Distributed Systems Lecture 12

Professor Louise E. Moser

Spring 2013

Security

- Introduction
- Secure channels
- Access control
- Security management

Dependability Revisited

- **Basics:** A **component** provides **services** to **clients**
- To provide services, a component might require services of other components and, thus, it might **depend** on other components
- Properties of dependability

Availability	Readiness for usage
Reliability	Continuity of service delivery
Safety	Very low probability of catastrophes
Maintainability	Ease with which a failed system be repaired
Confidentiality	No unauthorized disclosure of information
Integrity	No accidental or malicious alterations of information (even by authorized entities)

- **Note:** In distributed systems, **security** is the combination of **confidentiality** and **integrity**. 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 contents of transferred messages**	**Reading data contained in an object**
☐ **Modification**	**Changing message content**	**Changing an object's data**
☐ **Fabrication**	**Inserting messages**	**Spoofing an object**

Security Mechanisms

- **Security mechanism:** Protects against security threats. Some security mechanisms are:
  - **Encryption: Transform data into other data that an attacker cannot understand (confidentiality). Also used to check whether data have been modified (integrity)**
  - **Authentication: Verify that a subject is who he/she claims to be, i.e., verify the identity of a subject**
  - **Authorization: Determine whether a subject is permitted to use certain services**
  - **Auditing: Trace what data subjects accessed, and how they did so. Useful only if it can help catch an attacker**
- **Note:** Authorization makes sense only if the requesting subject has been authenticated

Security Policies

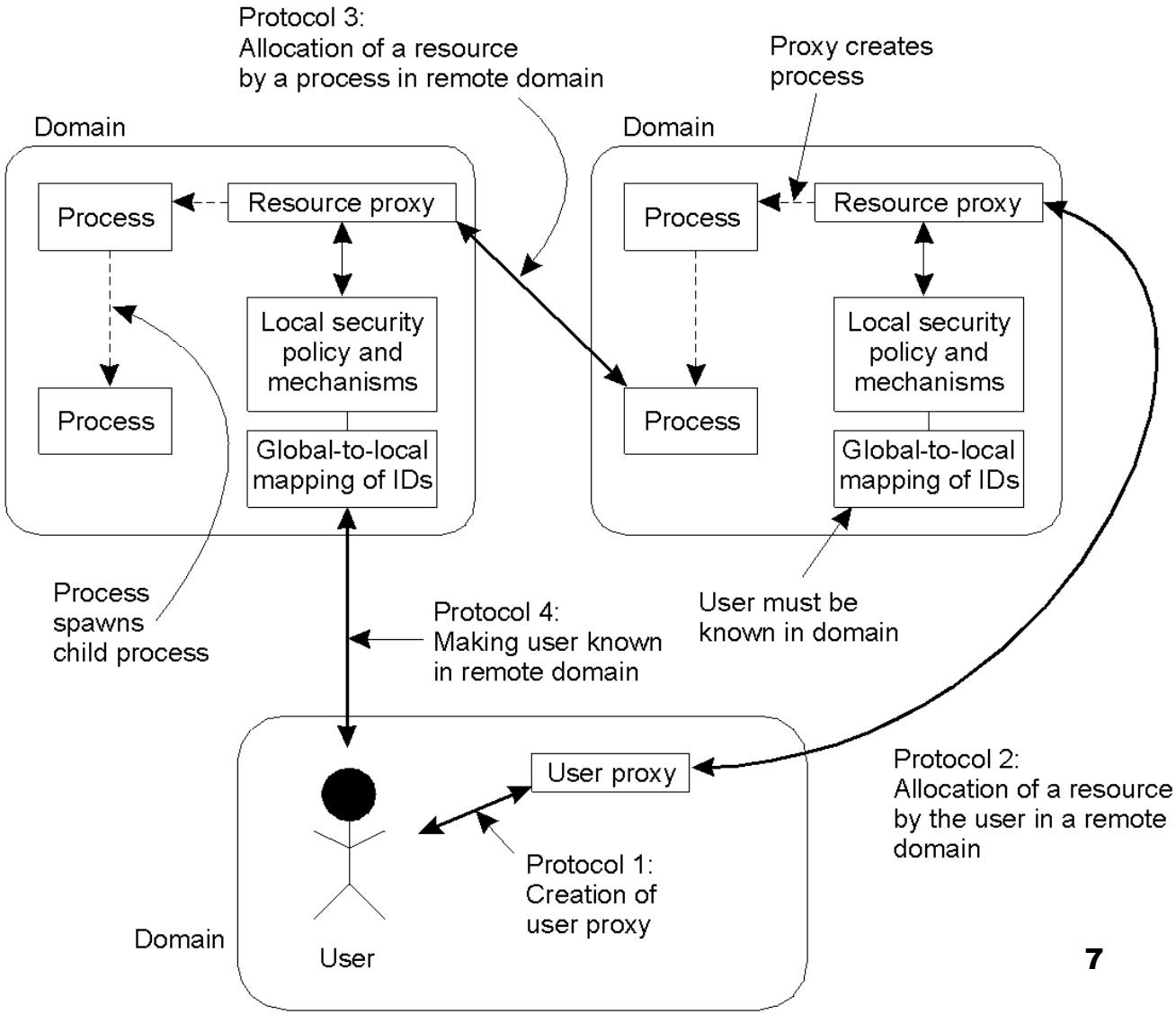
- **Security policy:** Prescribes how to use mechanisms to protect against attacks. Requires that a model of possible attacks is described, i.e., a **security architecture**

- **Example: Globus security architecture**
  - **There are multiple administrative domains**
  - **Local operations are subject to local security policies**
  - **Global operations require requester to be globally known**
  - **Inter-domain operations require mutual authentication**
  - **Global authentication replaces local authentication**
  - **Users can delegate privileges to processes**
  - **Processes in the same domain can share credentials**

