# ECE151 – Lecture 13

Chapter 8

Security

# Overview

Introduction

Secure channels

Access control

Security management

# Dependability Revisited

**Basics:** A *component* provides *services* to *clients*.
To provide services, the component may require the services from other components => a component may **depend** on some other component.

| | |
|---|---|
| **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 (even by authorized entities) |

**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

**Subject:** Entity capable of issuing a request for a service provided by an object

**Channel:** The carrier of requests and replies for services

**Object:** Entity providing services to subjects.

Channels and objects are subject to **security 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.

**Note:** authorization makes sense only if the requesting subject has been authenticated

# 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

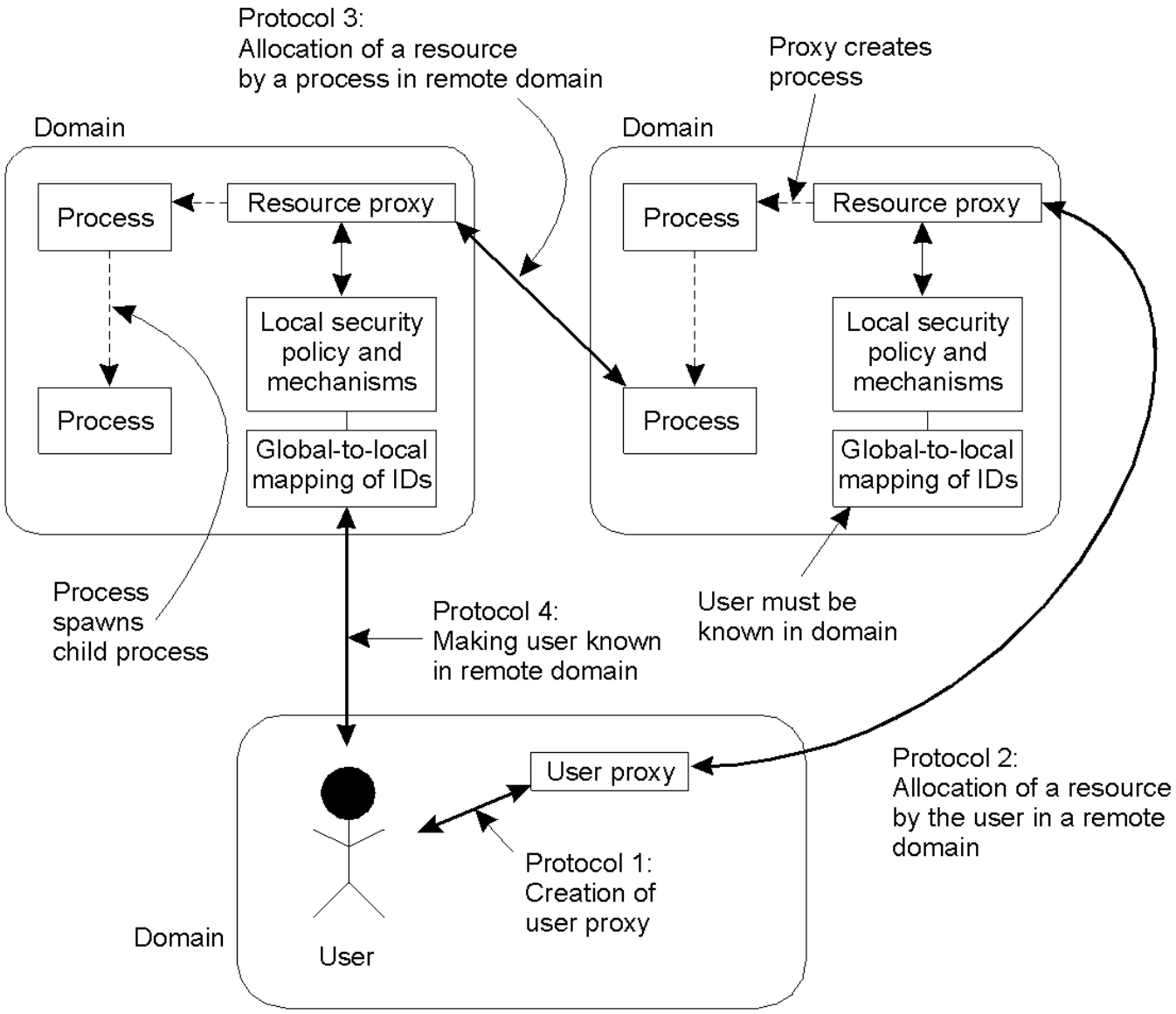
Users can delegate privileges to processes

Credentials can be shared between processes in the same domain

# Example: Globus Security Architecture

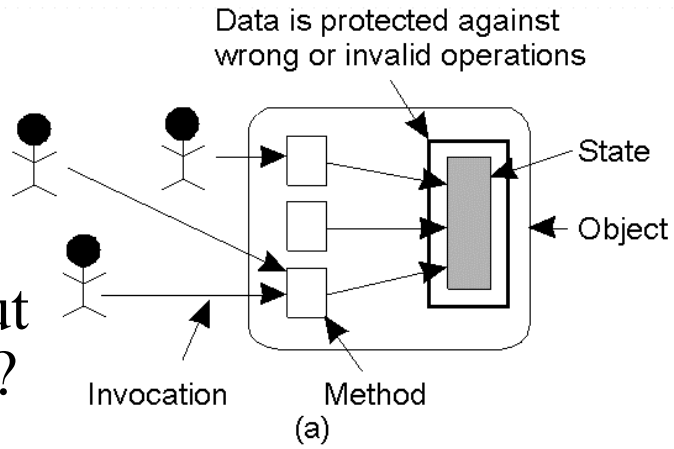
Diagram of Globus security architecture.

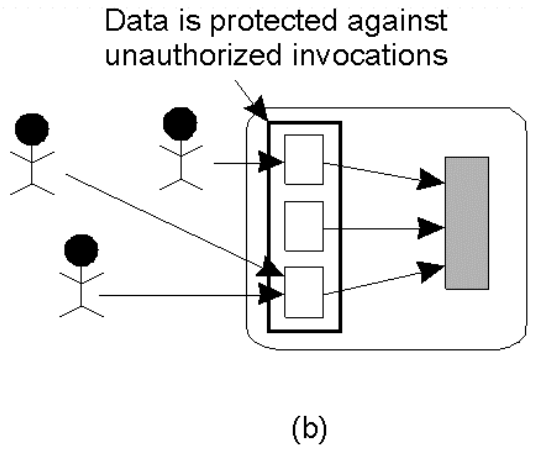
Policy statements lead to the introduction of mechanisms for cross-domain authentication and making users globally known
=>
**user proxies** and **resource proxies**



# Focus of Control

What is our focus when talking about protection of data?



