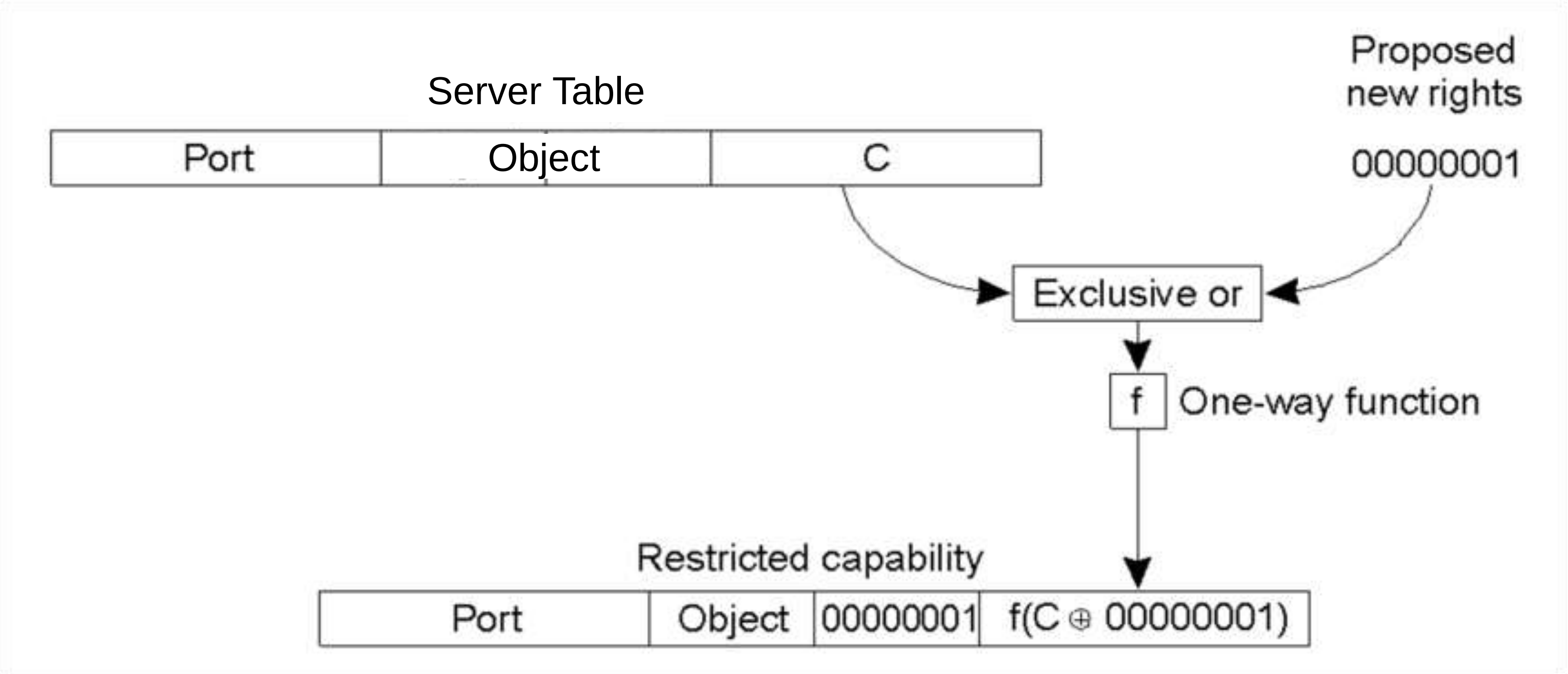
AC - Capabilities



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

Politecnico di Milano

AC - Capabilities

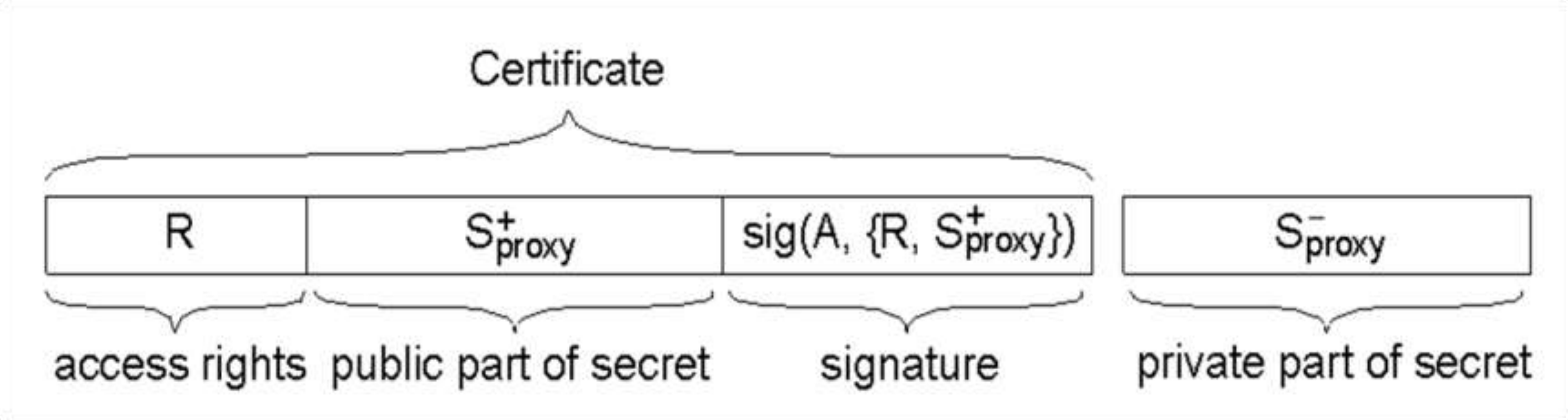


- Changing rights is not possible for the owner of the capability
  - Only the server knows C