Globus Security Architecture

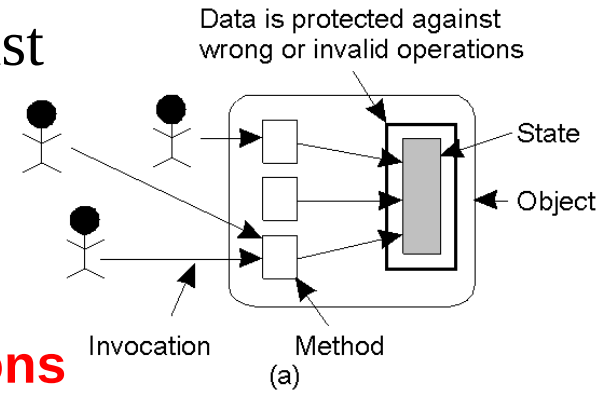
- Policies lead to mechanisms for cross-domain authentication and users who are globally known =>

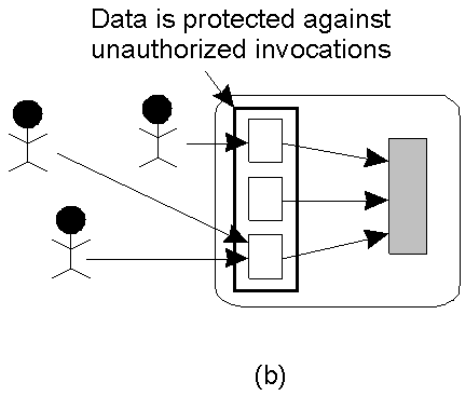
  user proxies and **resource proxies**

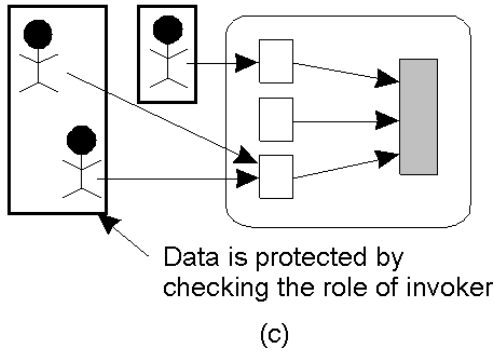


Protection of Data

- Three approaches for protection of data against security threats
  - **(a) Protection against invalid operations**
  - **(b) Protection against unauthorized invocations**
  - **(c) Protection against unauthorized users**
- Note: Generally, we need all three, but each requires different mechanisms