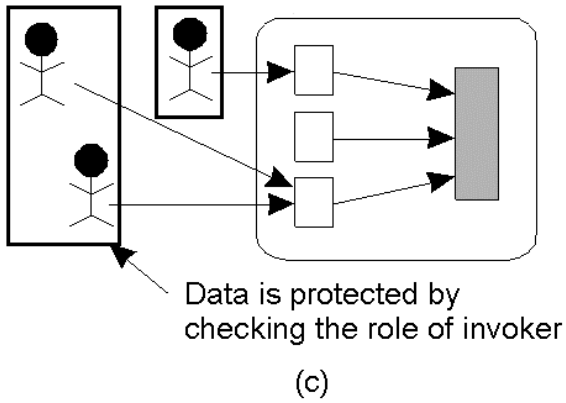




Three approaches for protection against security threats

a) Protection against invalid operations

b) Protection against unauthorized invocations

c) Protection against unauthorized users



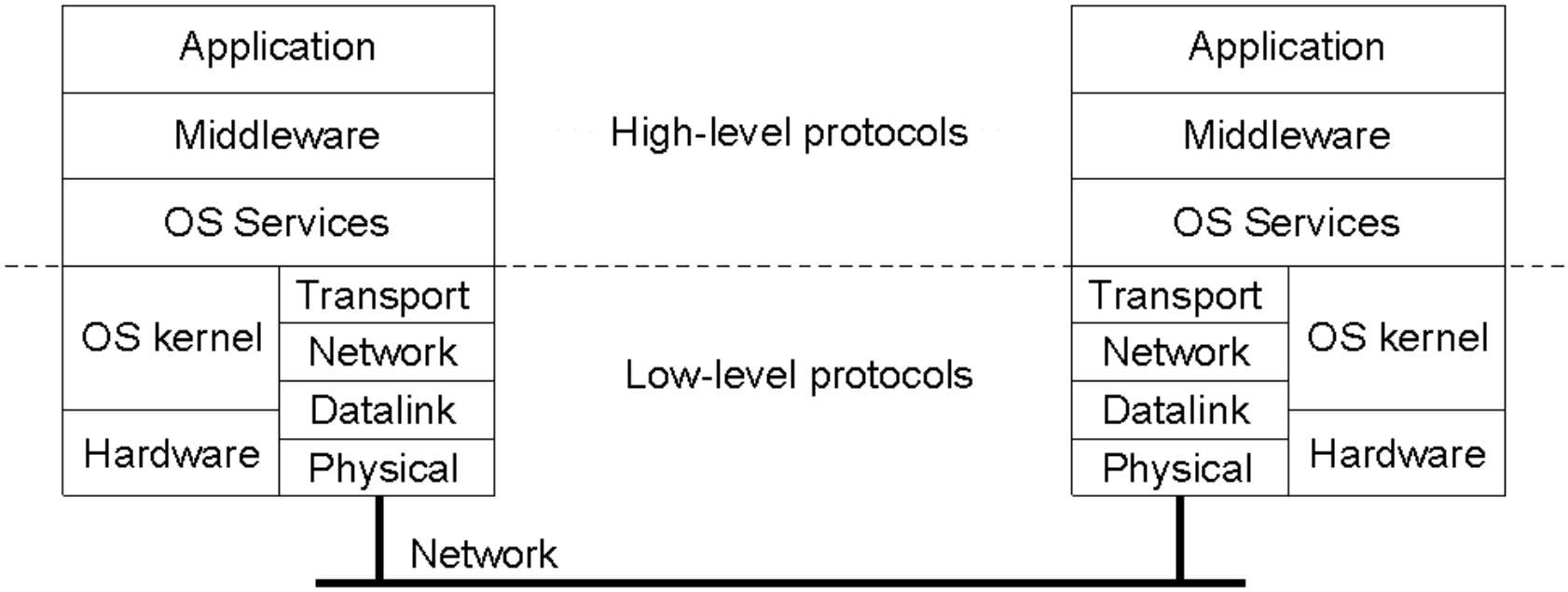

**Note:** We generally need all three, but each requires different mechanisms

# Layering of Security Mechanisms

The logical organization of a distributed system into several layers.
At which logical level are we going to implement security mechanisms?

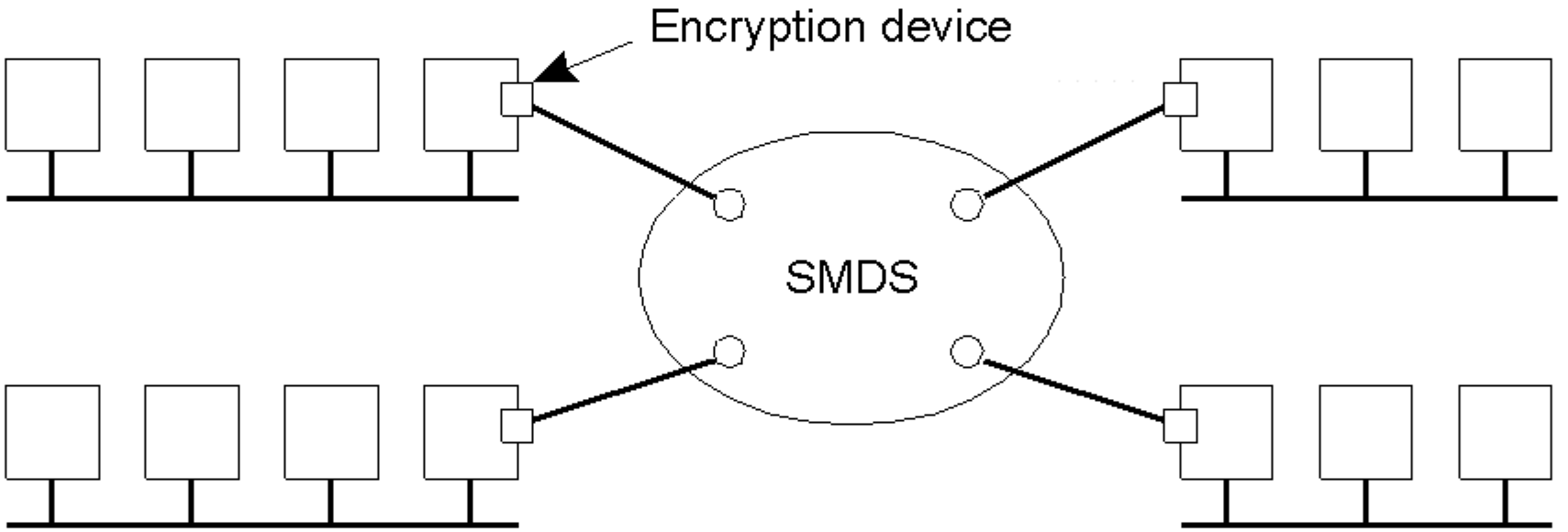
Whether security mechanisms are actually used is related to the **trust** a user has in those mechanisms. No trust => implement your own.