AC – Capabilities delegation

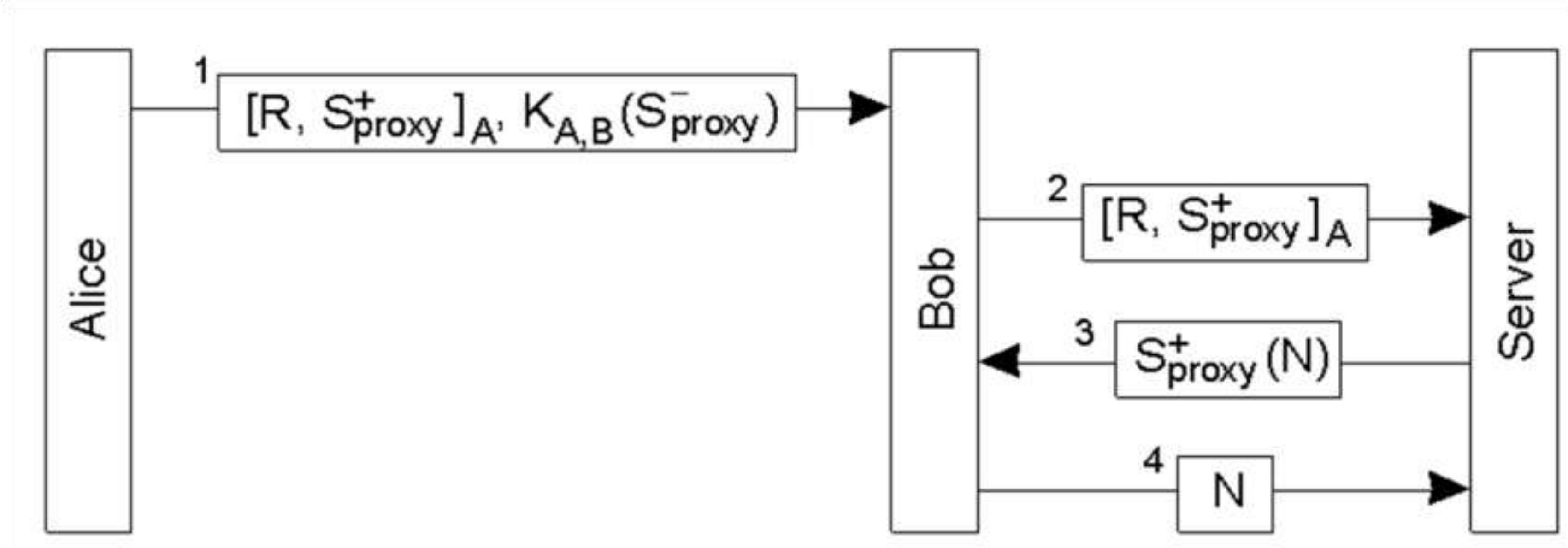
- Delegation
  - A process may want to delegate some of its rights to other processes
    - Amoeba capabilities can only be passed as they are, it is not possible to further restrict rights unless we request a restricted capability to the server
    - A general scheme that support delegation including 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 they own

AC – Proxy



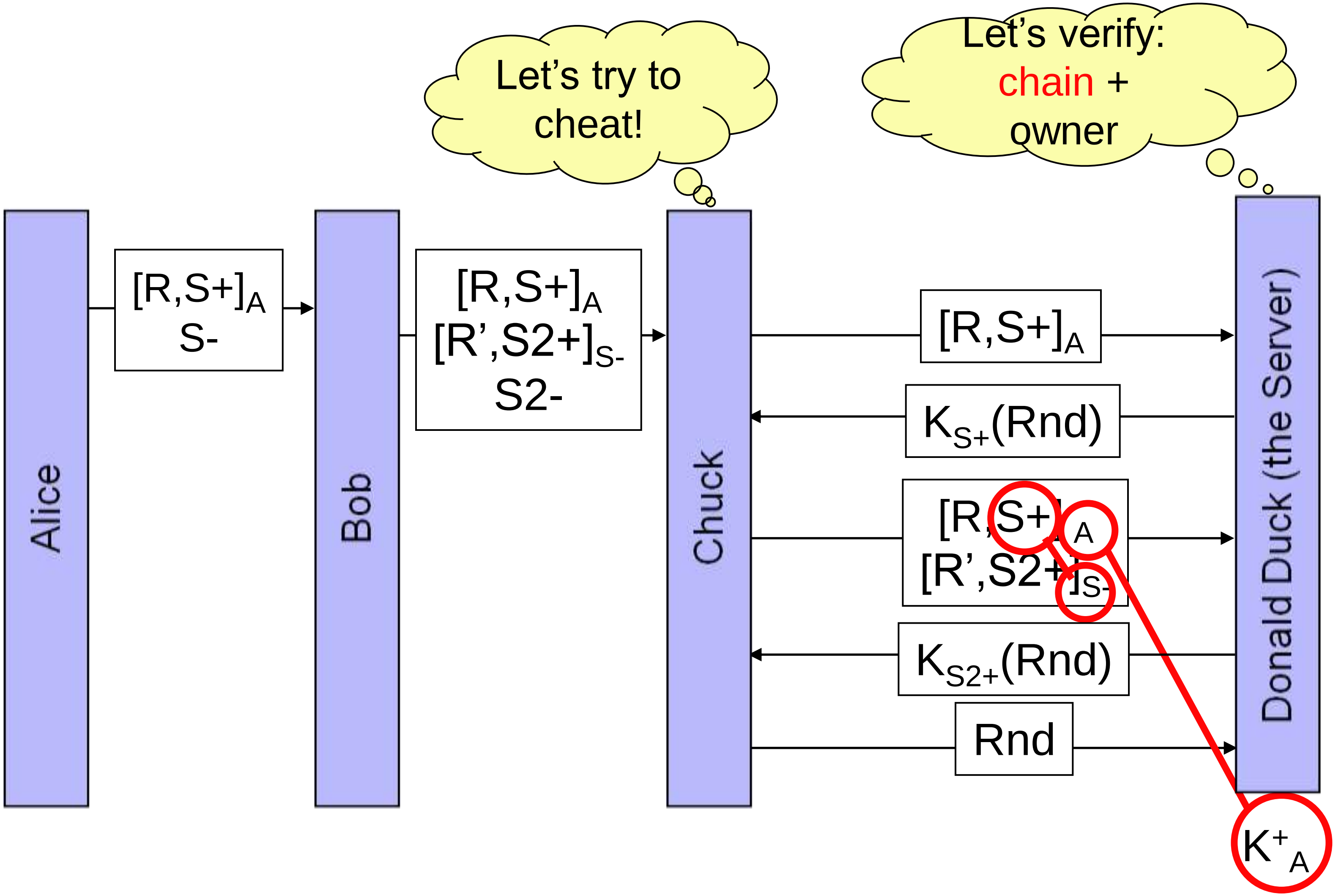
- A proxy has two parts: a certificate and a key
  - The certificate proves that the *grantor* A entitled R rights to some *grantee*
  - The key is the proof that a process is the *grantee*

AC - Proxy



Protocol for delegation and authorization

AC - Restricting a Proxy



Politecnico
di Milano

AC – Restricting a Proxy

- How can we restrict a proxy?
- A -> B -> C -> D (the server)
- B receives $[R,S^{+}]_{A}$ S^{-}
- B places R_2 restriction and transmit to C

  $[R,S^{+}]_{A}$

  $[R_2,S_2^{+}]_{S^-}$ S_2^{-}
- C can’t pretend that he is entitled with R because it doesn’t know S^{-}

Access Control - Mobile code

- Protection of the Agent
  - Agent can be altered (data and code)
  - Agent's data (possibly sensitive) can be scanned
  - Agent can be simply destroyed
- Protection of the Host
  - Typical access control issue, yet we want downloaded code to have very limited rights if compared to normal rights of the user who downloaded it

Access Control
Protection of the Agent

- Attack types

Leaking of code, data, control flow.	Denial of execution
Manipulation of code, data, control flow.	Leaking and manipulation of the interaction with other agents.
Incorrect execution of code.	Returning wrong results of system calls issued by the agent.
Masquerading of the host.	Tampering of agent itineraries

- Complete protection for the agent is not possible
  - at least until TCPA-Palladium or some other Big-Brother technique is massively deployed (so we wrote some years ago ☺)
- What can be done is detecting modification to an agent code or data: in Ajanta we have
  - Read only state: signed by the owner with private key
  - Targeted state: encrypted with a key not shipped with the agent
  - Append only logs

Access Control

- Protection of the Agent
  - Append only logs: data can only be appended, not removed or modified
    - Initially the log contains only the initial checksum C (stored at the creator of the agent)
      - $C=K^{+}(N)$ where N is a nonce
    - Addition of X at a server S: append X, and sign(X,S)
    - The new checksum C_new= K^{+}(C_old,sig(X,S),S)

Access Control

- Protection of the Host
  - Sandbox: each instruction can be fully controlled
    - Compiled code
      - Identify dangerous operation and shell them with code that check conditions at runtime
    - Interpreted code
      - Easier: the interpreter can control the security of each instruction execution
  - Java approach to sandbox
    - Trusted Class Loader (applets can’t change it)
    - Byte code verifier (check for malformed code)
    - Security manager (a reference monitor)

Access Control

- Protection of the Host
  - Playground (for increased security)
    - Downloaded code is diverted to a dedicated machine
    - Increased security: we have 3 level of protection
      - Sandbox
      - Playground machine OS
      - Limited connectivity from playground to other internal machines
  - Signed code
    - Security Manager can have several security policies for authenticated code

Access Control

- Protection of the Host
  - Advanced Java techniques for security:
    - Object reference as capabilities
      - We can leverage from Java strong type system: it's not possible to forge object references
      - If the application at loading time is not given a reference to e.g. a file factory the application can't use the file system
      - Requires a redesign of the API to permit reference navigation only in safe ways
        - Java-E language
    - Name space management
      - Import statement can be resolved, based on different policy settings to different packages: import java.io.File could be diverted for untrusted apps to security.File

Access Control - Java

- The access control architecture in Java platform protects access to sensitive resources (i.e. local files..) or sensitive application code
- All access control decisions are mediated by a security manager
  - Represented by the java.lang.SecurityManager class
- A security manager must be installed into the Java runtime in order to activate the access control checks
  - Java applets and web application are automatically run with a security manager installed
  - To run a local application with a security manger
    - Use the System.setSecurityManager() method or ...
    - ... Invoke java with –Djava.security.manager argument on the commandline

Access Control - Java

- Before Java 2 (JDK 1.2) the only way to specify access control policies was to write an ad-hoc security manager object
- Local applications were always considered as trusted ones
- Since Java 2 access control architecture has changed:
  - Code from different sources can be given different sets of permissions
    - Fine-grained levels of trust
  - Access control policies can be configured in different ways
    - Es. By editing a text file
  - Also locally installed code can be treated as untrusted or partially untrusted
    - Only system classes and standard extensions are considered as fully trusted
  - Policies determine Permission objects
    - Permission subclasses can be defined to govern access to new system resources

Access Control - Java

- How sandbox works (Java 1.0 – 1.1)
- Suppose that an applet attempts to read /etc/passwd
- First it must use a FileInputStream object passing the file name to the constructor
- FileInputStream can be a dangerous class
  - Its programmers designed its constructor to use a SecurityManager class
  - The constructor calls SecurityManager.checkRead()
  - CheckRead has no return value
    - It either returns normally or throws a SecurityException

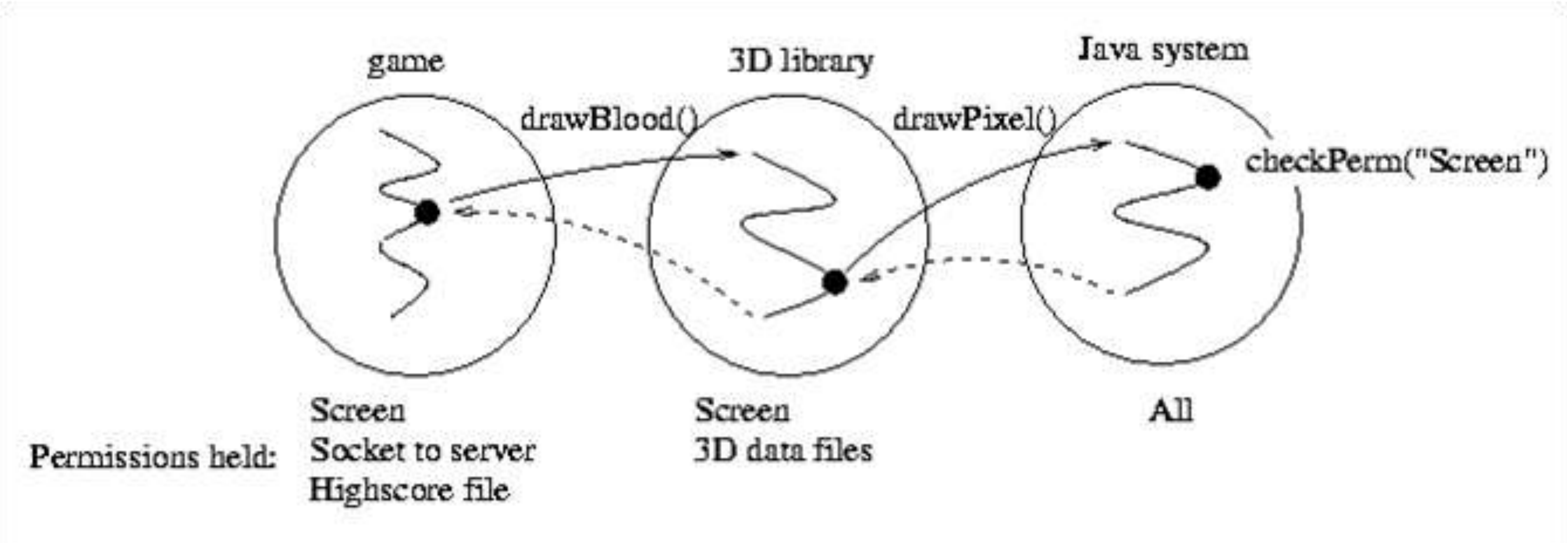
Access Control - Java

- How sandbox works (Java 2)
- What's new in Java 2 is how the Security Manager methods work
- In our example we described the checkRead() method
  - It creates a FilePermission
    - With target "/etc/passwd" and action "read"
  - It passes the Permission to the static CheckPermission() method of the AccessController class
  - One of the main features of a Permission subclass like FilePermission is the definition of the implies() method
    - In the case of files it can be used to determine whether permission to read "/etc/*" implies permission to read "/etc/passwd"

Access control - Java

- So, since Java 2 we can use policy file to tune rights given to a specific application
- The class loader automatically associates the following information with the code:
  - Where the code was loaded from
  - Who signed the code (if anyone)
  - Default permissions granted to code
    - Only a limited set, all the others are determined by the access control schema seen above
- Depending on where the class was downloaded from the Security manager applies a different policy

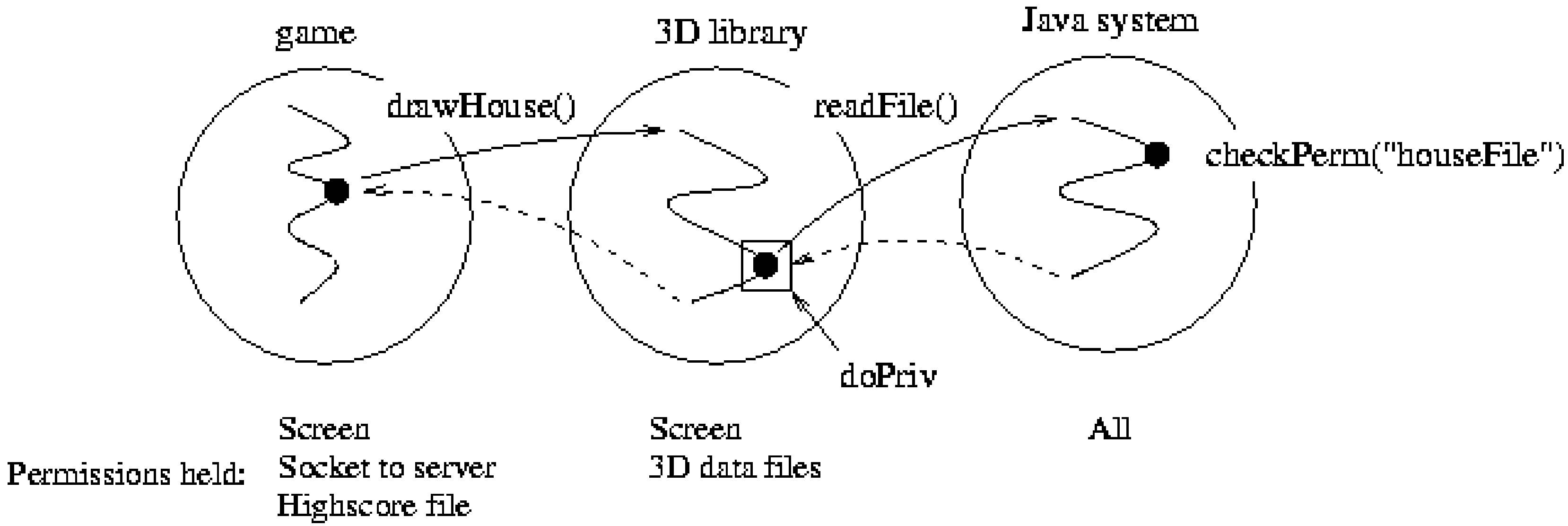
AC – Protection of the Host



- (Extended) Stack introspection
  - Each codebase can have its own security policy
  - What happens when we have multiple procedures on the stack with different policies?
  - Default behavior: check if the call is permitted in every method of the stack

AC – Protection of the Host

- Now 3D library wants to access 3D data files…



game

3D library

Java system

drawHouse()

readFile()

checkPerm("houseFile")

doPriv

Permissions held:

Screen
Socket to server
Highscore file

Screen
3D data files

All

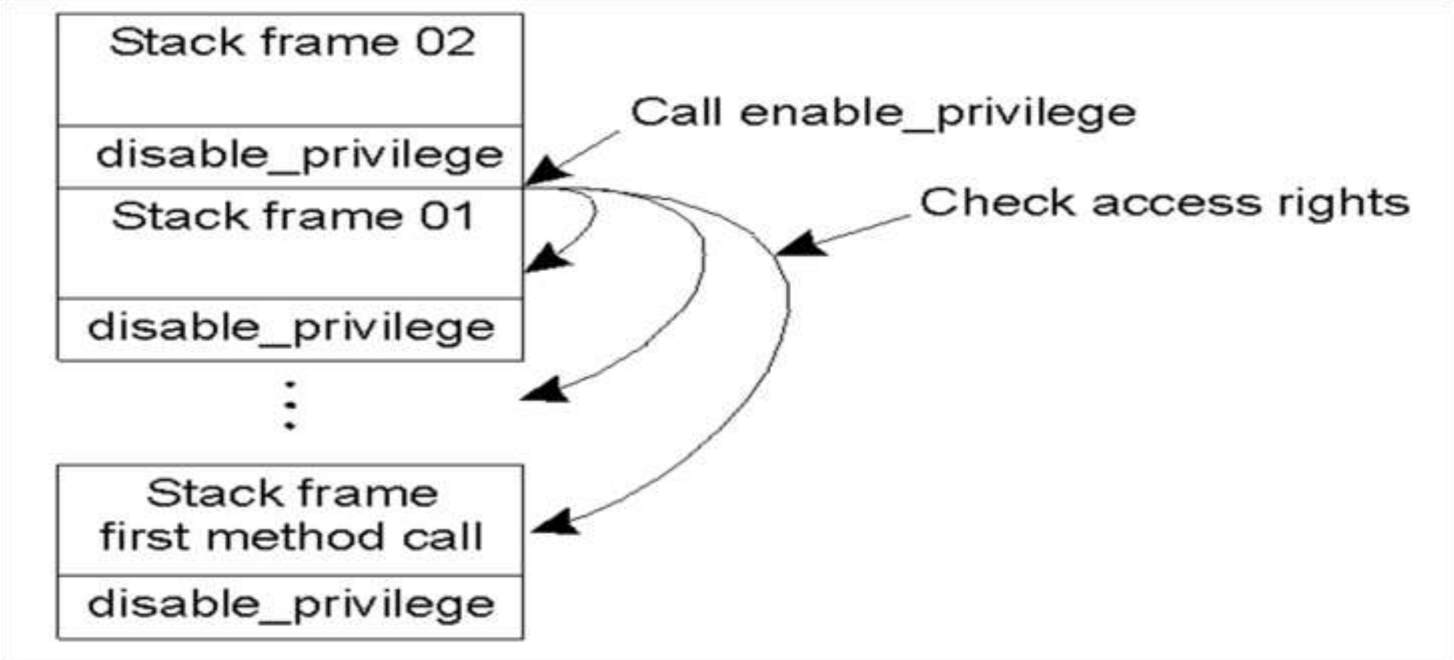
```
Object house = AccessController.doPrivileged(
  new PrivilegedAction() {
    public Object run() {
      return readFile("house");
    }
  });
```

AC - Protection of the Host

- (Extended) Stack introspection algorithm

```
Boolean checkPermission(Permission
  toCheck){
  Stack callStack = getCallStack();
  while( ! callStack.empty() ) {
    StackFrame here = callStack.pop();
    Domain thisDomain = here.getDomain();
    if(!thisDomain.implies(toCheck)
  )return false;
    if( here.isPrivileged() ) return true;
  }
  return true;
}
```

AC - Protection of the Host



- The old API (pre Java 1.2) used two construct to enter and leave a privileged section
  - the book talks about these old API

AC - Firewalls

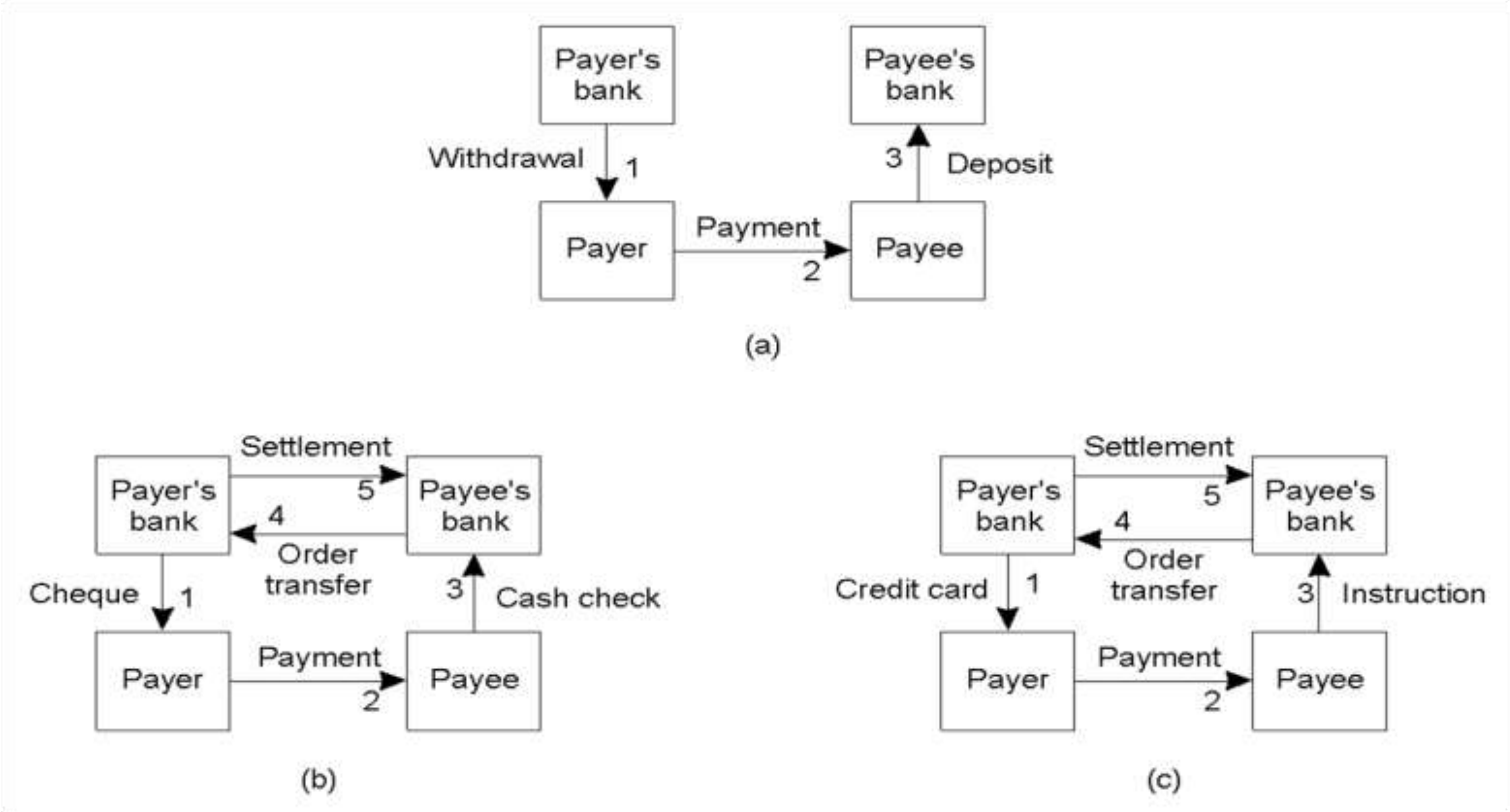
- So far we have seen approaches that work fine as long as all communicating parties play according to the same set of rules.
- This happens when the distributes system is isolated from the rest of the world.
- To protect resources in an open world, a different approach is needed:
  - Any external access is controlled by a special kind of reference monitor known as Firewall

AC - Firewalls

- Firewalls essentially come in two different flavours:
  - Packet filtering gateway: this type of firewall operates as a router and makes decision as to whether or not to pass a network packet based only on packet header's info (source and destination address / port)
  - Application level gateway: this type of firewall actually inspects the content of a message (example: discard incoming mail exceeding a certain size)

Electronic payment

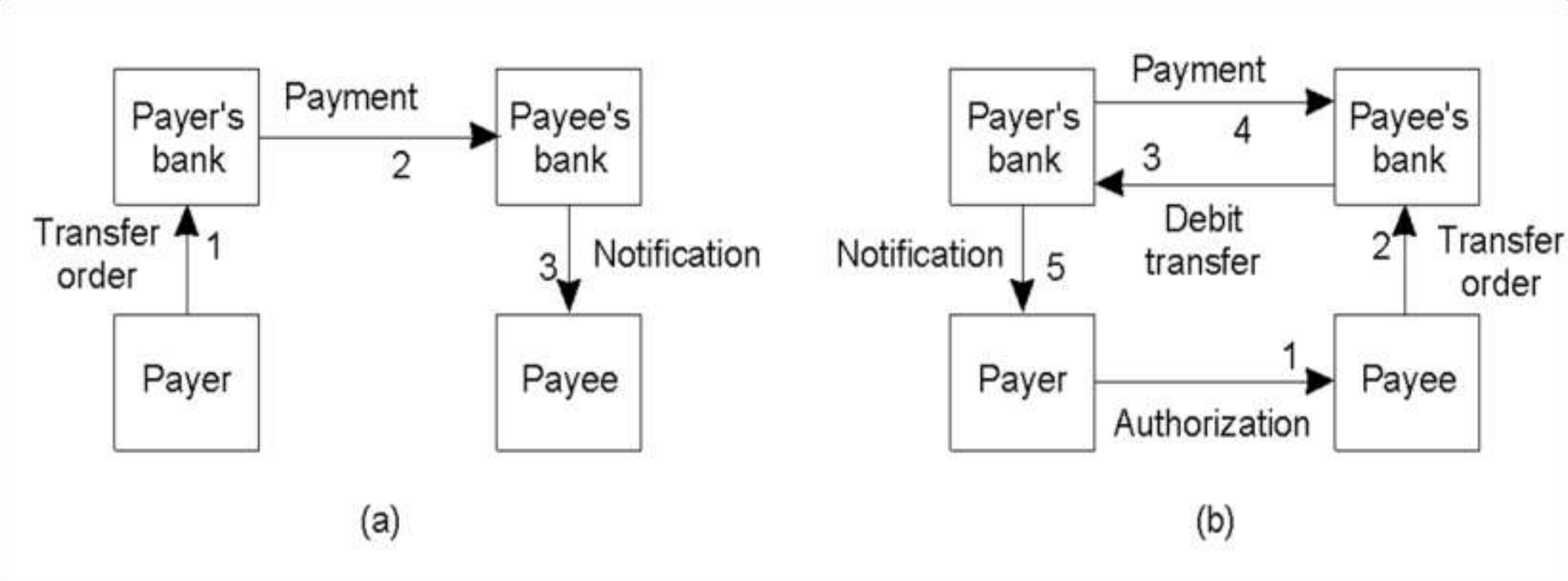
Electronic Payment



a) cash b) check c) credit card

- Traditional payment methods that require co-location of customer and merchant

Electronic Payment



a) money order b) debit order

- Other methods are used otherwise

Electronic Payment

- There is a privacy issue

CASH	Merchant	Customer	Date	Amount	Item
Merchant	Full	Partial	Full	Full	Full
Customer	Full	Full	Full	Full	Full
Bank	None	None	None	None	None
Observer	Full	Partial	Full	Full	Full

CREDIT CARD	Merchant	Customer	Date	Amount	Item
Merchant	Full	Full	Full	Full	Full
Customer	Full	Full	Full	Full	Full
Bank	Full	Full	Full	Full	None
Observer	Full	Partial	Full	Full	Full

Electronic Payment

- Two examples
  - E-cash
    - Provide a service similar to real cash (anonymity)
  - SET
    - Secure credit card based service
- E-cash basics
  - Alice “withdraws” cash notes signed by the bank
  - Bob can verify authenticity with bank signature
  - Bob contact the bank to protect from double spending
  - No privacy: the banks can match serial numbers

Electronic Payment

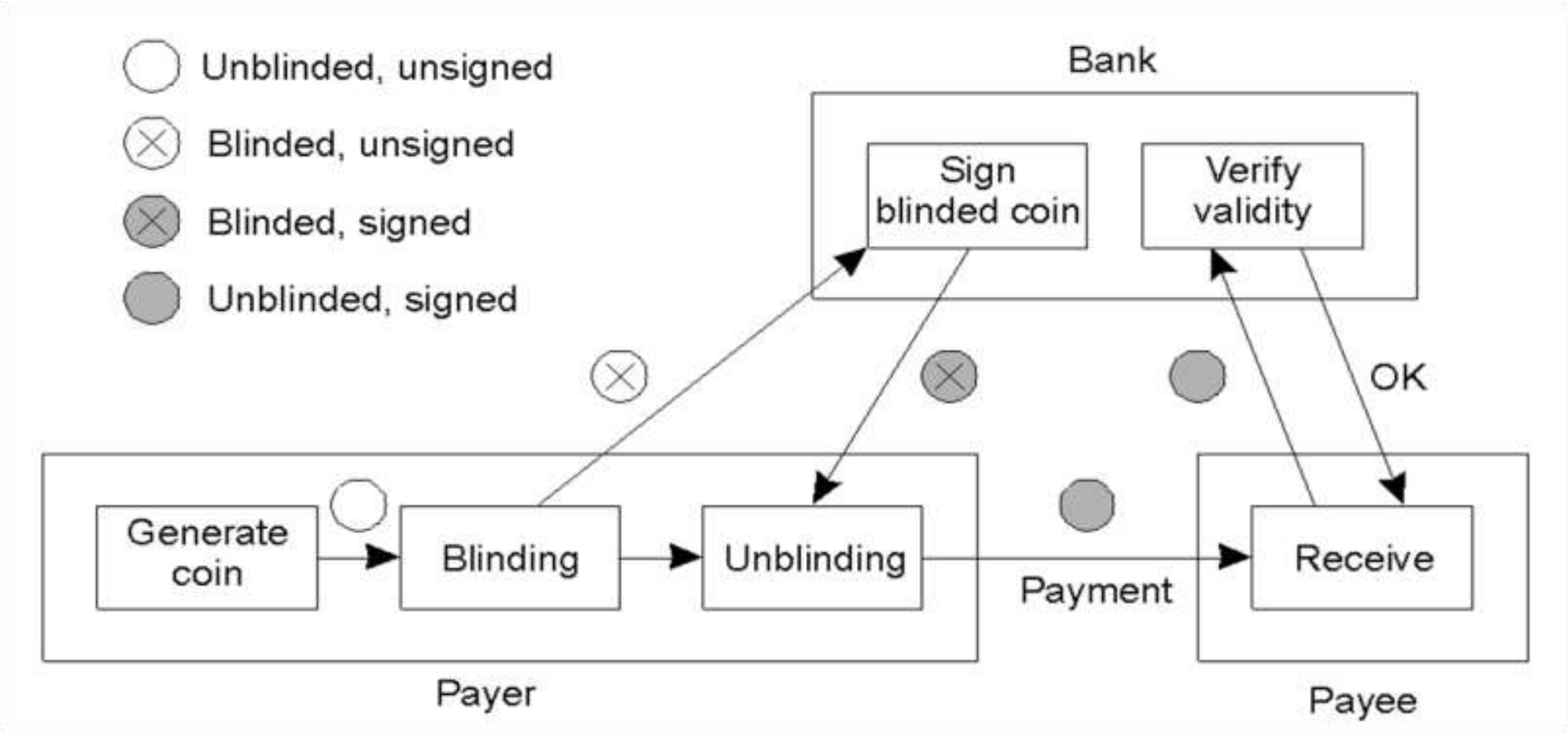
- E-Cash
  - Blind signature
    - Alice put something in an envelope whose inside is covered with carbon paper, the bank signs it
    - The bank doesn't know what's inside but can't deny having signed it
      - Since the bank doesn't know what's inside the envelope, the amount can't be on the note. It should be in the signature. That's why the bank uses a different keys for different values (K-10 for 10€, K-1 for 1€ etc.)
    - What's inside the envelope?
      - "this is my serial number: ……………."

Electronic Payment

- Blind Signature
  - With RSA
    - e bank’s public key
    - d bank’s private key
    - Alice builds
      - $t = mk^e \bmod n$
    - The bank signs it
      - t^d
    - Alice unblinds it
      - $s = t^d/k \bmod n = m^d k^{ed}/k \bmod n = m^d \bmod n$
    - That’s it: the bank signed m without knowing m

Politecnico di Milano

Electronic Payment



E-Cash

Electronic Payment

- E-Cash
  - There are a number of properties that should be useful in a digital cash system
    1. Independence (from the physical world)
    2. Security (cannot be copied and reused)
    3. Privacy
    4. Off-line payment
    5. Transferability
    6. Divisibility
  - Our protocol does not satisfy 4 and 6. There are other protocols that do satisfy all of them

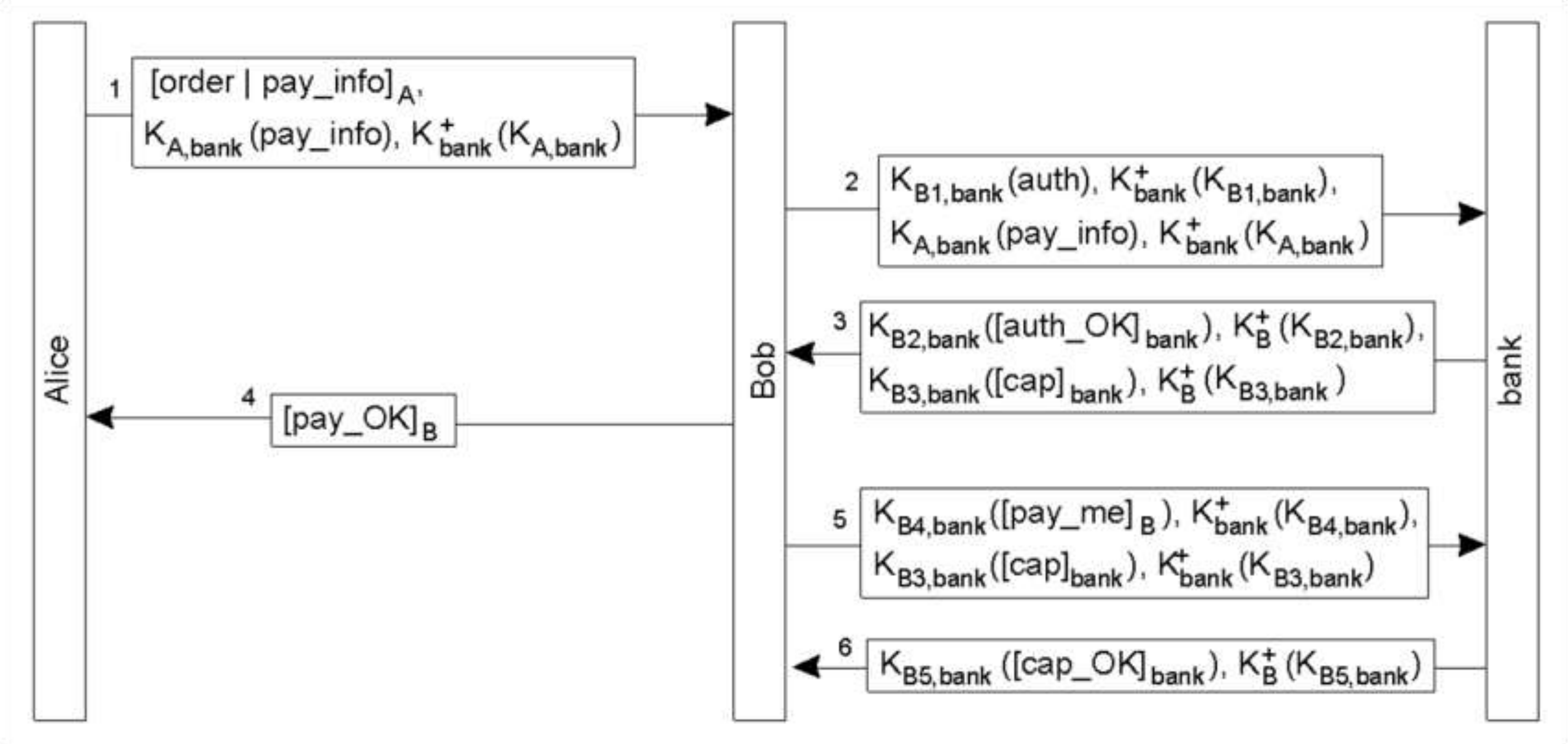
Electronic Payment

- SET
  - We saw that credit card payments leave many e-trails.
  - Using credit card on the internet requires that the merchant does not know payment details (e.g. credit card number) and the bank does not know order details.
  - How to support credit card payments?

Electronic Payment

- SET
  - Dual signature
  - Alice wants to send a message for approval to both Bank and Merchant, disclosing
    - the first part (m1) to Bob only (the merchant)
    - the second part (m2) to Charlie only (the bank)
  - $[m1|m2]_A = [m1,H(m2),[H(m1),H(m2)]_A]$
  - $[m2|m1]_A = [m2,H(m1),[H(m1),H(m2)]_A]$
  - Each approver verifies his own part and checks that it matches (with the hash of the other) the dual signature

Electronic Payment


Alice
Bob
bank
1
$[\text{order} \mid \text{pay\_info}]_A$,
$K_{A,bank}(\text{pay\_info})$, $K^+_{bank}(K_{A,bank})$
2
$K_{B1,bank}(\text{auth})$, $K^+_{bank}(K_{B1,bank})$,
$K_{A,bank}(\text{pay\_info})$, $K^+_{bank}(K_{A,bank})$
3
$K_{B2,bank}([\text{auth\_OK}]_{bank})$, $K^+_B(K_{B2,bank})$,
$K_{B3,bank}([\text{cap}]_{bank})$, $K^+_B(K_{B3,bank})$
4
$[\text{pay\_OK}]_B$
5
$K_{B4,bank}([\text{pay\_me}]_B)$, $K^+_{bank}(K_{B4,bank})$,
$K_{B3,bank}([\text{cap}]_{bank})$, $K^+_{bank}(K_{B3,bank})$
6
$K_{B5,bank}([\text{cap\_OK}]_{bank})$, $K^+_B(K_{B5,bank})$

SET