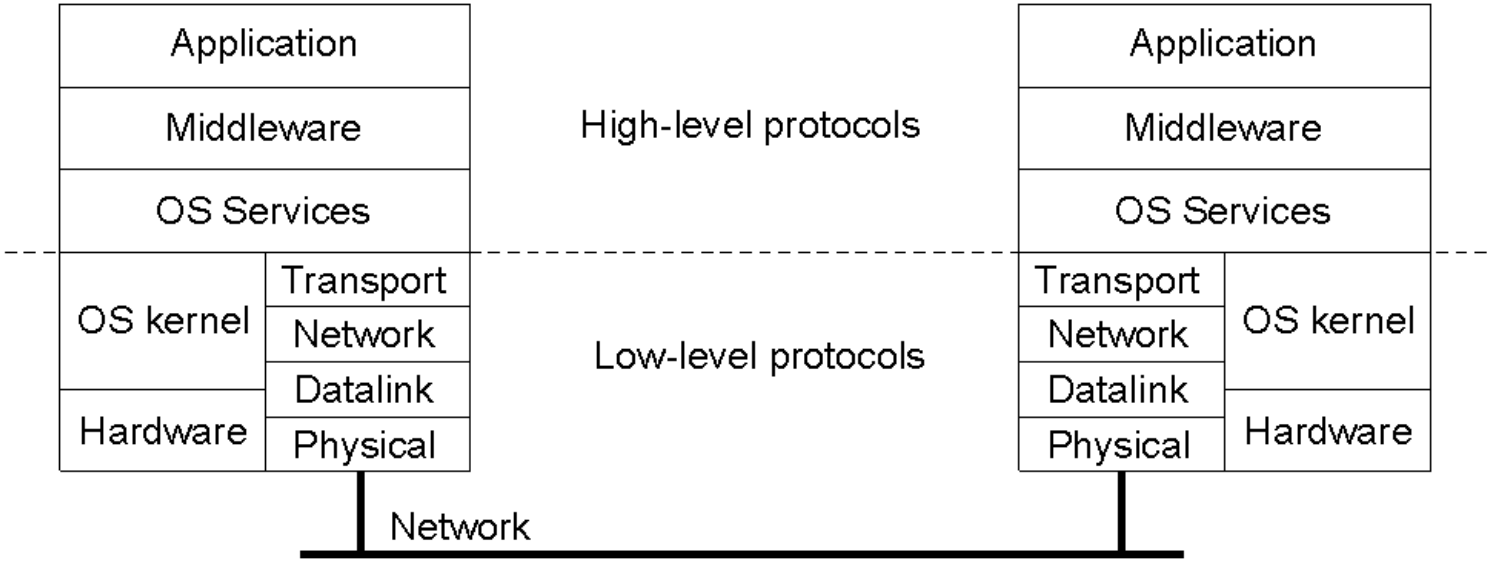






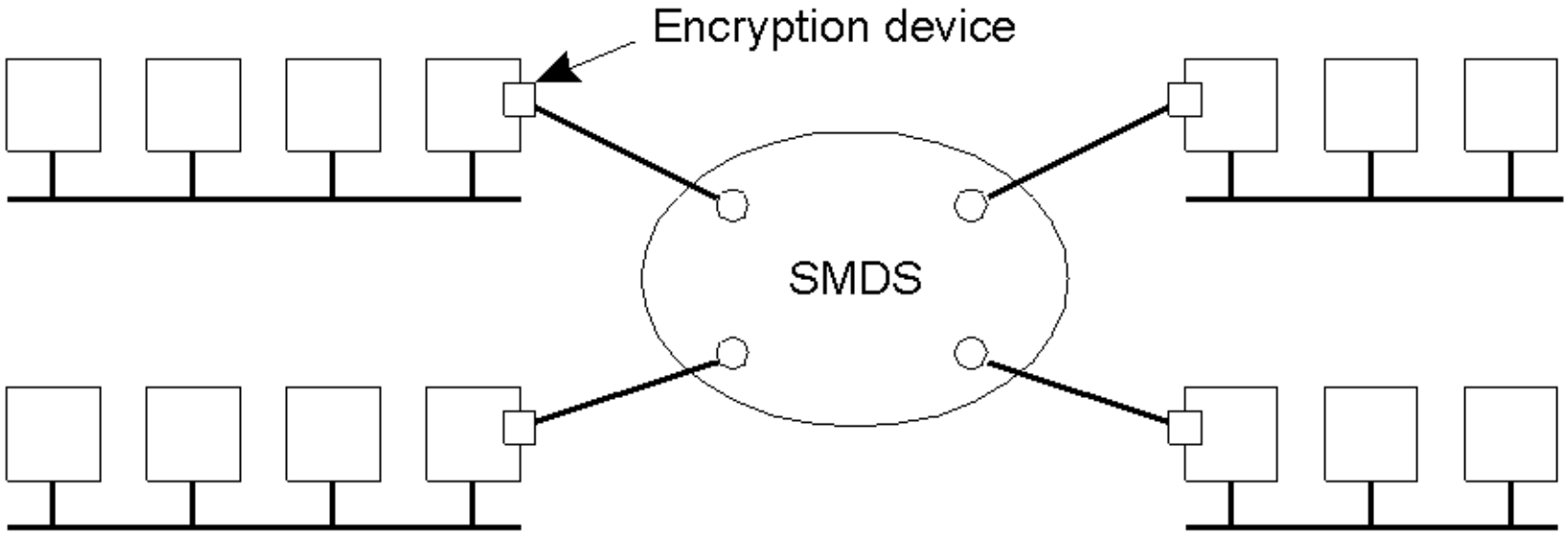
Layering of Security Mechanisms

- The logical organization of a distributed system into several layers
  At which logical level should we 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 (TCB):** What is the set of mechanisms needed to enforce a security policy? The smaller, the better



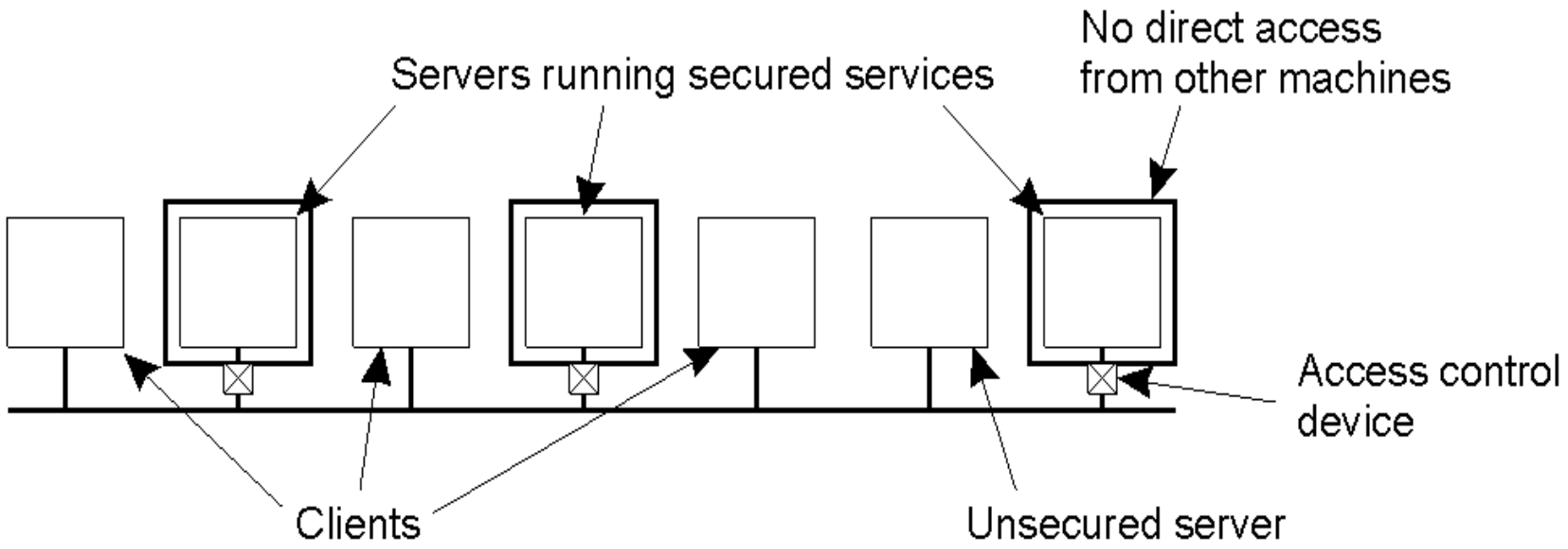
Layering of Security Mechanisms

- **Encryption devices** for local-area networks of computers connected through a wide-area backbone service, called the Switched Multi-Megabit Data Service (SMDS)