**Trusted Computing Base:** What is the set of mechanisms needed to enforce a policy. The smaller, the better.



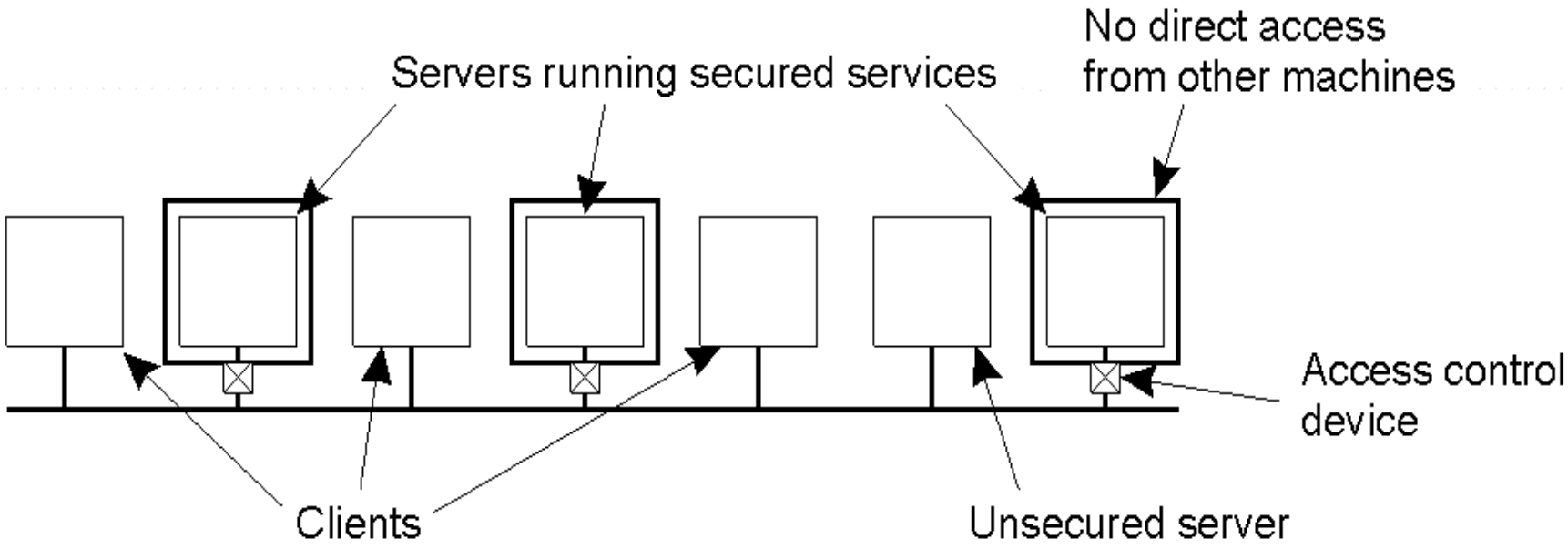
# Layering of Security Mechanisms

Several sites connected through a wide-area backbone service.

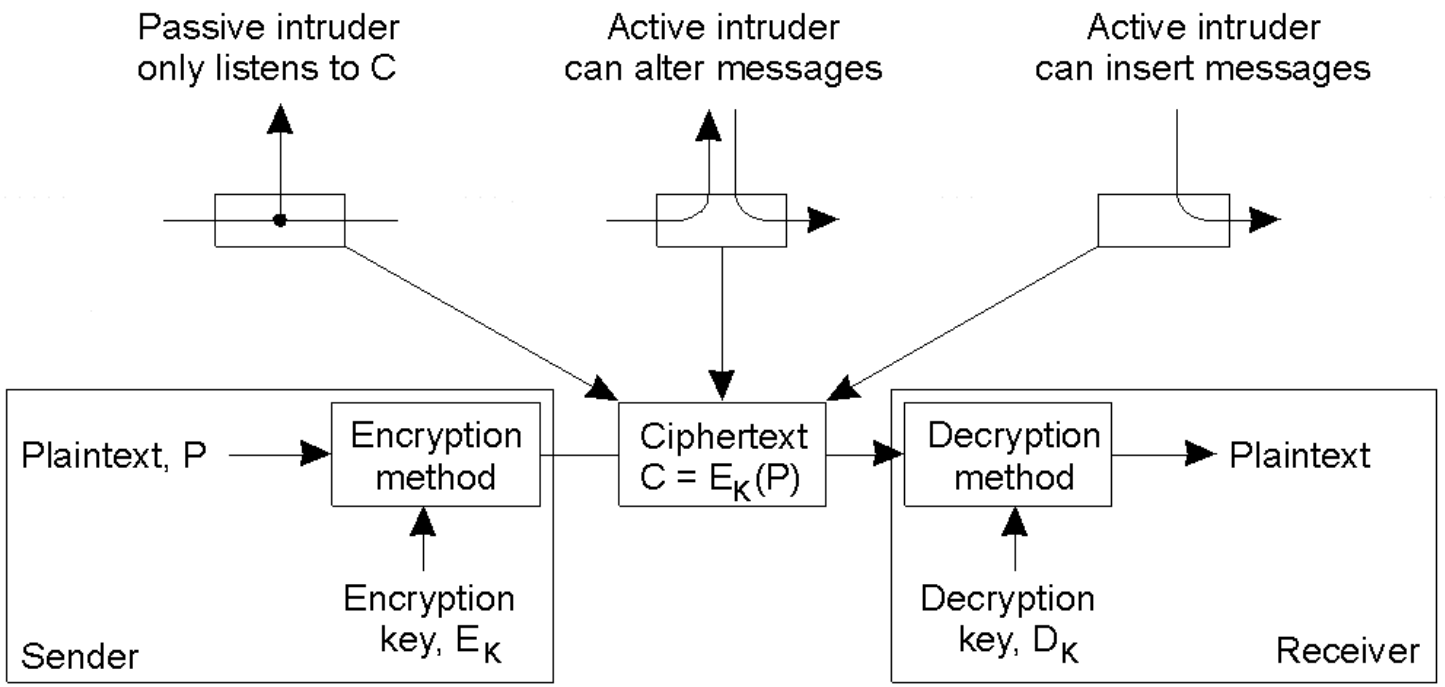


# Distribution of Security Mechanisms

The principle of RISSC as applied to secure distributed systems.



