Distributed Systems
Principles and Paradigms

Maarten van Steen

VU Amsterdam, Dept. Computer Science
Room R4.20, steen@cs.vu.nl

Chapter 09: Security

Version: November 30, 2011

vrije Universiteit amsterdam

Contents

Overview

- Introduction
- Secure channels
- Access control
- Security management

Security: Dependability revisited

Basics

A *component* provides *services* to *clients*. To provide services, the component may require the services from other components $\Rightarrow$ a component may depend on some other component.

Property	Description
Availability	Accessible and usable upon demand for authorized entities
Reliability	Continuity of service delivery
Safety	Very low probability of catastrophes
Confidentiality	No unauthorized disclosure of information
Integrity	No accidental or malicious alterations of information have been performed (even by authorized entities)

Security: Dependability revisited

Observation

In distributed systems, security is the combination of availability, integrity, and confidentiality. A dependable distributed system is thus fault tolerant and secure.

Security threats

The players

- Subject: Entity capable of issuing a request for a service as provided by objects
- Channel: The carrier of requests and replies for services offered to subjects
- Object: Entity providing services to subjects.

Security threats

The threats

Threat	Channel	Object
Interruption	Preventing message transfer	Denial of service
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

Issue

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 modified (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.

Security policies

Policy

Prescribes how to use mechanisms to protect against attacks. Requires that a model of possible attacks is described (i.e., security architecture).

Example: Globus security architecture

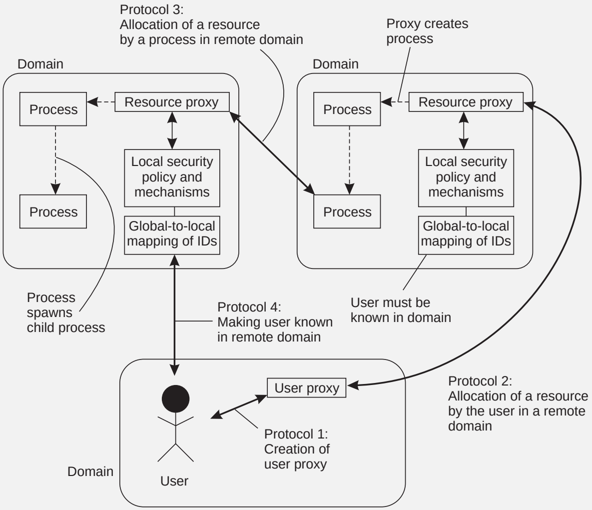
- There are multiple administrative domains
- Local operations subject to local security policies
- Global operations require requester to be globally known
- Interdomain operations require mutual authentication
- Global authentication replaces local authentication
- Users can delegate privileges to processes
- Credentials can be shared between processes in the same domain

Security policies

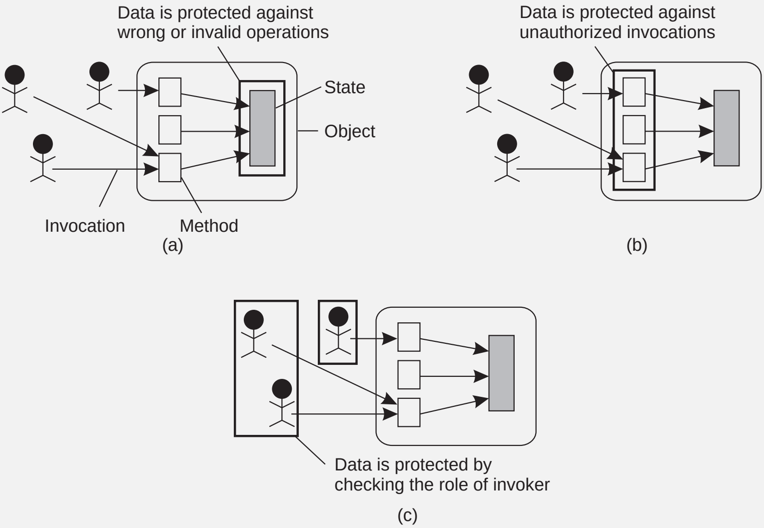
Globus

Policy statements leads to the introduction of mechanisms for cross-domain authentication and making users globally known $\Rightarrow$ user proxies and resource proxies

Security policies: Globus



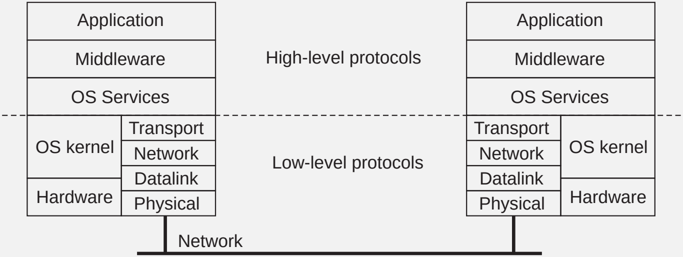
Design issue: Focus of control



Design issue: Layering of mechanisms and TCB

Issue

At which logical level are we going to implement security mechanisms?



Design issue: Layering of mechanisms and TCB

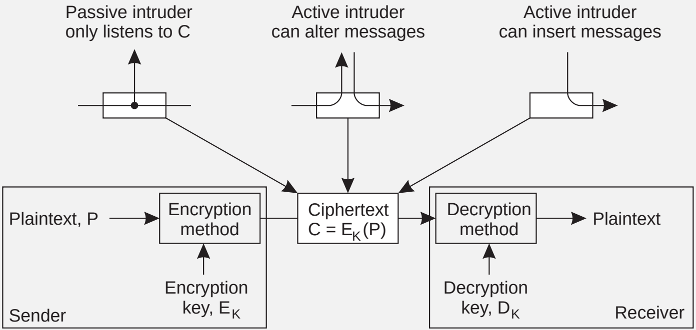
Important

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

Trusted Computing Base

What is the set of mechanisms needed to enforce a policy. The smaller, the better.

Cryptography



Symmetric system: Use a single key to (1) encrypt and (2) decrypt. Requires that sender and receiver share the secret key.

Asymmetric system: Use different keys for encryption and decryption, of which one is private, and the other public.

Hashing system: Only encrypt data and produce a fixed-length digest. There is no decryption; only comparison is possible.

Cryptographic functions

Essence

Make the encryption method E public, but let the encryption as a whole be parameterized by means of a key S (Same for decryption)

- One-way function: Given some output m_{out} of E_S, it is (analytically or) computationally infeasible to find $m_{in} : E_S(m_{in}) = m_{out}$
- Weak collision resistance: Given the pair $\langle m, E_S(m) \rangle$, it is computationally infeasible to find an $m^* \neq m$ such that $E_S(m^*) = E_S(m)$
- Strong collision resistance: It is computationally infeasible to find any two different inputs m^* and m such that $E_S(m^*) = E_S(m)$

Cryptographic functions

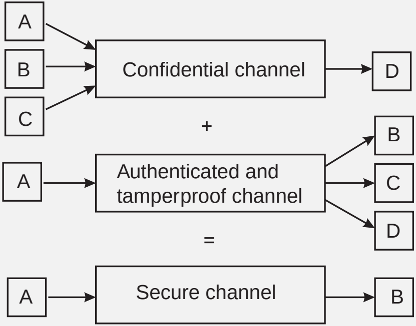
Essence (cnt'd)

- One-way key: Given an encrypted message m_{out}, message m_{in}, and encryption function E, it is analytically and computationally infeasible to find a key K such that $m_{out} = E_K(m_{in})$
- Weak key collision resistance: Given a triplet $\langle m, K, E \rangle$, it is computationally infeasible to find an $K^* \neq K$ such that $E_{K^*}(m) = E_K(m)$
- Strong key 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

- Authentication
- Message Integrity and confidentiality
- Secure group communication

Secure channels



What's a secure channel

- Both parties know who is on the other side (authenticated).
- Both parties know that messages cannot be tampered with (integrity).
- Both parties know messages cannot leak away (confidentiality).

Authentication versus integrity

Important

Authentication and data integrity rely on each other: Consider an active attack by Trudy on the communication from Alice to Bob.

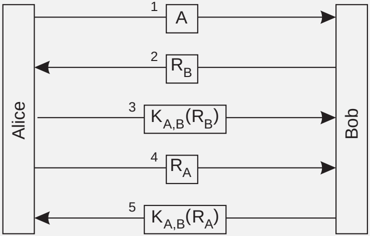
Authentication without integrity

Alice's message is authenticated, and intercepted by Trudy, who tampers with its content, but leaves the authentication part as is. Authentication has become meaningless.

Integrity without authentication

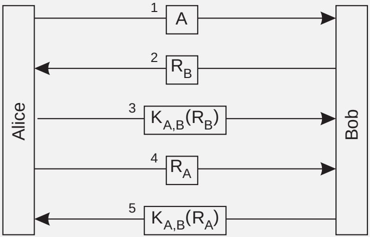
Trudy intercepts a message from Alice, and then makes Bob believe that the content was really sent by Trudy. Integrity has become meaningless.

Authentication: Secret keys



1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret keys



1: Alice sends ID to Bob

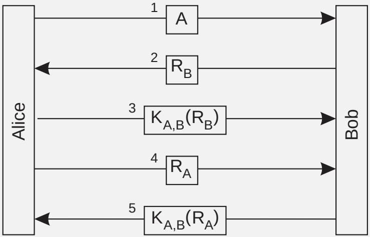
2: Bob sends challenge R_B to Alice

3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.

4: Alice sends challenge R_A to Bob

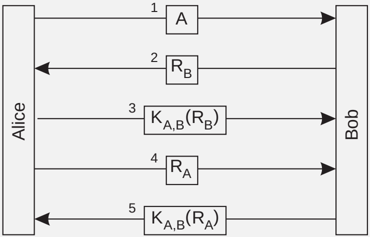
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret keys



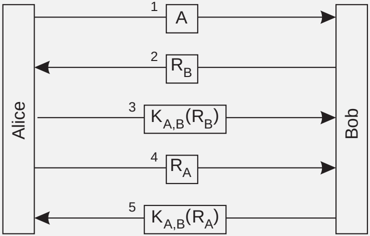
1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret keys



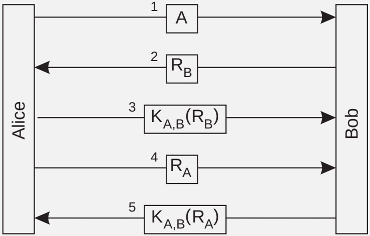
1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret keys



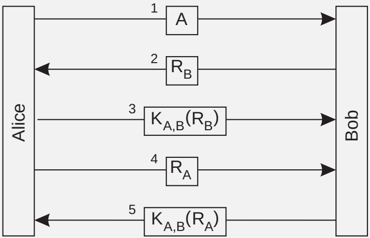
1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret keys



1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

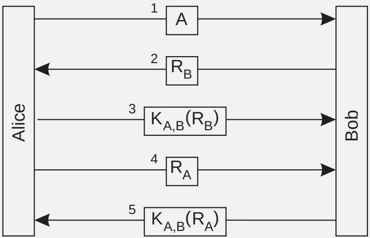
Authentication: Secret keys



Improvement

Combine steps 1&4, and 2&5. Price to pay: correctness.

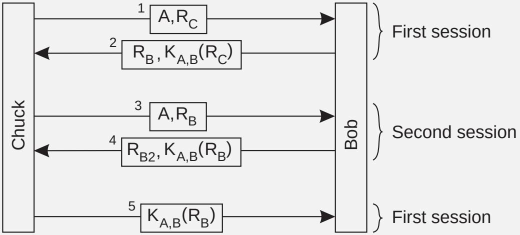
Authentication: Secret keys



Improvement

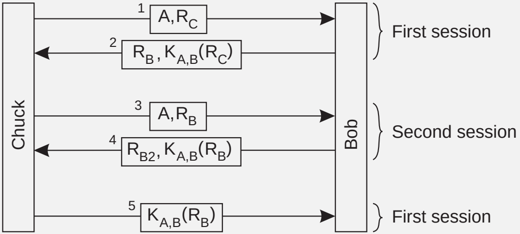
Combine steps 1&4, and 2&5. Price to pay: correctness.

Authentication: Secret keys reflection attack



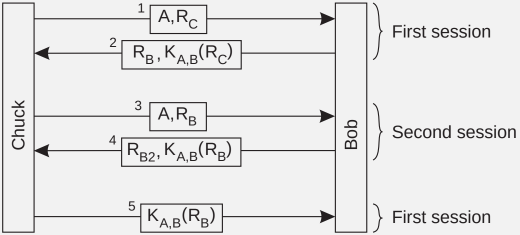
1: Chuck claims he's Alice, and sends challenge R_C
2: Bob returns a challenge R_B and the encrypted R_C
3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B
4: Bob sends back a challenge, plus $K_{A,B}(R_B)$
5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove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

Authentication: 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: Secret keys reflection attack



1: Chuck claims he's Alice, and sends challenge R_C

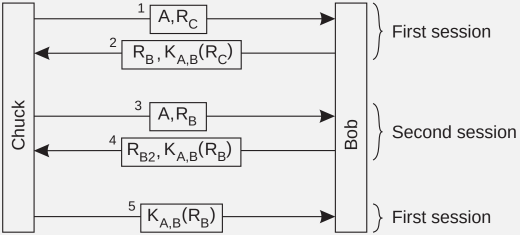
2: Bob returns a challenge R_B and the encrypted R_C

3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B

4: Bob sends back a challenge, plus $K_{A,B}(R_B)$

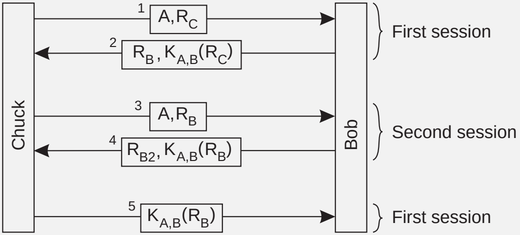
5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice.

Authentication: Secret keys reflection attack



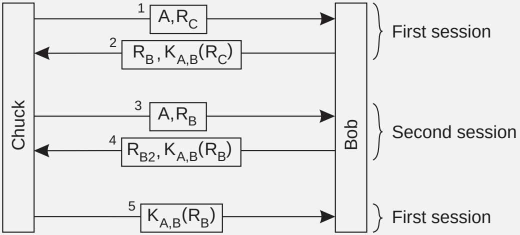
1: Chuck claims he's Alice, and sends challenge R_C
2: Bob returns a challenge R_B and the encrypted R_C
3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B
4: Bob sends back a challenge, plus $K_{A,B}(R_B)$
5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove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

Authentication: Secret keys reflection attack



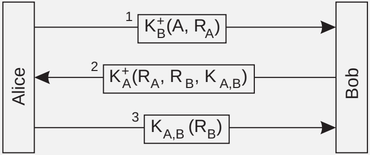
1: Chuck claims he's Alice, and sends challenge R_C
2: Bob returns a challenge R_B and the encrypted R_C
3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B
4: Bob sends back a challenge, plus $K_{A,B}(R_B)$
5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove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

Authentication: 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: Public key

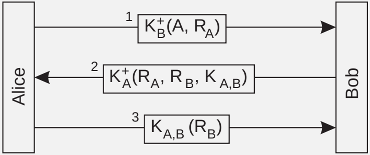


1: Alice sends a challenge R_A to Bob, encrypted with Bob's public key K_B^+.

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 $K_{A,B}$

Authentication: Public key

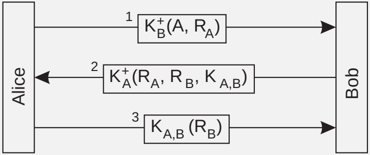


1: Alice sends a challenge R_A to Bob, encrypted with Bob's public key K_B^+.

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^+.

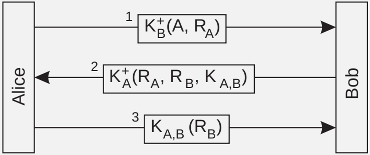
3: Alice proves she's Alice by sending back the decrypted challenge, encrypted with generated secret key $K_{A,B}$

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 $K_{A,B}$

Authentication: Public key

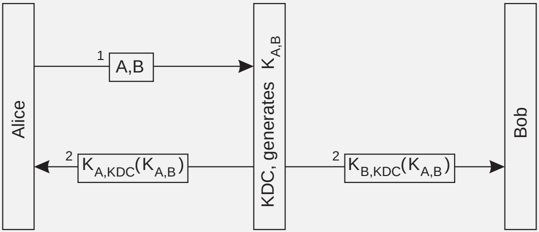


1: Alice sends a challenge R_A to Bob, encrypted with Bob's public key K_B^+.
2: Bob decrypts the message, generates a secret key $K_{A,B}$ (session key), proves he's Bob (by sending R_A back), and sends a challenge R_B to Alice. Everything's encrypted with Alice's public key K_A^+.
3: Alice proves she's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

Authentication: KDC

Problem

With N subjects, we need to manage $N(N-1)/2$ keys, each subject knowing $N-1$ keys $\Rightarrow$ use a trusted Key Distribution Center that generates keys when necessary.



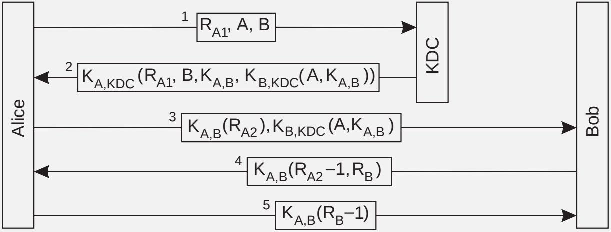
Question

How many keys do we need to manage?

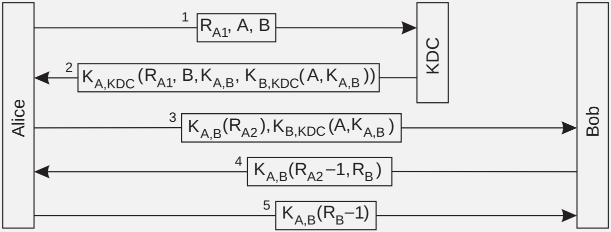
Authentication: KDC (Needham-Schroeder)

Inconvenient

We need to ensure that Bob knows about $K_{A,B}$ before Alice gets in touch ⇒ let Alice do the work and pass her a ticket to set up a secure channel with Bob.



Needham-Schroeder: Subtleties



Some issues

Q1: Why does the KDC put *B*ob into its reply message, and *A*lice into the ticket?

Q2: The ticket sent back to Alice by the KDC is encrypted with Alice's key. Is this necessary?

Needham-Schroeder: Subtleties

Security flaw

Suppose Trudy finds out Alice's key $\Rightarrow$ she can use that key anytime to impersonate Alice, even if Alice changes her private key at the KDC.

Reasoning

Once Trudy finds out Alice's key, she can use it to decrypt a (possibly old) ticket for a session with Bob, and convince Bob to talk to her using the old session key.

Solution

Have Alice get an encrypted number from Bob first, and put that number in the ticket provided by the KDC $\Rightarrow$ we're now ensuring that every session is known at the KDC.

Confidentiality

Solutions

Secret key: Use a shared secret key to encrypt and decrypt all messages sent between Alice and Bob

Public key: If Alice sends a message m to Bob, she encrypts it with Bob's public key: $K_B^+(m)$

Problems with keys

- Keys wear out: The more data is encrypted by a single key, the easier it becomes to find that key $\Rightarrow$ don't use keys too often
- Danger of replay: Using the same key for different communication sessions, permits old messages to be inserted in the current session $\Rightarrow$ don't use keys for different sessions

Confidentiality

Problems with keys

- Compromised keys: If a key is compromised, you can never use it again. Really bad if *all* communication between Alice and Bob is based on the same key over and over again $\Rightarrow$ don't use the same key for different things.
- Temporary keys: Untrusted components may play along perhaps just once, but you would never want them to have knowledge about your really good key for all times $\Rightarrow$ make keys disposable

Confidentiality

Essence

Don’t use valuable and expensive keys for all communication, but only for authentication purposes.

Consequnce

Introduce a “cheap” session key that is used only during one single conversation or connection (“cheap” also means efficient in encryption and decryption).

Digital signatures

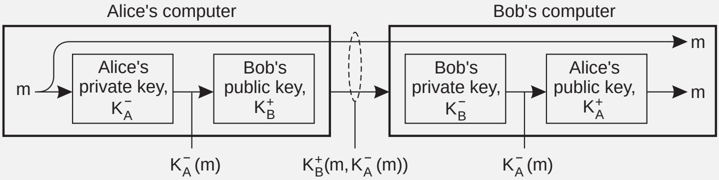
Harder requirements

- Authentication: Receiver can verify the claimed identity of the sender
- Nonrepudiation: The sender can later not deny that he/she sent the message
- Integrity: The 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



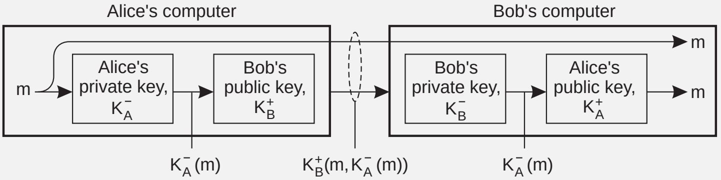
1: Alice encrypts her message m with her private key $K_A^- \Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



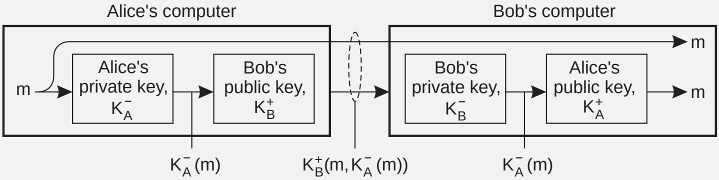
1: Alice encrypts her message m with her private key $K_A^- \Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



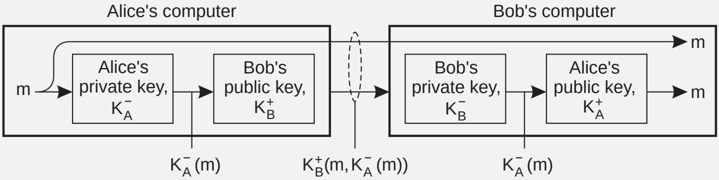
1: Alice encrypts her message m with her private key $K_A^- \Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



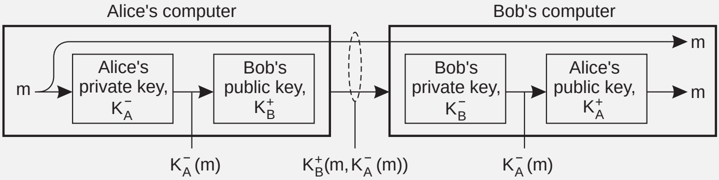
1: Alice encrypts her message m with her private key $K_A^- \Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



1: Alice encrypts her message m with her private key $K_A^- \Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

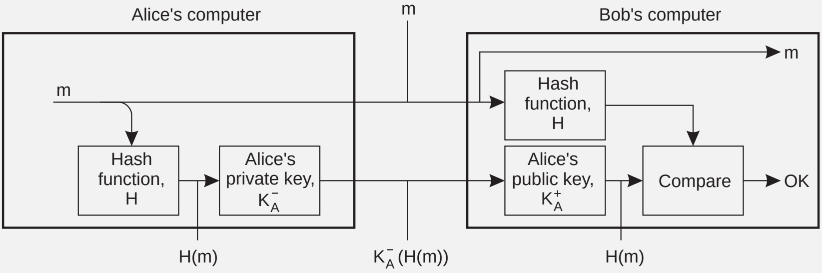
3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Message digests

Basic idea

Don't mix authentication and secrecy. Instead, it should also be possible to send a message in the clear, but have it signed as well ⇒ take a message digest, and sign that.



Secure group communication

Design issue

How can you share secret information between multiple members without losing everything when one member turns bad.

Confidentiality

Follow a simple (hard-to-scale) approach by maintaining a separate secret key between each pair of members.

Secure group communication

Replication

You also want to provide replication transparency. Apply secret sharing:

- No process knows the entire secret; it can be revealed only through joint cooperation
- Assumption: at most k out of N processes can produce an incorrect answer
- At most $c \leq k$ processes have been corrupted

Note

We are dealing with a k fault tolerant process group.

Secure group communication

Replication

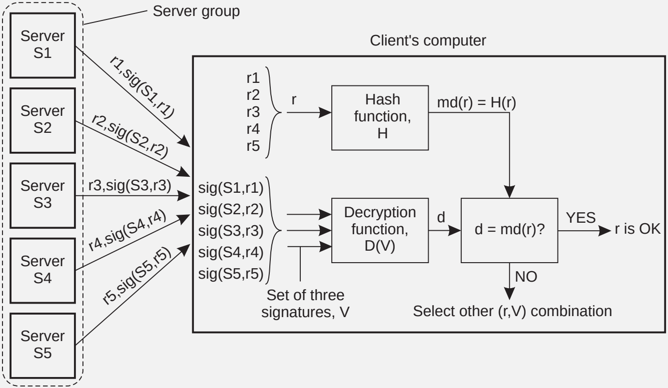
You also want to provide replication transparency. Apply secret sharing:

- No process knows the entire secret; it can be revealed only through joint cooperation
- Assumption: at most k out of N processes can produce an incorrect answer
- At most $c \leq k$ processes have been corrupted

Note

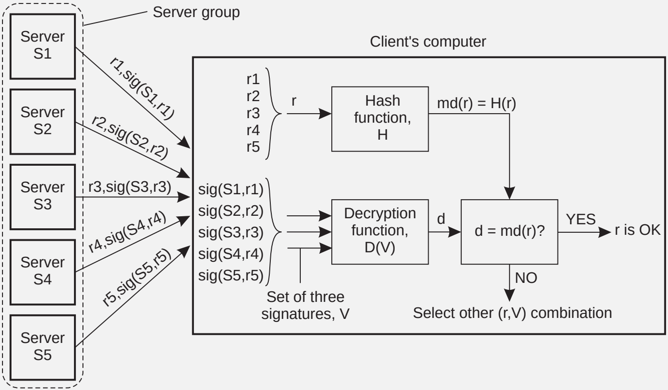
We are dealing with a k fault tolerant process group.

Secure replicated group



- N=5, c=2
- Each server S_i sees each request and responds with r_i
- r_i is sent with digest $md(r_i)$, and signed with private key K_i^-.

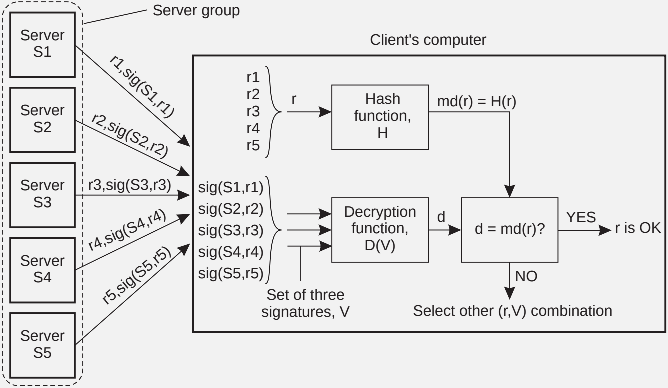
Secure replicated group



- Client uses special decryption function *D* that computes a single digest *d* from *three* signatures:

$$d = D(sig(S, r), sig(S', r'), sig(S'', r''))$$

Secure replicated group



- If $d = md(r_i)$ for some r_i, r_i is considered correct
- Also known as (m,n)-threshold scheme (with $m = c + 1, n = N$)

Access control

- General issues
- Firewalls
- Secure mobile code

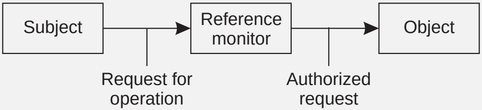
Authorization versus authentication

Definition

- 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.

Note

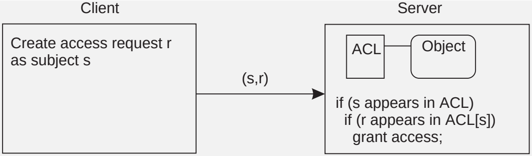
Authorization makes sense only if the requesting subject has been authenticated



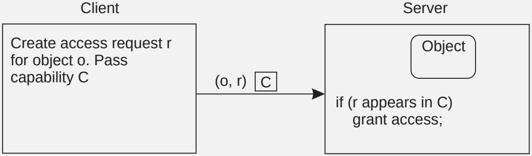
Access Control Matrix (ACM)

Essence

Maintain an access control matrix *ACM* in which entry *ACM[S,O]* contains the permissible operations that subject *S* can perform on object *O*.

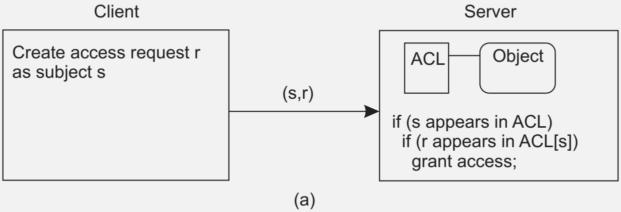


(a)



(b)

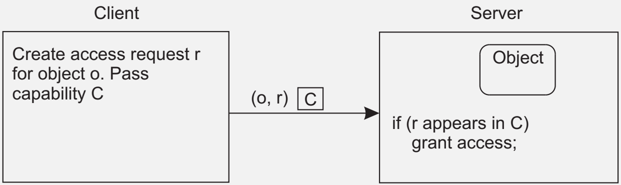
Access Control Matrix (ACM)



Access Control List (ACL)

Each object *O* maintains an access control list (ACL): *ACM[*,O]* describing the permissible operations per subject (or group of subjects).

Access Control Matrix (ACM)



(b)

Capabilities

Each subject *S* has a capability: *ACM[S,*]* describing the permissible operations per object (or category of objects).

Protection domains

Issue

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

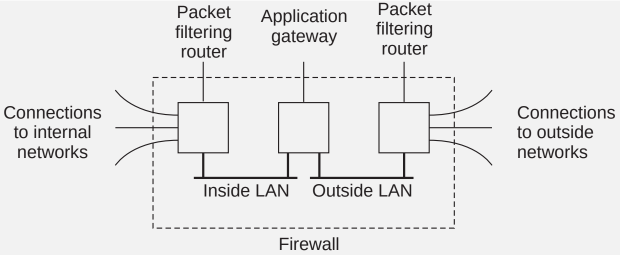
Common implementation of protection domains

- 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

Essence

Sometimes it's better to select service requests at the lowest level: network packets. Packets that do not fit certain requirements are simply removed from the channel ⇒ protect by a firewall: it implements access control.



Question

What do you think would be the biggest breach in firewalls?

Secure mobile code

Problem

Mobile code is great for balancing communication and computation, but:

- it is hard to implement a general-purpose mechanism that allows different security policies for local-resource access
- we may need to protect the mobile code (e.g., agents) against malicious hosts.

Protecting an agent

Ajanta

Detect that an agent has been tampered with while it was on the move. Most important: append-only logs:

- Data can only be appended, not removed
- There is always an associated checksum. Initially, $C_{init} = K^{+}_{owner}(N)$, with N a nonce.
- Adding data X by server S:

$$C_{new} = K^{+}_{owner}(C_{old}, sig(S,X), S)$$

- Removing data from the log:

$$K^{-}_{owner}(C) \rightarrow C_{prev}, sig(S,X), S$$

allowing the owner to check integrity of X

Protecting an agent

Ajanta

Detect that an agent has been tampered with while it was on the move. Most important: append-only logs:

- Data can only be appended, not removed
- There is always an associated checksum. Initially, $C_{init} = K^{+}_{owner}(N)$, with N a nonce.
- Adding data X by server S:

$$C_{new} = K^{+}_{owner}(C_{old}, sig(S, X), S)$$

- Removing data from the log:

$$K^{-}_{owner}(C) \rightarrow C_{prev}, sig(S, X), S$$

allowing the owner to check integrity of X

Protecting an agent

Ajanta

Detect that an agent has been tampered with while it was on the move. Most important: append-only logs:

- Data can only be appended, not removed
- There is always an associated checksum. Initially, $C_{init} = K^{+}_{owner}(N)$, with N a nonce.
- Adding data X by server S:

$$C_{new} = K^{+}_{owner}(C_{old}, sig(S,X), S)$$

- Removing data from the log:

$$K^{-}_{owner}(C) \rightarrow C_{prev}, sig(S,X), S$$

allowing the owner to check integrity of X

Protecting an agent

Ajanta

Detect that an agent has been tampered with while it was on the move. Most important: append-only logs:

- Data can only be appended, not removed
- There is always an associated checksum. Initially, $C_{init} = K^{+}_{owner}(N)$, with N a nonce.
- Adding data X by server S:

$$C_{new} = K^{+}_{owner}(C_{old}, sig(S, X), S)$$

- Removing data from the log:

$$K^{-}_{owner}(C) \rightarrow C_{prev}, sig(S, X), S$$

allowing the owner to check integrity of X

Protecting an agent

Ajanta

Detect that an agent has been tampered with while it was on the move. Most important: append-only logs:

- Data can only be appended, not removed
- There is always an associated checksum. Initially, $C_{init} = K^+_{owner}(N)$, with N a nonce.
- Adding data X by server S:

$$C_{new} = K^+_{owner}(C_{old}, sig(S,X), S)$$

- Removing data from the log:

$$K^-_{owner}(C) \rightarrow C_{prev}, sig(S,X), S$$

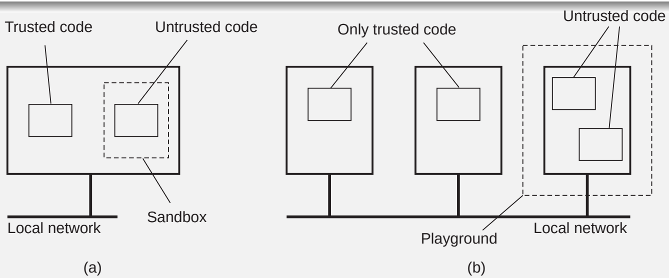
allowing the owner to check integrity of X

Protecting a host

Simple solution

Enforce a (very strict) single policy, and implement that by means of a few simple mechanisms:

- Sandbox model: Policy: Remote code is allowed access to only a pre-defined collection of resources and services. Mechanism: Check instructions for illegal memory access and service access
- Playground model: Same policy, but mechanism is to run code on separate "unprotected" machine.



Protecting a host

Observation

We need to be able to distinguish local from remote code before being able to do anything.

Refinement 1

We need to be able to assign a set of permissions to mobile code before its execution and check operations against those permissions at all times

Protecting a host

Observation

We need to be able to distinguish local from remote code before being able to do anything.

Refinement 2

We need to be able to assign different sets of permissions to different units of mobile code ⇒ authenticate mobile code (e.g. through signatures)

Question

What would be a very simple policy to follow (Microsoft's approach)?

Security management

- Key establishment and distribution
- Secure group management
- Authorization management

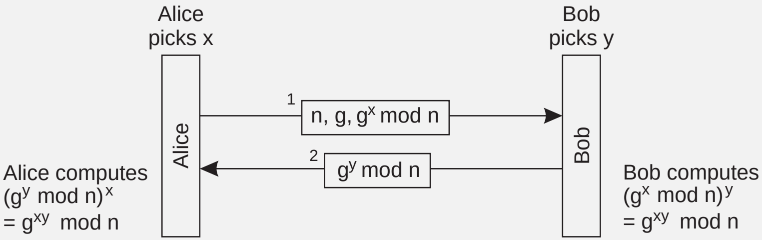
Key establishment: Diffie-Hellman

Observation

We can construct 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 (prime) and g. Both numbers may be public.
- Alice chooses large number x, and keeps it to herself. Bob does the same, say y.

Key establishment: Diffie-Hellman

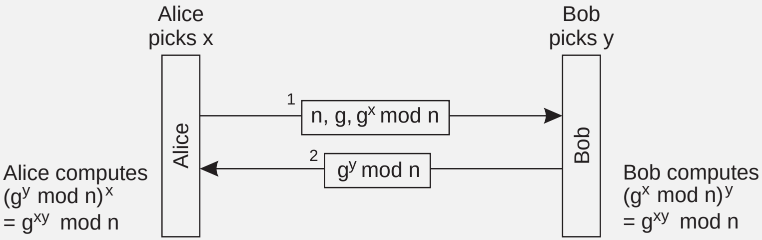


1: Alice sends $(n, g, g^x \bmod n)$ to Bob
2: Bob sends $(g^y \bmod n)$ to Alice
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime $>$ 160 bits. In practice, n, g ¿ 512 bits.

Key establishment: Diffie-Hellman

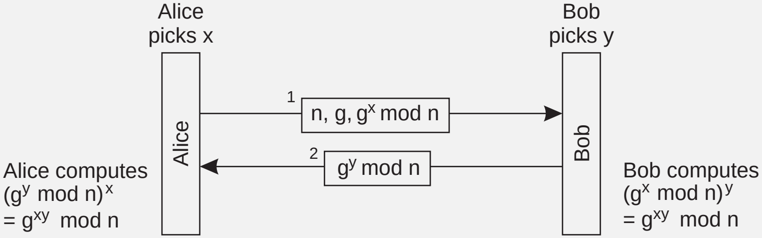


1: Alice sends $(n, g, g^x \bmod n)$ to Bob
2: Bob sends $(g^y \bmod n)$ to Alice
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime $>$ 160 bits. In practice, n, g ¿ 512 bits.

Key establishment: Diffie-Hellman

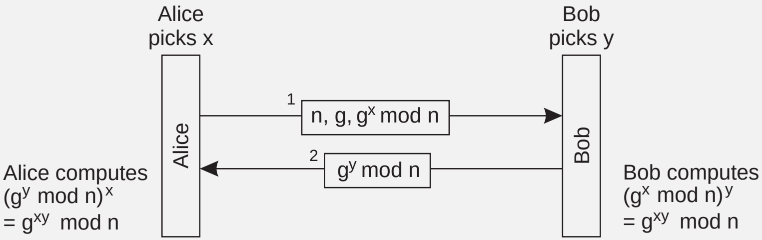


1: Alice sends $(n, g, g^x \bmod n)$ to Bob
2: Bob sends $(g^y \bmod n)$ to Alice
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime $>$ 160 bits. In practice, n, g ¿ 512 bits.

Key establishment: Diffie-Hellman

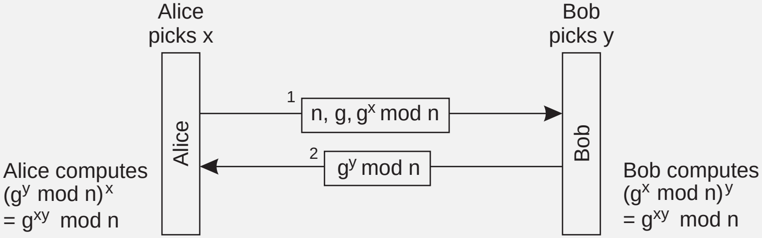


1: Alice sends $(n, g, g^x \bmod n)$ to Bob
2: Bob sends $(g^y \bmod n)$ to Alice
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime $>$ 160 bits. In practice, n, g ¿ 512 bits.

Key establishment: Diffie-Hellman



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

Note

$n = kq + 1$, with q being prime $>$ 160 bits. In practice, n, g ¿ 512 bits.

Key distribution

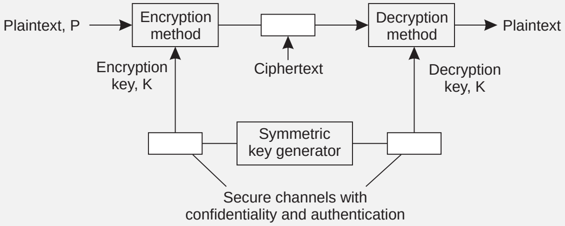
Essence

If authentication is based on cryptographic protocols, and we need session keys to establish secure channels, who's responsible for handing out keys?

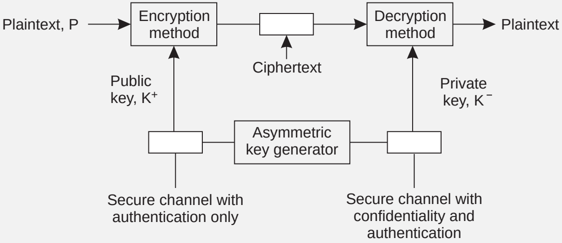
- Secret keys: Alice and Bob will have to get a shared key. They can invent their own and use it for data exchange. Alternatively, they can trust a key distribution center (KDC) and ask it for a key.
- Public keys: Alice will need Bob's public key to decrypt (signed) messages from Bob, or to send private messages to Bob. But she'll have to be sure about actually having Bob's public key, or she may be in big trouble. Use a trusted certification authority (CA) to hand out public keys.

  A public key is put in a certificate, signed by a CA.

Key distribution: getting keys to owners



(a)

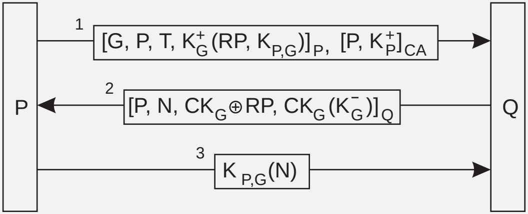


(b)

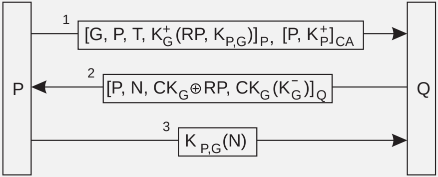
Secure group management

Organization

Group uses a key pair (K_G^+, K_G^-) for communication with nongroup members. There is a separate shared secret key CK_G for internal communication. Assume process P wants to join the group and contacts Q.



Secure group management

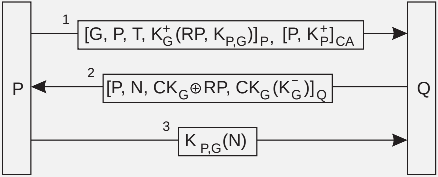


1: P generates a one-time *reply pad* RP, and a secret key $K_{P,G}$. It sends a join request to Q, signed by itself (notation: $[JR]_P$), along with a certificate containing its public key K_P^+.

2: Q authenticates P, checks whether it can be allowed as member. It returns the group key CK_G, encrypted with the one-time pad, as well as the group's private key, encrypted as $CK_G(K_G^-)$.

3: Q authenticates P and sends back $K_{P,G}(N)$ letting Q know that it has all the necessary keys.

Secure group management

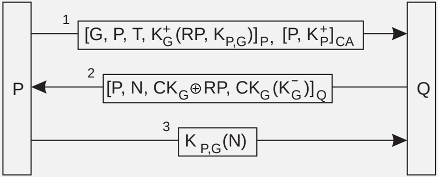


1: *P* generates a one-time *reply pad RP*, and a secret key $K_{P,G}$. It sends a join request to *Q*, signed by itself (notation: $[JR]_P$), along with a certificate containing its public key K_P^+.

2: *Q* authenticates *P*, checks whether it can be allowed as member. It returns the group key CK_G, encrypted with the one-time pad, as well as the group's private key, encrypted as $CK_G(K_G^-)$.

3: *Q* authenticates *P* and sends back $K_{P,G}(N)$ letting *Q* know that it has all the necessary keys.

Secure group management

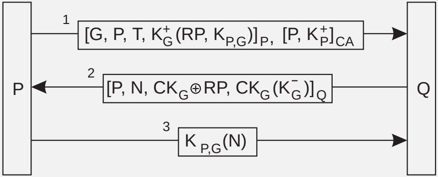


1: *P* generates a one-time *reply pad RP*, and a secret key $K_{P,G}$. It sends a join request to Q, signed by itself (notation: $[JR]_P$), along with a certificate containing its public key K_P^+.

2: *Q* authenticates *P*, checks whether it can be allowed as member. It returns the group key CK_G, encrypted with the one-time pad, as well as the group's private key, encrypted as $CK_G(K_G^-)$.

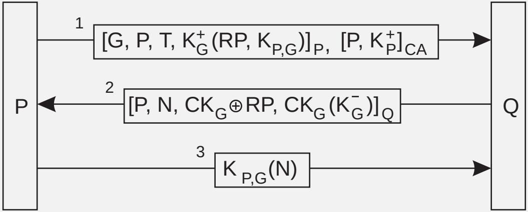
3: *Q* authenticates *P* and sends back $K_{P,G}(N)$ letting *Q* know that it has all the necessary keys.

Secure group management



1: *P* generates a one-time *reply pad RP*, and a secret key $K_{P,G}$. It sends a join request to *Q*, signed by itself (notation: $[JR]_P$), along with a certificate containing its public key K_P^+.
2: *Q* authenticates *P*, checks whether it can be allowed as member. It returns the group key CK_G, encrypted with the one-time pad, as well as the group's private key, encrypted as $CK_G(K_G^-)$.
3: *Q* authenticates *P* and sends back $K_{P,G}(N)$ letting *Q* know that it has all the necessary keys.

Secure group management



Question

Why didn't we send $K_P^+(CK_G)$ instead of using RP?

Authorization management

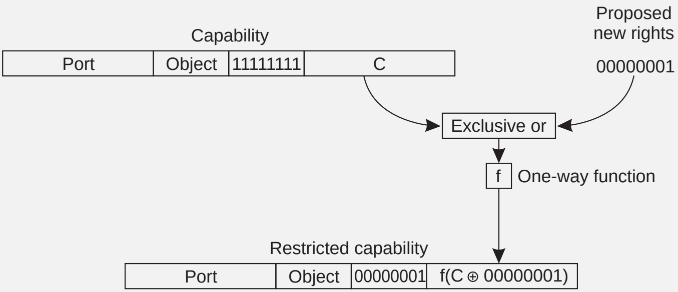
Issue

To avoid that each machine needs to know about all users, we use capabilities and attribute certificates to express the access rights that the holder has.

Authorization management

Amoeba

Restricted access rights are encoded in a capability, along with data for an integrity check to protect against tampering



Delegation

Observation

A subject sometimes wants to delegate its privileges to an object *O1*, to allow that object to request services from another object *O2*.

Example

A client tells the print server *PS* to fetch a file *F* from the file server *FS* to make a hard copy $\Rightarrow$ the client delegates its read privileges on *F* to *PS*

Nonsolution

Simply hand over your attribute certificate to a delegate (which may pass it on to the next one, etc.)

Delegate privileges

Problem

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?

Solution

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.

Delegate privileges