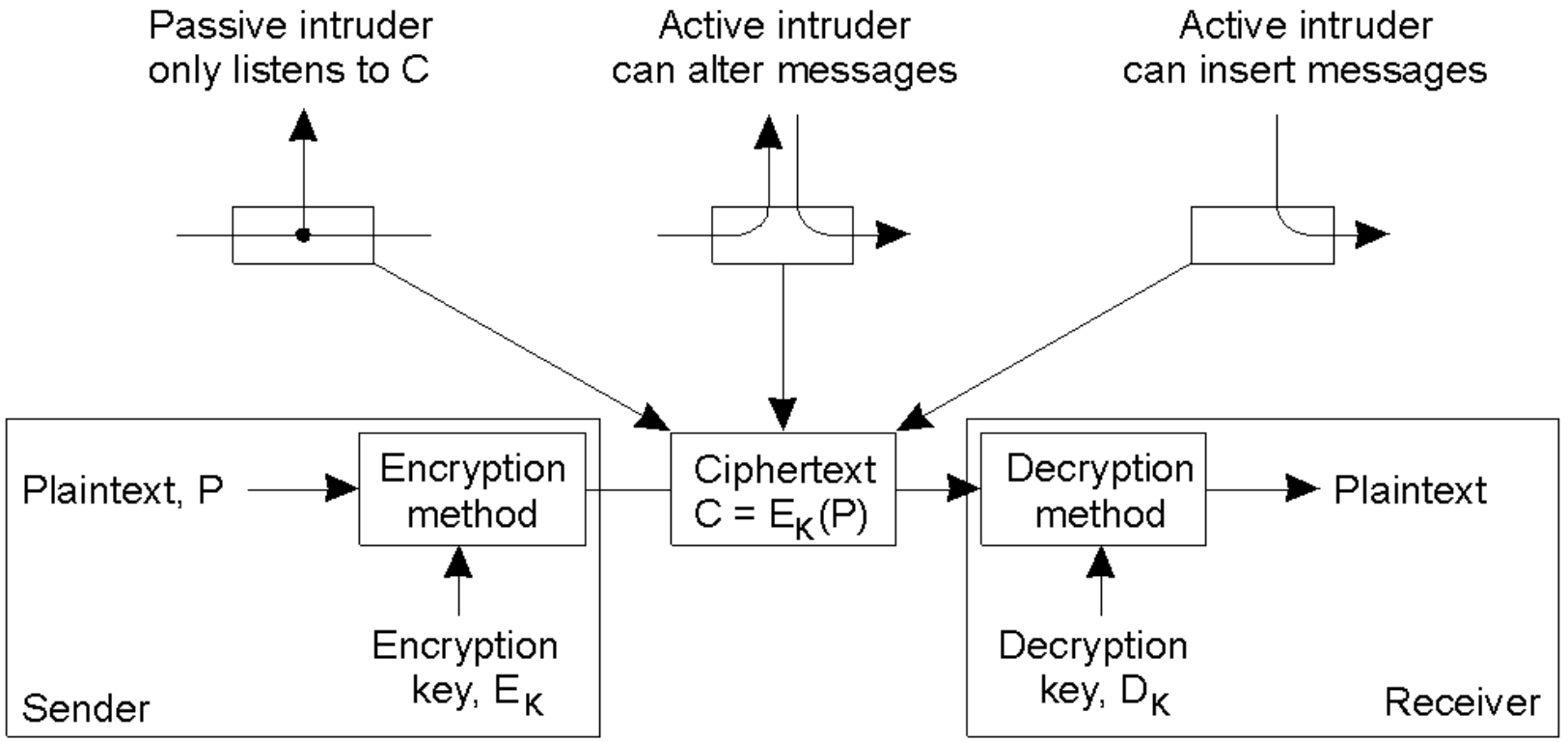
Layering of Security Mechanisms

- **Access control devices** on several computers that provide Reduced Interfaces for Secure System Component (RISSC)



Cryptography

- Intruders and eavesdroppers in communication, and the use of encryption and decryption

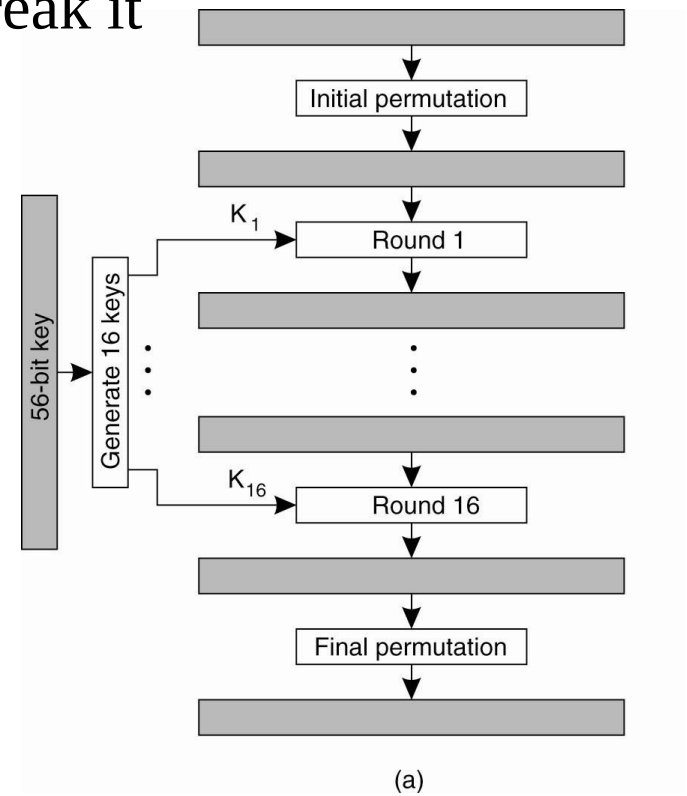


Cryptography

- **Different kinds of Cryptographic Systems**
  - **Symmetric System:** Use a **single** key to encrypt the plaintext and to decrypt the ciphertext. The sender and the receiver **share** a secret key
  - **Asymmetric System (Public Key Cryptosystem):** Use different keys for encryption and decryption, one **public** and one **private**
  - **Hashing System:** Encrypt the data and produce a fixed-length **digest**. No decryption; only comparison