# Cryptography




**Symmetric system:** Use a single key to encrypt the plaintext and decrypt the ciphertext. 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.

# Cryptography

Notation used in this chapter.

| Notation | Description |
|---|---|
| $K_{A, B}$ | Secret key shared by A and B |
| $K_A^+$ | Public key of A |
| $K_A^-$ | Private key of A |

# 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 $m_{\text{out}}$ of $E_S$, it is (analytically or) computationally infeasible to find $m_{\text{in}} : E_S(m_{\text{in}}) = m_{\text{out}}$

**Weak collision resistance:** Given a pair $(m, E_S(m))$, 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

**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 $m, S, E$, 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 $m$: $E_K(m) = E_{K^*}(m)$

**Note:** Not all cryptographic functions have keys (such as hash functions)

# Secure Channels

**Goal:** Set up a channel allowing for secure communication between two processes:

They both know who is on the other side (authenticated).

They both know that messages cannot be tampered with (integrity).

They both know messages cannot leak out (confidentiality).

# Authentication versus Integrity

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

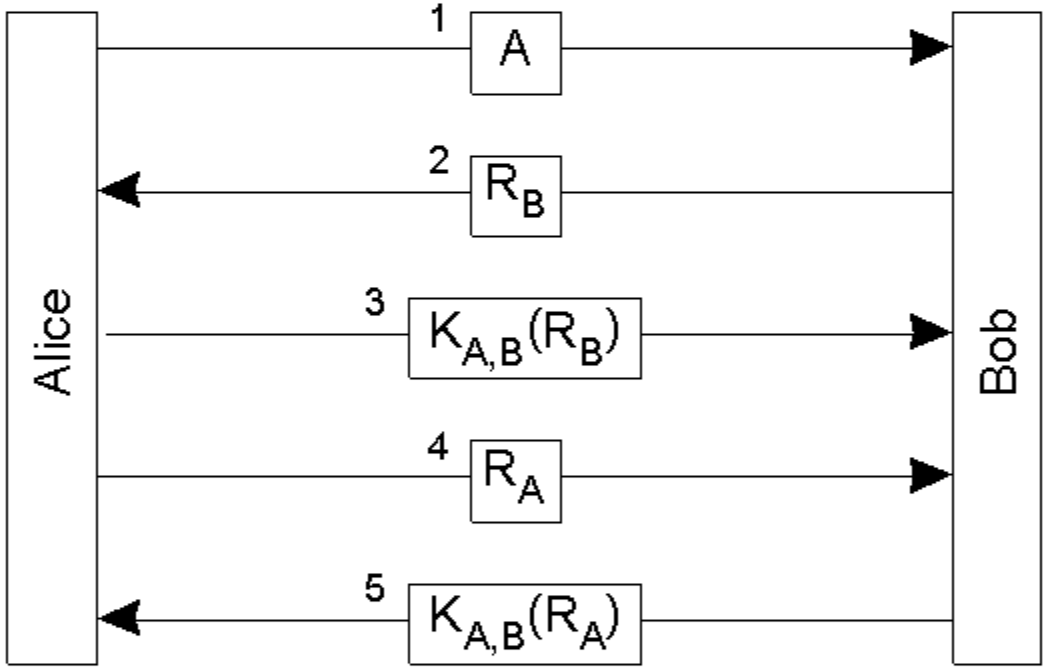
**Authentication without integrity:** Alice's message is authenticated, and intercepted by Trudy, who tampers with its content, but leaves the authentication part as is. Authentication has become meaningless.

**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.

**Question:** What can we say about confidentiality versus authentication and integrity?

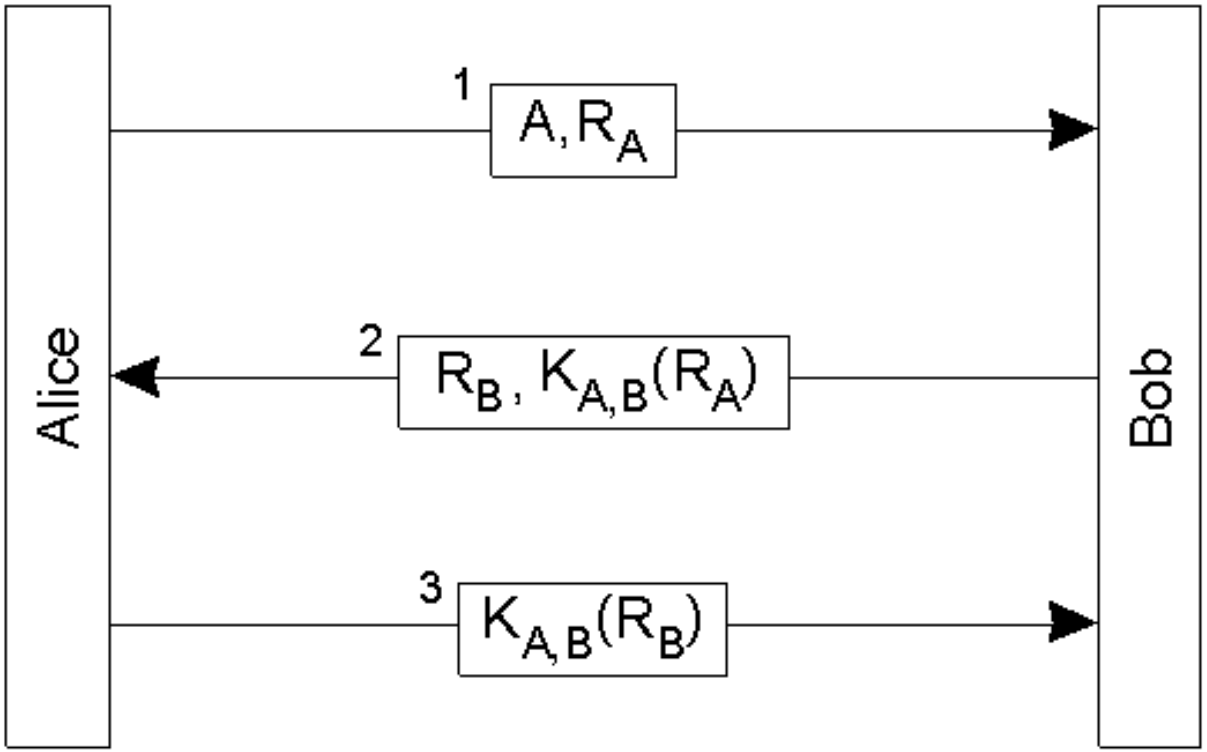
# Authentication

Authentication based on a shared secret key.



# Authentication

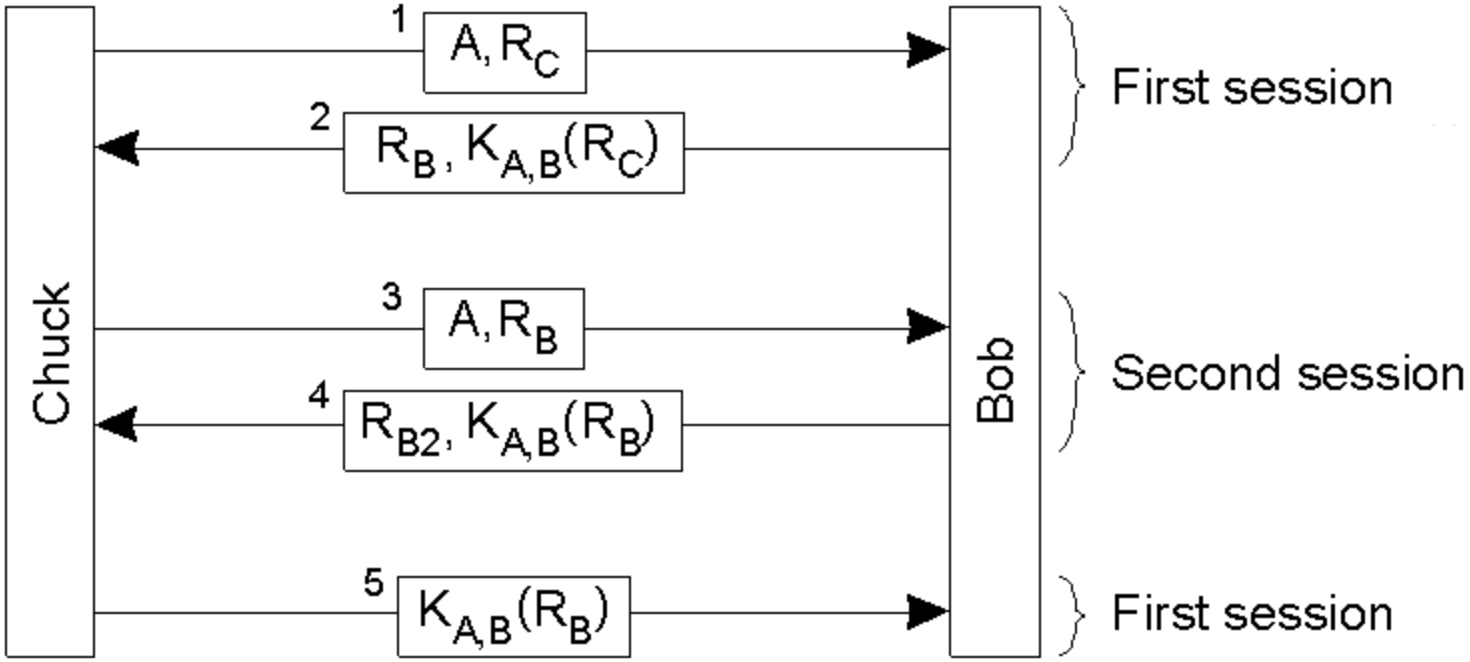
Authentication based on a shared secret key, but using three instead of five messages.




**This is not sound**

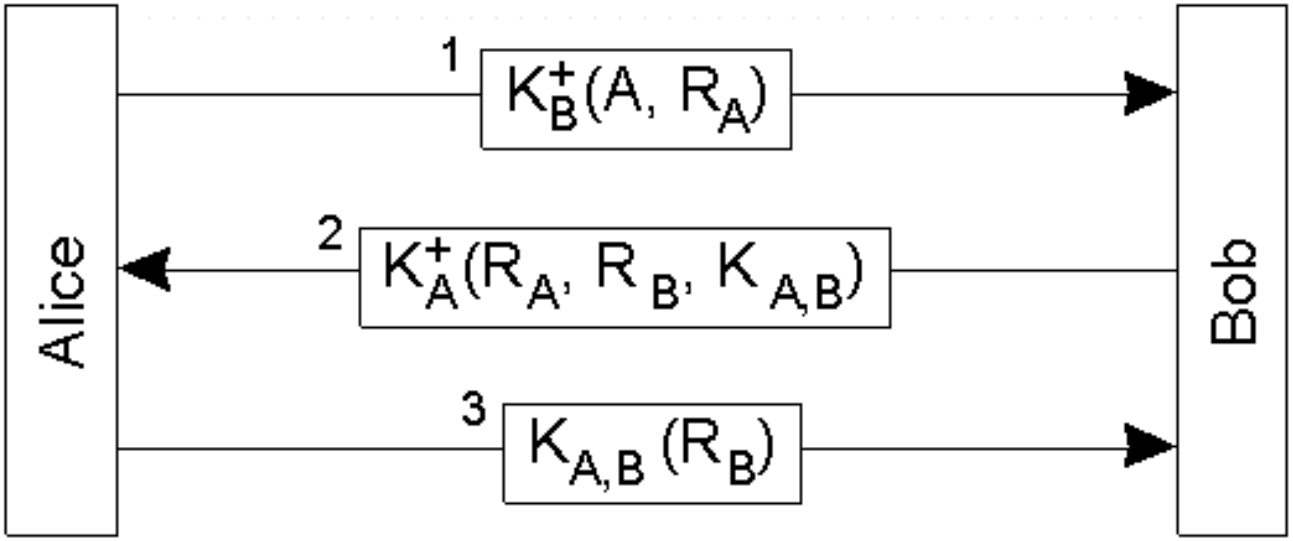
# Authentication

The reflection attack.



# Authentication Using Public-Key Cryptography

Mutual authentication in a public-key cryptosystem.