Symmetric Cryptosystem

- **Data Encryption Standard (DES) Algorithm**, U.S. standard for unclassified information, developed by IBM (1974)
- **Aim:** To make the encryption algorithm so complicated that not even a computer can break it
- Same key for encryption and decryption
- Encrypts in 64-bit blocks
- Uses 56-bit key
- Has 19 stages, 16 stages that each use a different key K_i as the parameter of the function f

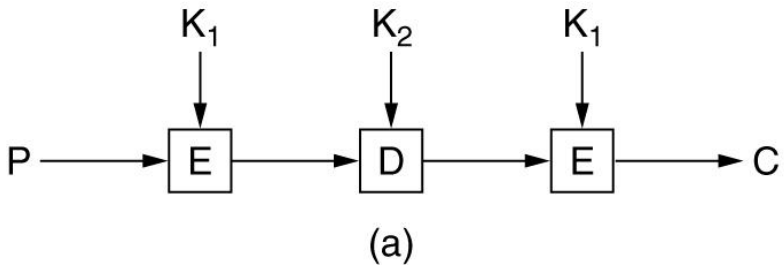


Symmetric Cryptosystem

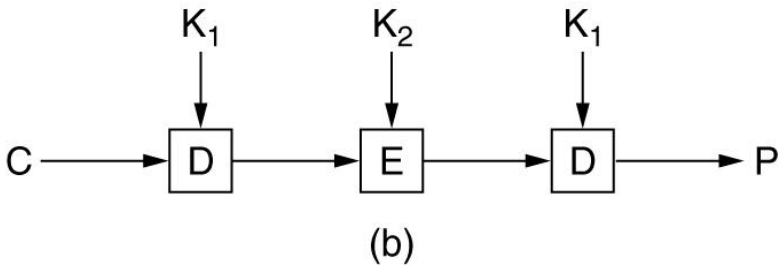
- **DES Algorithm**
  - **Each iteration i of the 16 iterations uses a different key K_i**
    1. Before the algorithm starts, apply a 56-bit transposition to the key
    2. Before each iteration i, partition the key into two 28-bit numbers, and rotate the left half by the number of bits determined by the iteration number i
    3. Obtain the key K_i from the rotated key by applying another 56-bit transposition
  - **The function f involves four steps**
    1. Construct a 48-bit number E by expanding a 32-bit number R_{i-1} according to a transposition and duplication rule
    2. XOR E and K_i
    3. a. Partition the output into 8 groups of 6 bits each
       b. Input each group to a different substitution box
       c. Substitution box produces 4 output bits
       d. Result is 8 4-bit numbers
    4. Pass 32 bits through a permutation box

Symmetric Cryptosystem

- **Triple DES** – Effectively increases the key length, uses two keys K_1 and K_2 and three stages
  - **In the first stage, encrypt the plaintext using DES in the usual way with the key K_1**
  - **In the second stage, run DES in decryption mode, using the key K_2**
  - **In the third stage, encrypt again using DES with the key K_1**



Triple DES Encryption



Triple DES Decryption

Asymmetric Cryptosystem

- **Diffie Hellman Algorithm** (1976) – Radically different from a symmetric cryptosystem; uses *different* keys (rather than the same key) for encryption and decryption
- Uses an encryption algorithm E and a decryption algorithm D, such that even if you know E you can't determine D
- Three requirements:
  - **D(E(P)) = P**
  - **Can't deduce D (private) from E (public)**
  - **Can't break D by a plaintext attack**
- A makes public its encryption algorithm and key E_A
  B makes public its encryption algorithm and key E_B
- A encrypts A's message P using B's public key E_B to obtain $E_B(P)$
  B decrypts A's message using B's private key D_B to obtain $D_B(E_B(P)) = P$

Asymmetric Cryptosystem

- **RSA algorithm** - named after Rivest, Shamir, Adelman (1978)
- Four steps:
  1. **Choose two large primes p and q (typically 1024 bits)**
  2. **Compute $n = p \times q$ and $z = (p-1) \times (q-1)$**
  3. **Find a number d relatively prime to z**
  4. **Find a number e such that $e \times d \equiv 1 \bmod z$**
- Divide the plaintext into blocks P of k bits where k is the largest integer such that $2^k < n \Rightarrow 0 \leq P < n$
- The **public** key consists of the pair (e, n)
  The **private** key consists of the pair (d, n)
- To **encrypt** P, compute $C = P^e \bmod n$
  To **decrypt** C, compute $P = C^d \bmod n$
- RSA algorithm is based on the difficulty of factoring large numbers
  - **If you can factor n, then you know p and q, and also z**
  - **If you know z and e, you can find d**

Asymmetric Cryptosystem

- A simple example of the RSA algorithm
  - **p = 3, q = 11 => n = p x q = 33 and z = (p-1) x (q-1) = 20**
  - **Find d relatively prime to z, e.g., d = 7**
  - **Find e by solving 7 x e ≡ 1 mod 20, e.g., e = 3**
  - **$C \equiv P^3 \bmod 33$, $P \equiv C^7 \bmod 33$**