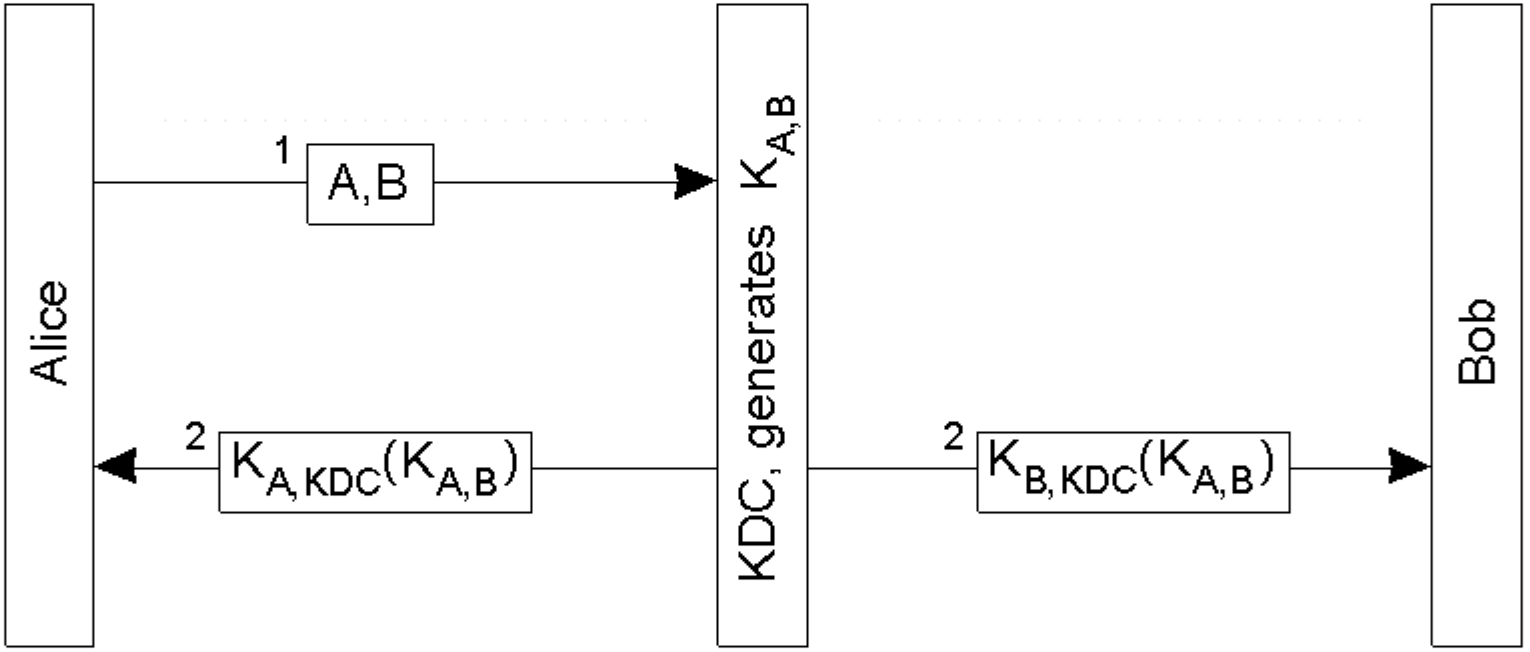



$K_{A,B}$ is also known as a **session key**

# Authentication Using a Key Distribution Center

**Problem:** With *N* subjects, we need to manage *N*(*N*-1)/2 keys, each subject knowing *N*-1 keys.

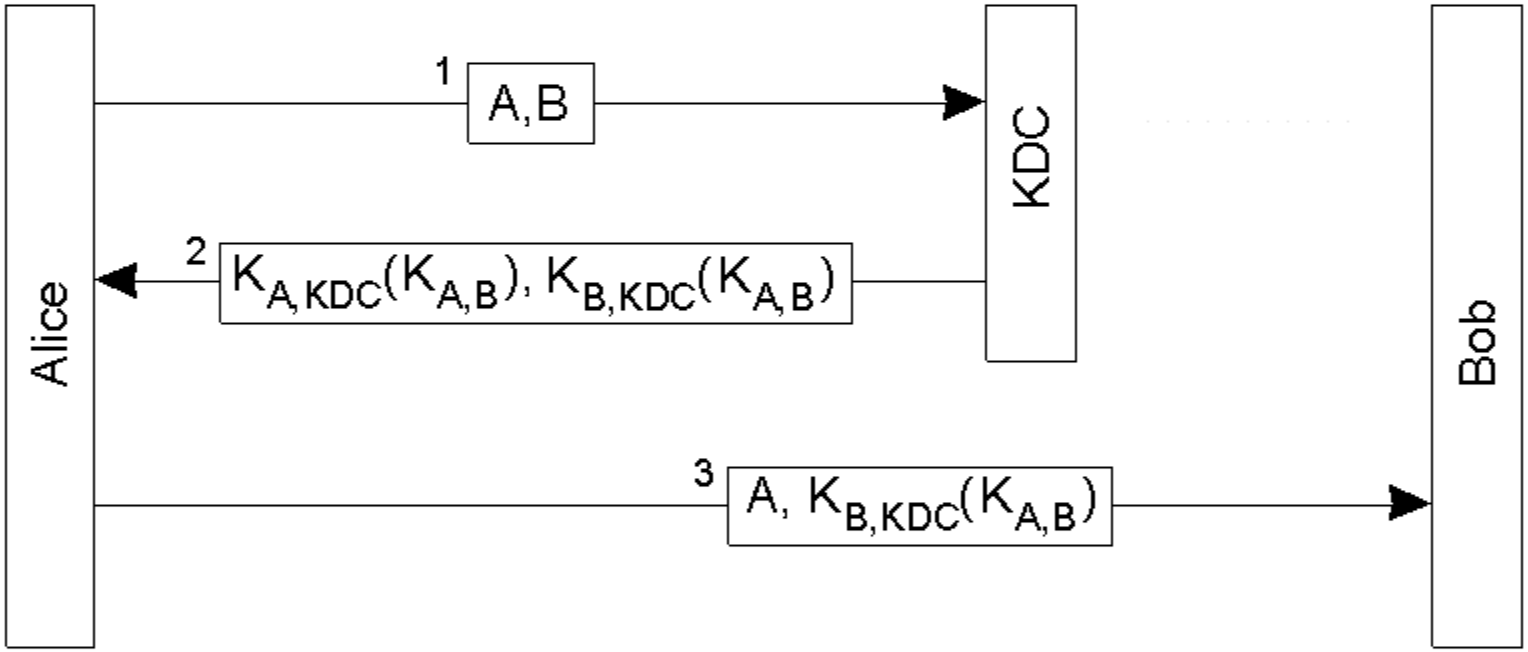
**Solution:** Use a trusted **Key Distribution Center** that generates keys when necessary.