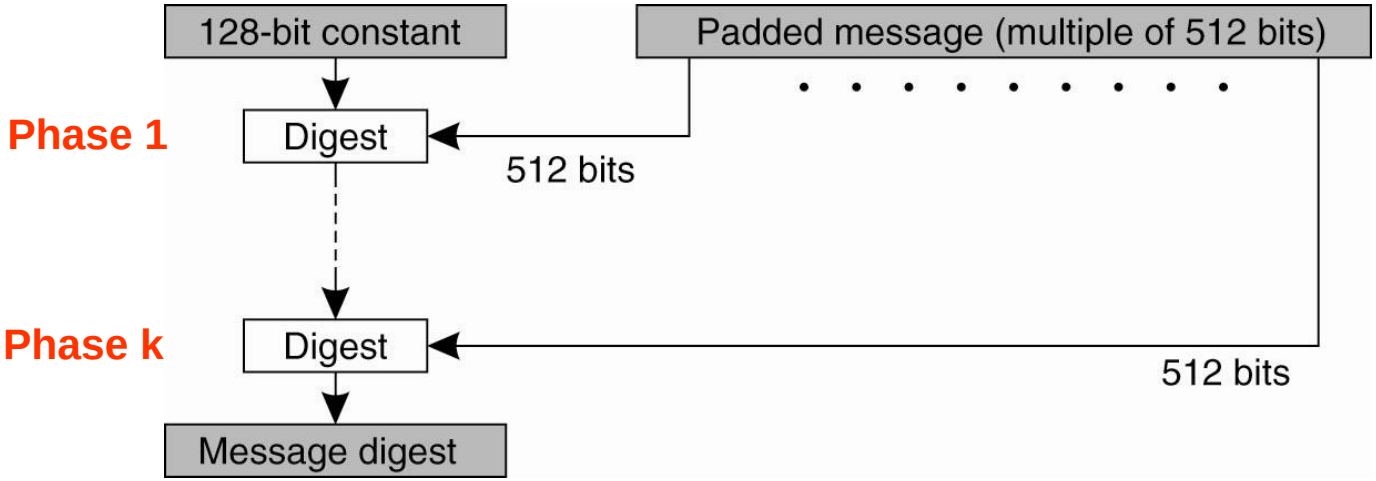
Plaintext (P)			Ciphertext (C)		After decryption	
Symbolic	Numeric	P^3	P^3 (mod 33)	C^7	C^7 (mod 33)	Symbolic
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E
Sender's computation				Receiver's computation		

Hash Function

- **MD5** – Hash function for computing a 128-bit fixed-length **message digest** from an arbitrary-length binary input string
  - **Convert the input string to a sequence of k 512-bit blocks**

    (a) Pad the input string to a total length of 448 bits mod 512

    (b) Add the length of the original bit string as a 64-bit integer



Hash Function

- **MD5** (continued)
  - **The algorithm then proceeds in k phases**
    During each phase, a 128-bit digest is computed from the 512-bit block and the 128-bit digest in the preceding phase
  - **A phase consists of four rounds of computation**
    Each round uses one of the following four functions:
    F(x,y,z) = (x AND y) OR ((NOT x) AND z)
    G(x,y,z) = (x AND z) OR (y AND (NOT z))
    H(x,y,z) = x XOR y XOR z
    I(x,y,z) =y XOR (x OR (NOT z))
  - **MD5 uses only AND, OR, NOT, XOR, and can be implemented in hardware, but is still quite complex (see the text p. 396 for an example)**

Cryptographic Functions

- **Essence:** Make the encryption method E public, but let the encryption as a whole be parameterized by means of a key K (same for decryption)
- **One-way function:** Given an output message m_{out} of E_K, it is computationally infeasible to find an input message m_{in} such that $E_K(m_{in}) = m_{out}$
- **Weak collision resistance:** Given a pair m, $E_K(m)$, it is computationally infeasible to find a different input message $m^* \neq m$ such that $E_K(m^*) = E_K(m)$
- **Strong collision resistance:** It is computationally infeasible to find two different input messages $m \neq m^*$ such that $E_K(m) = E_K(m^*)$

Cryptographic Functions

- **One-way key:** Given an encrypted message m_{out}, a message m_{in}, and an encryption function E, it is computationally infeasible to find a key K such that $m_{out} = E_K(m_{in})$
- **Weak key collision resistance:** Given a triplet m, K, E, it is computationally infeasible to find a different key $K^* \neq K$ such that $E_{K^*}(m) = E_K(m)$
- **Strong key collision resistance:** It is computationally infeasible to find two different keys $K \neq K^*$ such that for all messages m $E_K(m) = E_{K^*}(m)$
- **Note:** Not all cryptographic functions have keys, for example, hash functions do not

Secure Channels

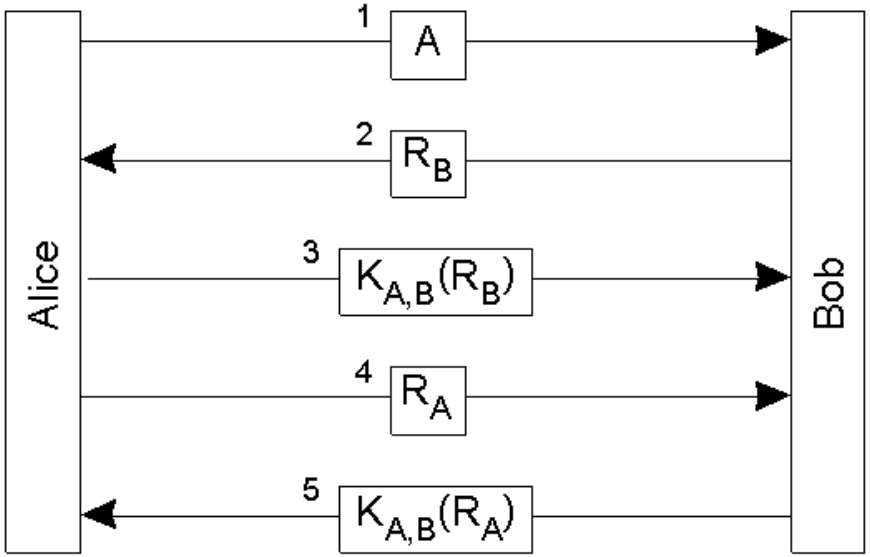
- **Goal:** Set up a channel for secure communication between two processes with the following properties
  - **Authentication:** **They both know who is on the other side of the channel**
  - **Integrity:** **They both know that messages cannot be tampered with**
  - **Confidentiality:** **They both know messages cannot be leaked**

Authentication vs. Integrity

- **Note:** Authentication and 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, but it is intercepted by Trudy, who tampers with its contents, but leaves the authentication part in tact
  - **Authentication has become meaningless**
- **Integrity without authentication:** Trudy intercepts a message from Alice, and then makes Bob believe that the contents were really sent by Trudy
  - **Integrity has become meaningless**
- **Question:** What can we say about authentication and integrity vs. confidentiality?

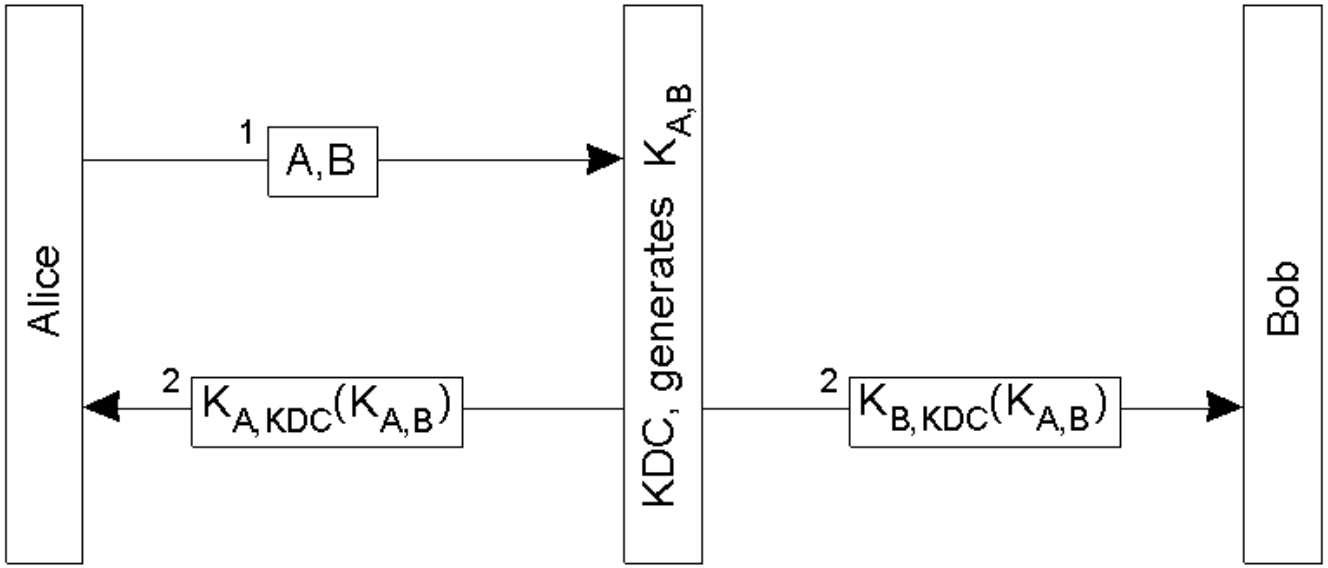
Authentication

- Authentication based on a **shared secret key** $K_{A,B}$ with a **nonce** R_A for Alice and a **nonce** R_B for Bob
  - **A nonce is a number used only once, such as a timestamp, that is used to relate two (or more) messages to each other and that the response is not a replay of an older message**



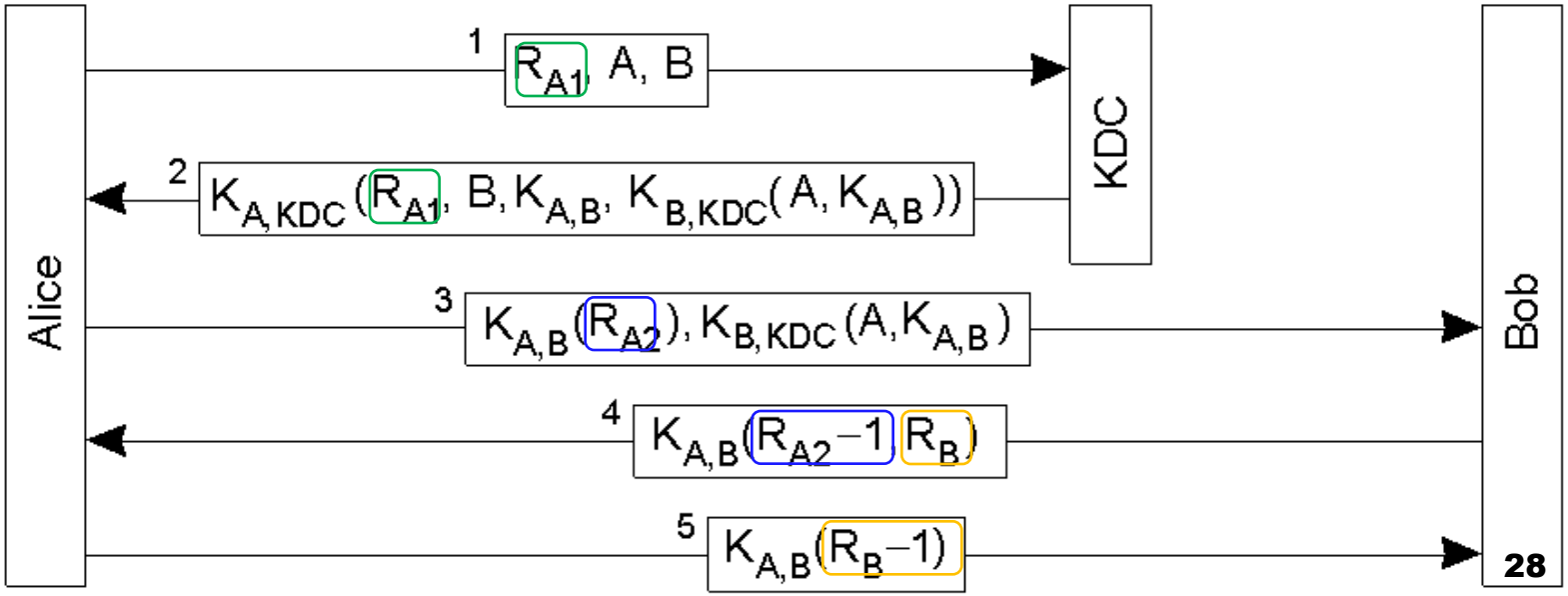
Authentication Using a Key Distribution Center

- **Problem:** With *N* subjects, we need to manage ***N*(*N*-1)/2** shared keys, each subject knowing *N*-1 keys
- **Solution:** Use a **trusted Key Distribution Center (KDC)** that generates a **shared session key** $K_{A,B}$ when necessary
  - Then we need only ***N*** keys: $K_{A,KDC}$ for Alice, $K_{B,KDC}$ for Bob, etc.



Authentication Using a Key Distribution Center

- **Needham-Schroeder authentication protocol** that uses:
  - **Shared session key** $K_{A,B}$
  - **Ticket** $K_{B,KDC}(A,K_{A,B})$ that enables Alice to tell Bob what the shared key is
  - **Nonce** R_{A1} for Alice with the KDC
  - **Nonce** R_{A2} for Alice with Bob
  - **Nonce** R_B for Bob with Alice

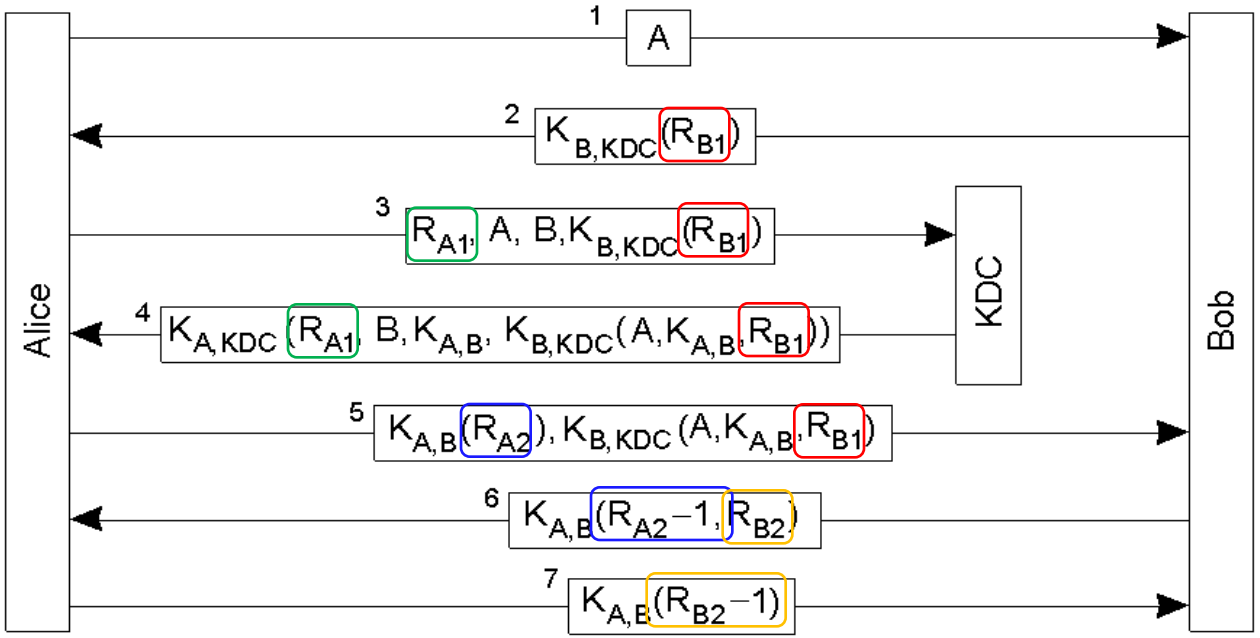


Needham-Schroeder: Subtleties

- **Question 1:** Why does the KDC put Bob into its reply message, and Alice into the ticket?
- **Question 2:** The message sent back to Alice by the KDC is encrypted with Alice's key. Is this necessary?
- **Security flaw:** Suppose Chuck discovers Alice's private key $K_{A,KDC}$ He can use that key anytime to impersonate Alice, even if Alice changes her private key at the KDC
- **Reasoning:** Once Chuck discovers Alice's private key $K_{A,KDC}$, 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 nonce from Bob first, and put the encrypted nonce in the ticket provided by the KDC => thus ensuring that every session is known at the KDC

Authentication Using a Key Distribution Center

- Fixing the security flaw in the Needham-Schroeder protocol to protect against malicious reuse of a previous session key
  - Bob communicates with Alice providing a **nonce R_{B1}** for Bob with Alice
  - The nonce R_{B1} is included in four messages, so Chuck can't replay an old message sequence



Authentication Using Public Key Cryptography

- Mutual authentication in a public key cryptosystem, using:
  - **Bob's public key K_B^+**
  - **Alice's public key K_A^+**
  - **Alice's nonce R_A**
  - **Bob's nonce R_B**
- Bob sends a **session key $K_{A,B}$** to Alice, encrypted with Alice's public key K_A^+
- Alice decrypts it with her private key K_A^-