# Authentication Using a Key Distribution Center

**Problem:** We give Bob the key $K_{A,B}$ before Bob knows about Alice

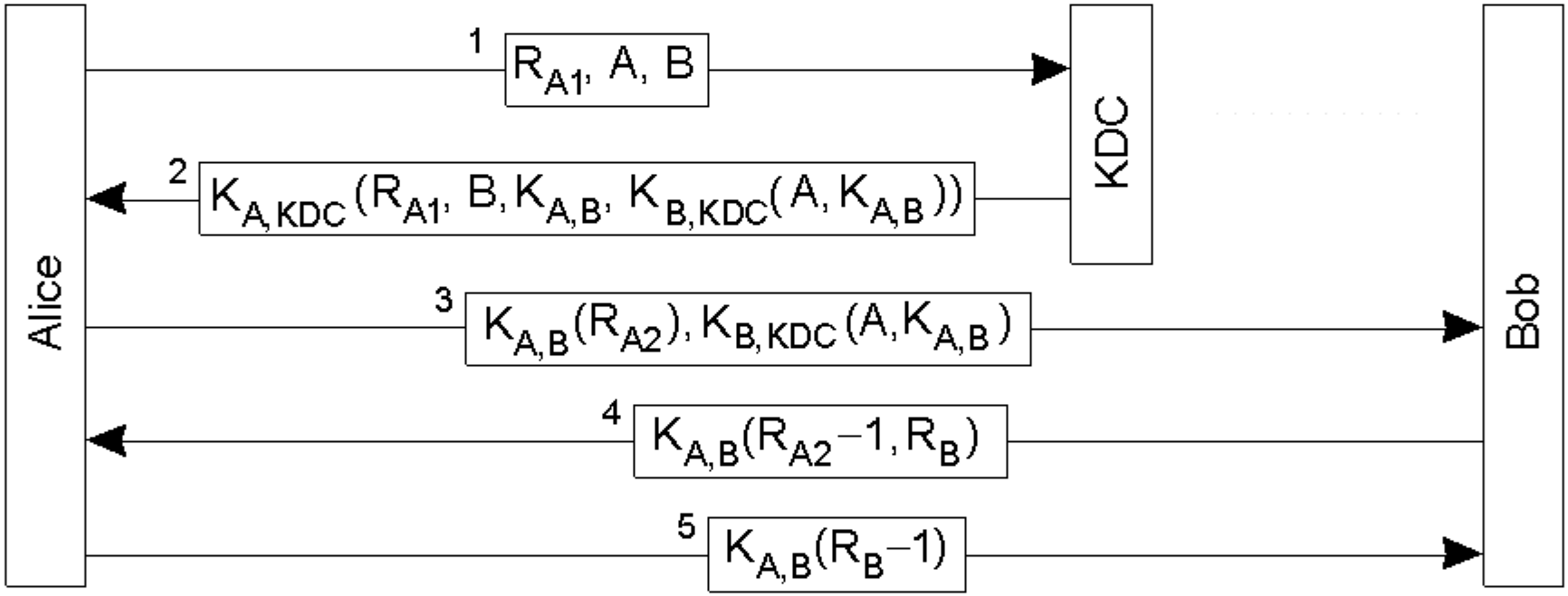
**Solution:** Give Alice a **ticket** to set up a secure channel with Bob




**Vulnerable to replay attacks**

# Authentication Using a Key Distribution Center

The Needham-Schroeder authentication protocol.



# Needham-Schroeder: Subtleties

Q1: Why does the KDC put Bob into its reply message, and Alice into the ticket?

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

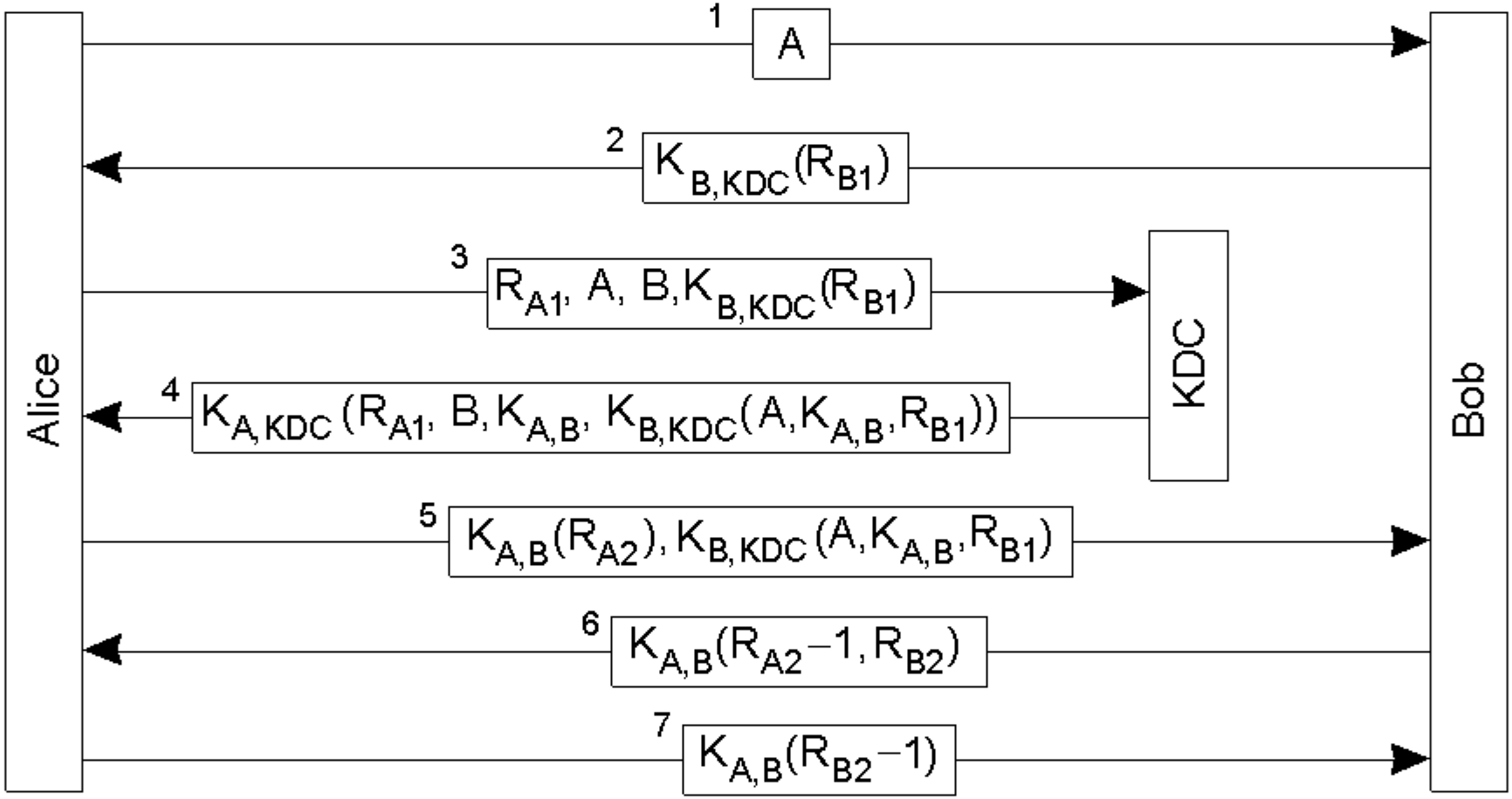
**Security flaw:** Suppose Chuck finds out Alice's key he can use that key anytime to impersonate Alice, even if Alice changes her private key at the KDC.

**Reasoning:** Once Chuck finds out Alice's key, he can use it to decrypt a (possibly old) ticket for a session with Bob, and convince Bob to talk to him 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 => we're now ensuring that every session is known at the KDC.

# Authentication Using a Key Distribution Center

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



# Confidentiality

**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: $K_B^+(m)$

**There are a number of problems with keys:**

**Keys wear out:** The more data is encrypted by a single key, the easier it is to find that key => *don't use keys too often*

**Danger of replay:** Using the same key for different sessions, permits old messages to be inserted into the current session => *don't use keys for different sessions*

# Confidentiality

**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
*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
*make keys disposable*

**Essence:** Don't use valuable and expensive keys for all communication, but only for authentication purposes.

**Solution:** 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

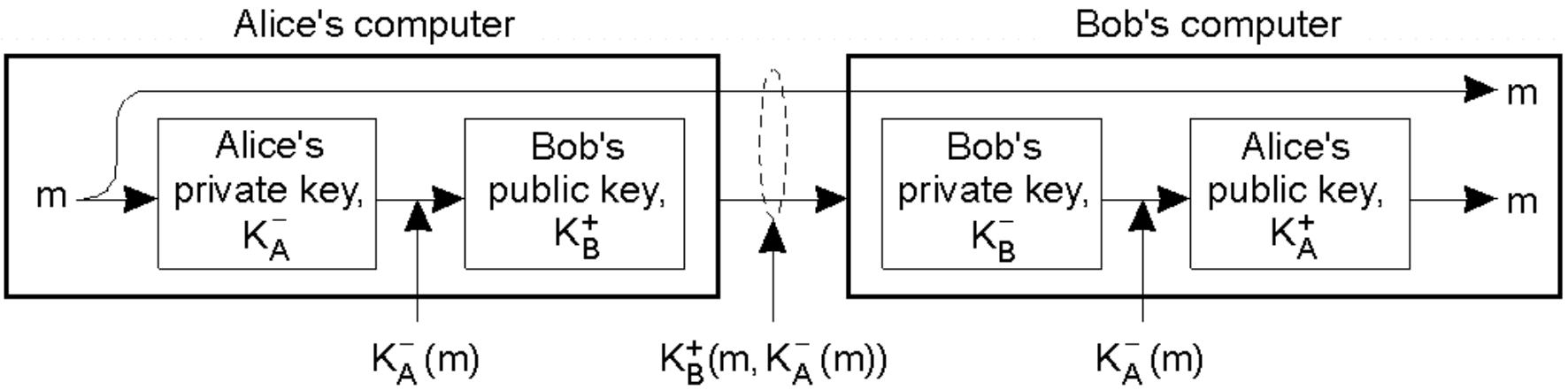
**Integrity:** The message cannot be maliciously altered during, or after receipt

**Solution:** Let a sender sign all transmitted messages, in such a way that

(1) the signature can be verified and

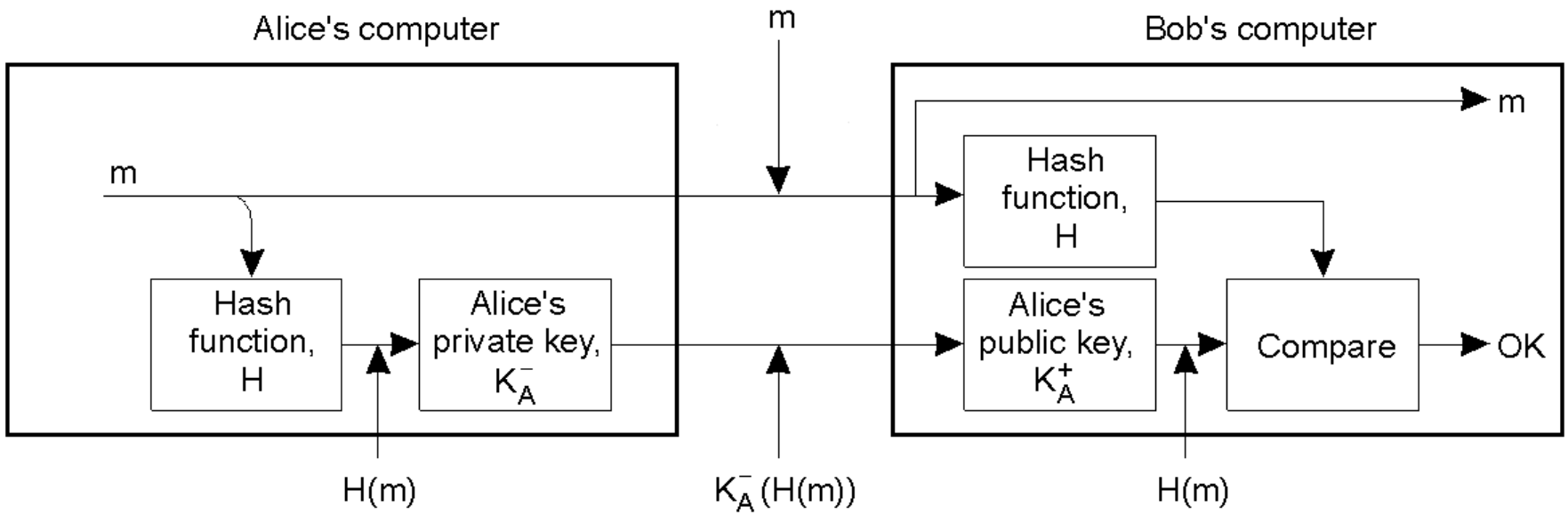
(2) message and signature are uniquely associated

# Digital Signatures




Digital signing a message using public-key cryptography.

# Digital Signatures




**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.

**Solution:** take a message digest, and sign that: