Message Digest

- Can provide data integrity and non-repudation
  - Used to verify the authentication of a message
- Idea: compute a hash on the message and send it along with the message
- Receiver can apply the same hash function on the message and see whether the result coincides with the received hash

Hash Function

- A hash function h = H(m) takes a message m of arbitrary length as input and produces a fixed-length bit string h as output.
- A hash function is a one-way function, i.e., it is computationally infeasible to find the input m that corresponds to a known output h.
- The weak collision resistance property, i.e., given m and h = H(m), it is computationally infeasible to find another m' (m'≠ m), such that H(m) = H(m').
- The strong collision resistance property, i.e., when only given H, it is computationally infeasible to find two different m and m', such that H(m) = H(m').

Secure digest functions

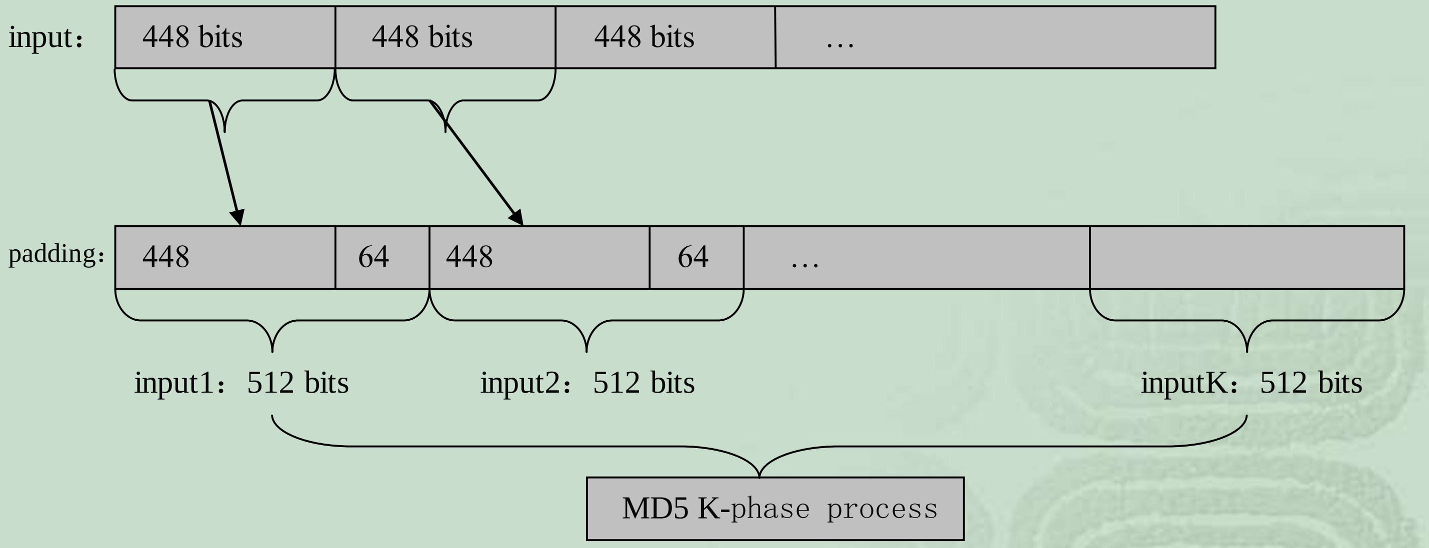
- h = H(M) is a secure digest function that has the following properties:
  - Given M, it is easy to compute h.
  - Given h, it is hard to compute M.
  - Given M, it is hard to find another message M’, such that H(M) = H(M’)

➔ *One-way hash functions*

- Examples: MD5, SHA-1
  - MD5 by Rivest
    - Uses 4 rounds each applying one of four non-linear functions to each of 16 32-bit segments of a 512 bit block of source. The result is a 128-bit digest.

MD5: Message-Digest algorithm 5

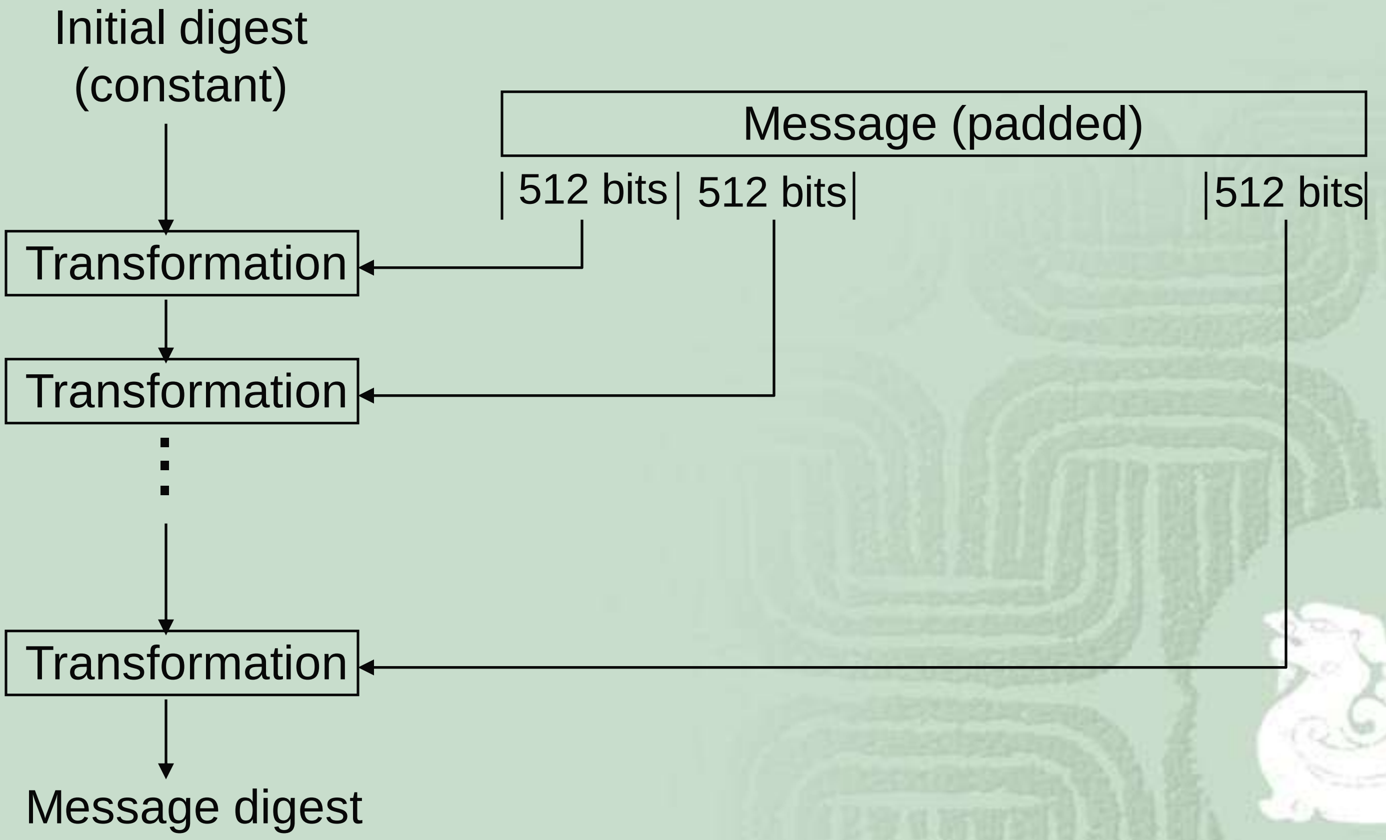
Initialization:



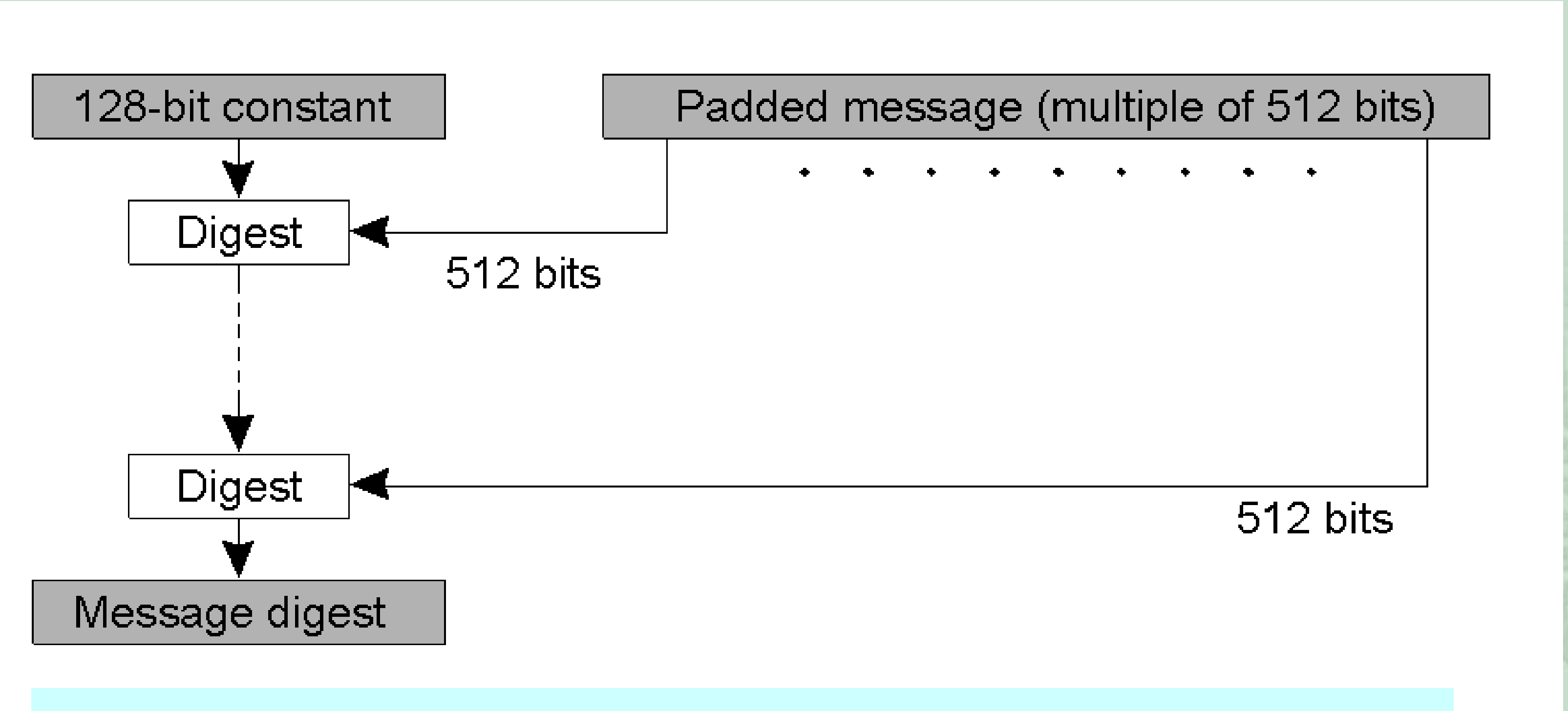
- MD5 is a hash function for computing a 128-bit, fixed-length message digest from an arbitrary length binary input.
- Initialization: dividing input into 448-bit blocks and then padding these blocks into 512-bit blocks.

Message Digest Operation

- Transformation contains complex operations



MD5: K-phase hashing



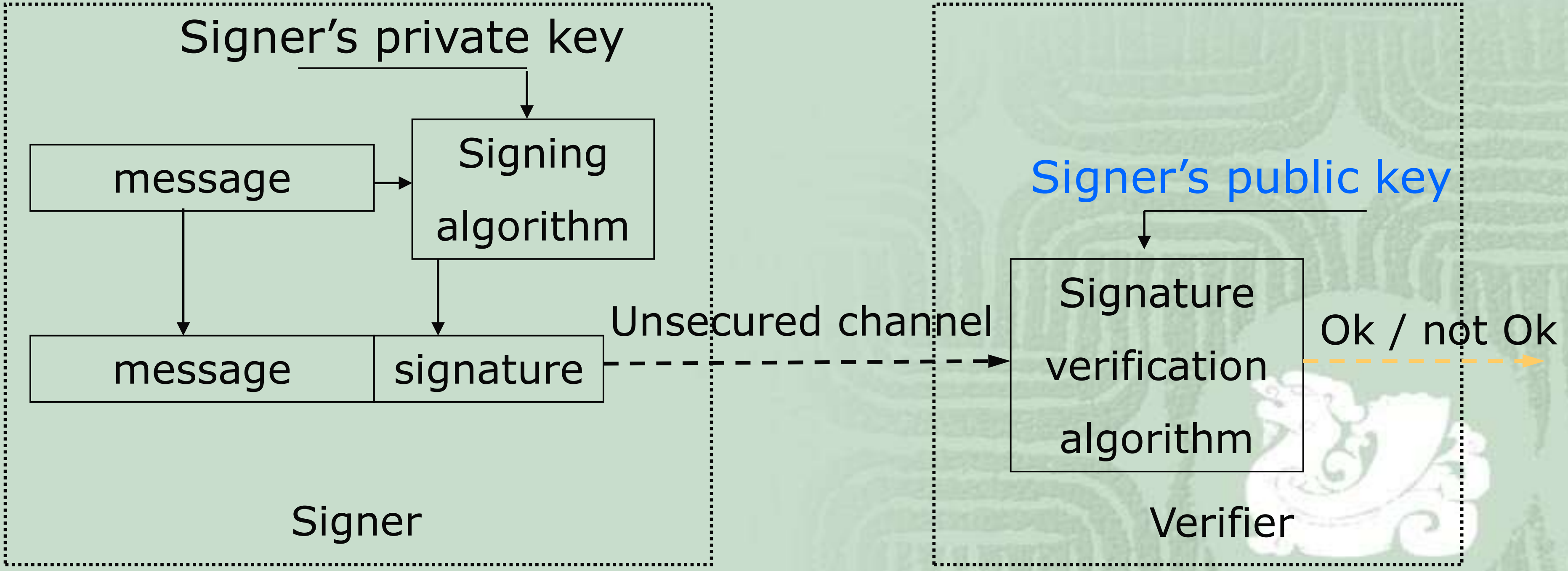
- K is the number of padded blocks
- Each phase consists four rounds of computations by using four different functions.
- Typical application of MD5 is Digital Signature.

Digital Signatures

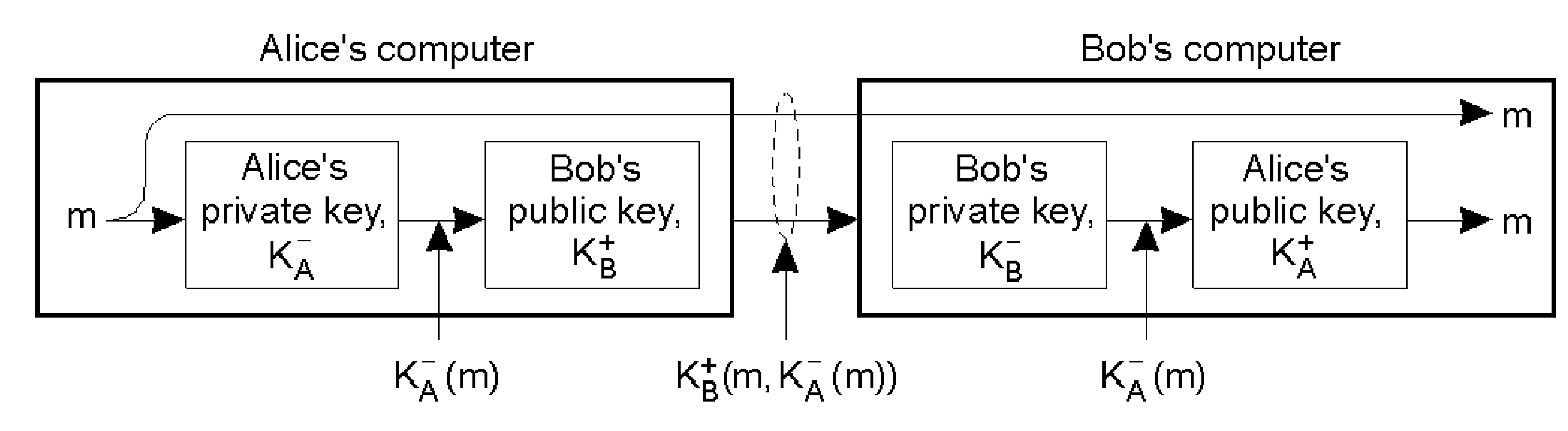
- A digital signature has the same authentication and legally binding functions as a handwritten signature.
- An electronic document or message M can be signed by an entity A by encrypting a copy of M in a key K_A and attaching it to a plain-text copy of M and A's identifier, such as $<M, A, E(M, K_A)>$.
- Once a signature is attached to a electronic document, it should be possible (1) any party that receives a copy of message to verify that the document was originally signed by the signatory, and (2) the signature can not be altered either in transmit or the receivers.

Digital Signatures Scheme

- Used to provide
  - Data integrity
  - Message authentication
  - Non-repudiation



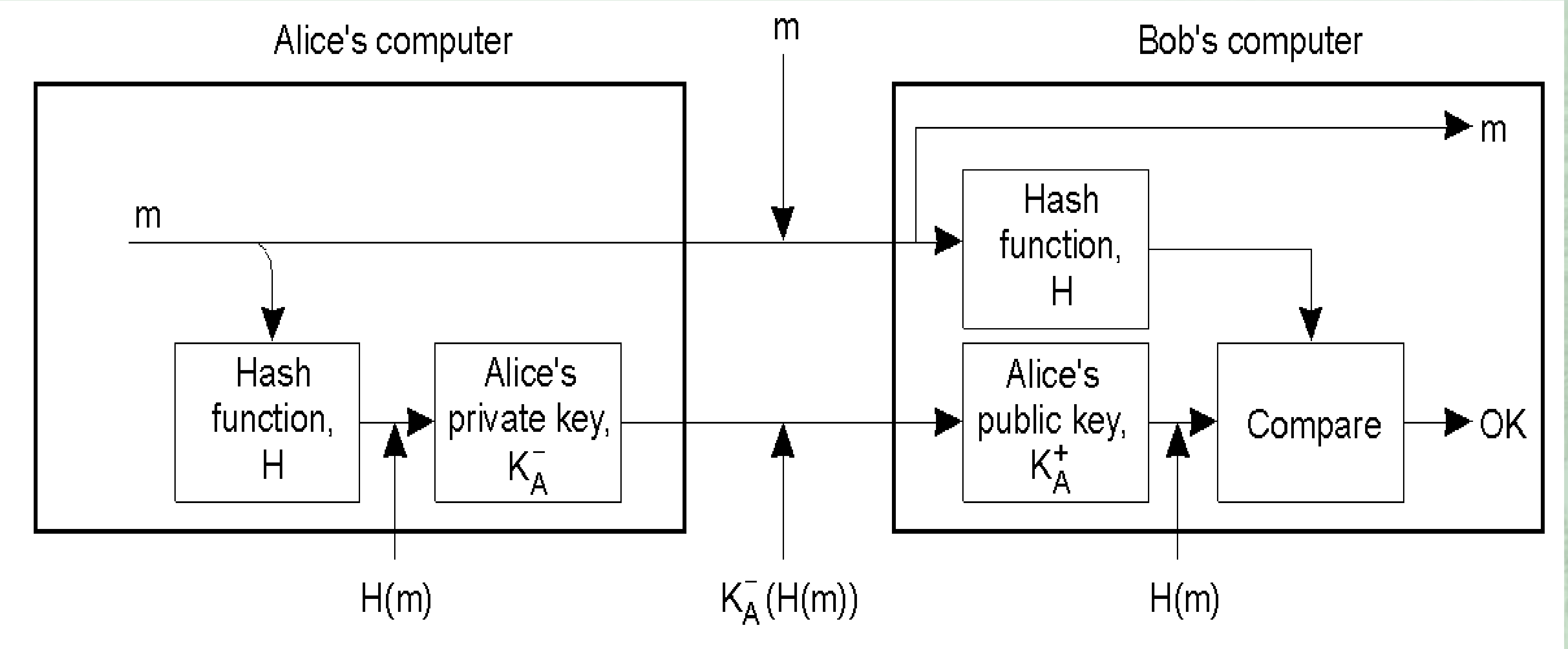
Public Key Digital Signatures (1)



- Digital signing a message using public-key cryptography.
- Problem: the validity of Alice’s signature holds only as long as Alice’s private key remains a secret and unchanged.
- Problem: the signature is too big.

Public Key Digital Signatures (2)

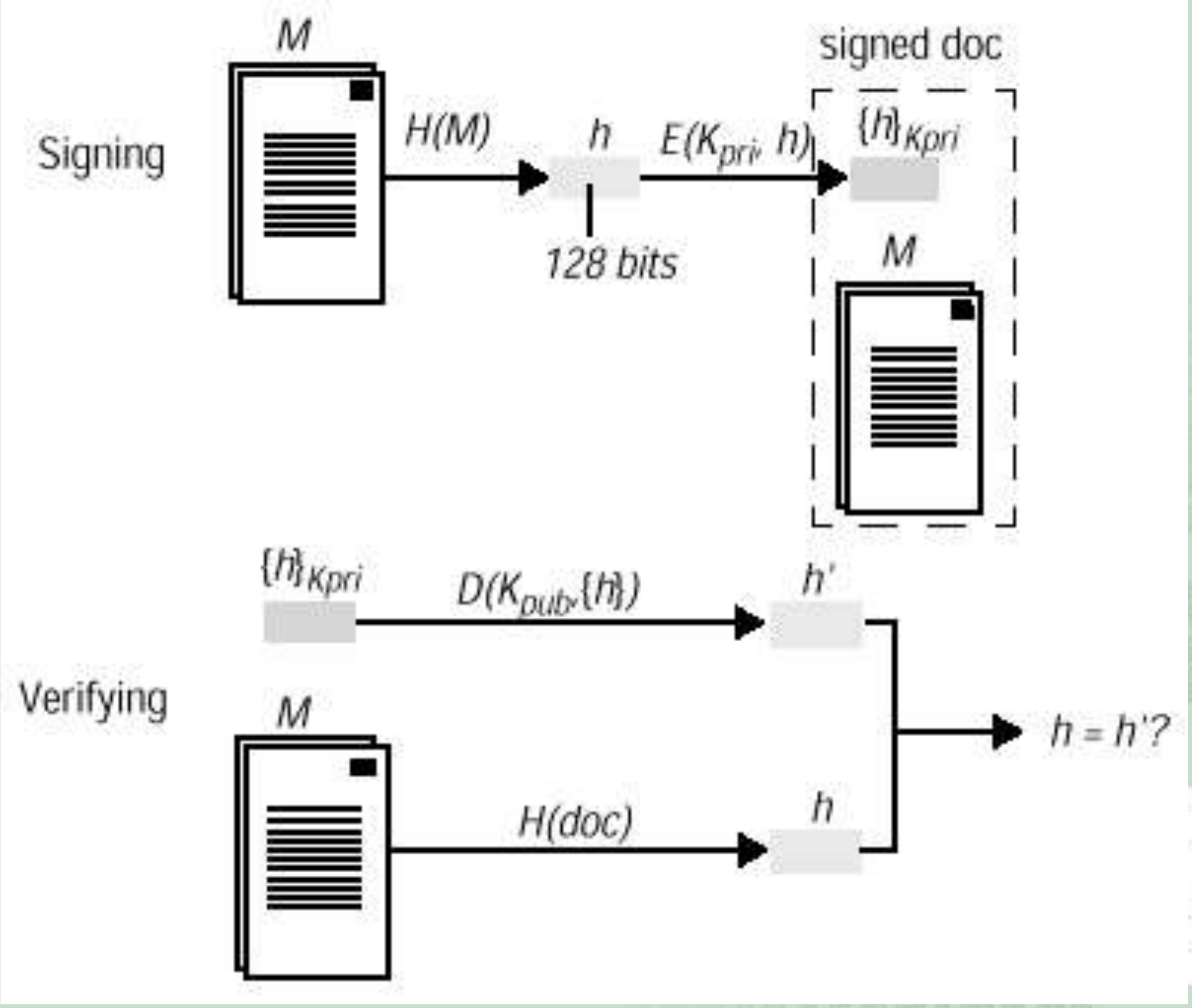
- In practice someone cannot alter the message without modifying the digest
  - Digest operation very hard to invert
- Encrypt digest with sender's private key
- K_A^-, K_A^+: private and public keys of A



Digital Signature Properties

- Integrity: an attacker cannot change the message without knowing A’s private key
- Confidentiality: if needed, encrypt message with B’s public key

Digital Signatures with Public Keys



Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

Digital Signatures with Secret Keys

- There is no reason why a secret-key encryption algorithm should not be used to encrypt a digital signature
  - To verify such signatures the key must be disclosed
  - The signer must arrange for the verifier to receive the secret key used for signing securely
  - At the time of signing the signer may not know the identities of the verifier ---- verification could be delegated to a trusted third party who holds secret keys for all signers --- adds complexity to the security model and requires secure communication with the trusted third party
- For all these reasons, the public-key method for generating and verifying signatures offers the most convenient solution in most situations

Digital Signatures with Secret Keys

- An exception arises when a secure channel is used to transmit unencrypted messages but there is a need to verify the authenticity of the messages.
  - Use the secure channel to establish a shared secret key using the hybrid method
  - Use this shared secret key to produce low-cost signatures --- *message authentication codes* (*MAC*)

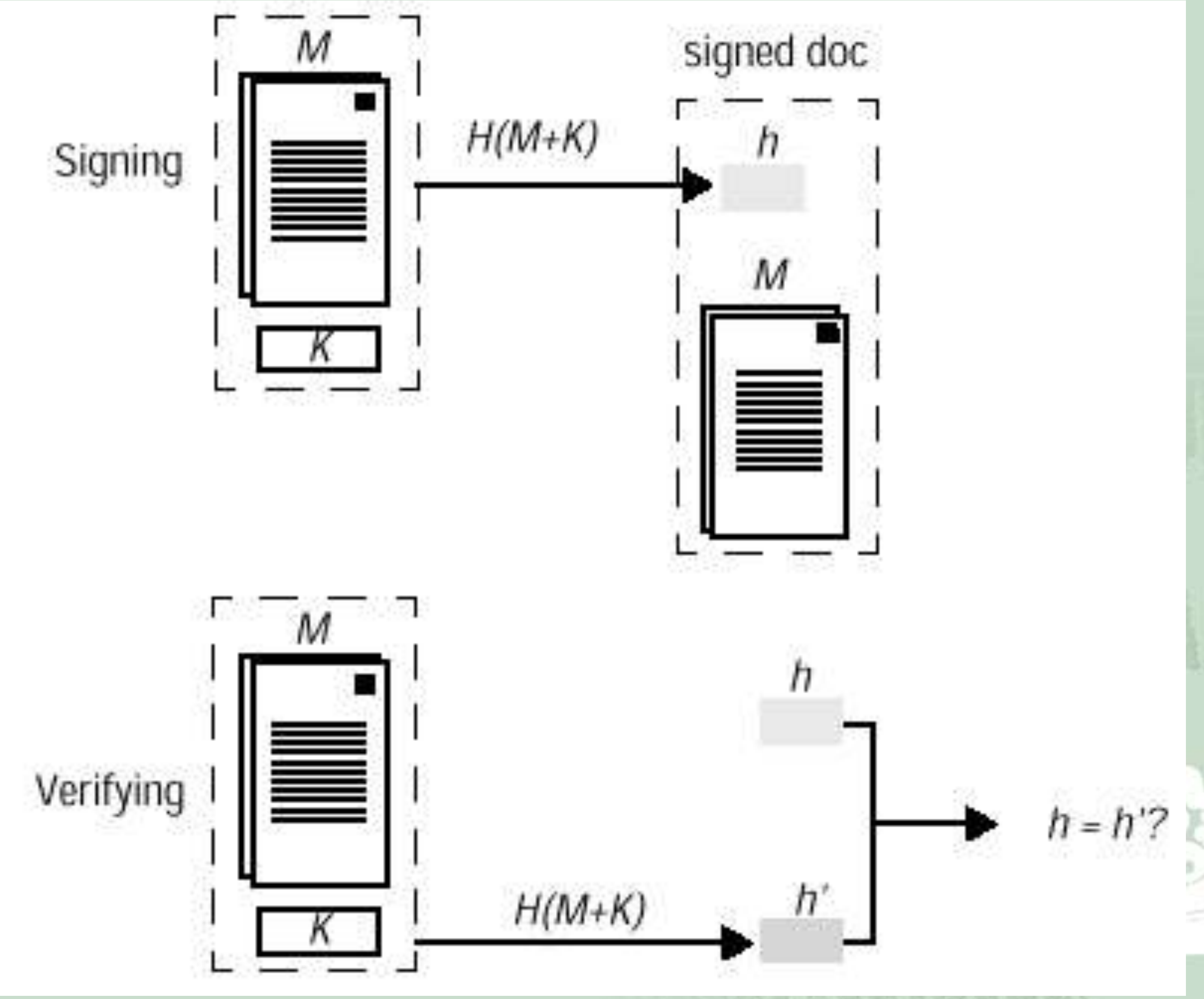
Digital Signatures with Secret Keys

- *A* generates a random key *k* for signing and distributes it using secure channels to one or more users who will need to authenticate messages received from *A*

- For any document *M* that *A* wishes to sign, *A* concatenates *M* with *K*, computes the digest $h = H(M + K)$, and sends the signed document <*M, h*> (the digest *h* is a *MAC*)

- The receiver, *B*, concatenates the secret key *K* with the received document *M* and compute the digest $h' = H(M + K)$. The signature is verified if $h = h'$.

Digital Signatures with Secret Keys



Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

Difference between MAC and digital signature

- To prove the validity of a MAC to a third party, you need to reveal the key
- If you can verify a MAC, you can also create it
- MAC does not allow a distinction to be made between the parties sharing the key
- Computing a MAC is (usually) much faster than computing a digital signature
  - Important for devices with low computing power

Digital Certificates

- A ***digital certificate*** is a document containing a statement (usually short) signed by a principal
  - It can be used to establish the authenticity of many types of statement.
- To make certificate useful, two things are needed
  - A standard format and representation so that certificate issuers and certificate users can successfully construct and interpret them
  - Agreement on the manner in which chains of certificates are constructed and in particular the notion of a *trusted authority*

Authentication: Certificates

Digital certificate:

A document, containing a statement signed by a principal

Scenario: Bob is a Bank, Alice is a customer

When a customer is contacting Bob, customers need to be sure that they are talking to “real” Bob, even if they have never contacted him before.

Bob needs to authenticate his customers before granting them access

Alice’s Bank Account Certificate

1. *Certificate type*:	Account number
2. *Name*:	Alice
3. *Account*:	6262626
4. *Certifying authority*:	Bob’s Bank
5. *Signature*:	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

Public-key certificate for Bob’s Bank

A third party, Carol, before accepting Alice’s account needs to verify the authenticity of Bob’s private key

For this a “Public-key” certificate of Bob’s bank is provided by a well-known and trusted third party Fred

In the Internet there are some trusted certifying authorities such as Verisign, CERN.

A Public Key Certificate of Bob's Bank

1. *Certificate type*:	Public key
2. *Name*:	Bob's Bank
3. *Public key*:	K_{Bpub}
4. *Certifying authority*:	Fred – The Bankers Federation
5. *Signature*:	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

Authentication

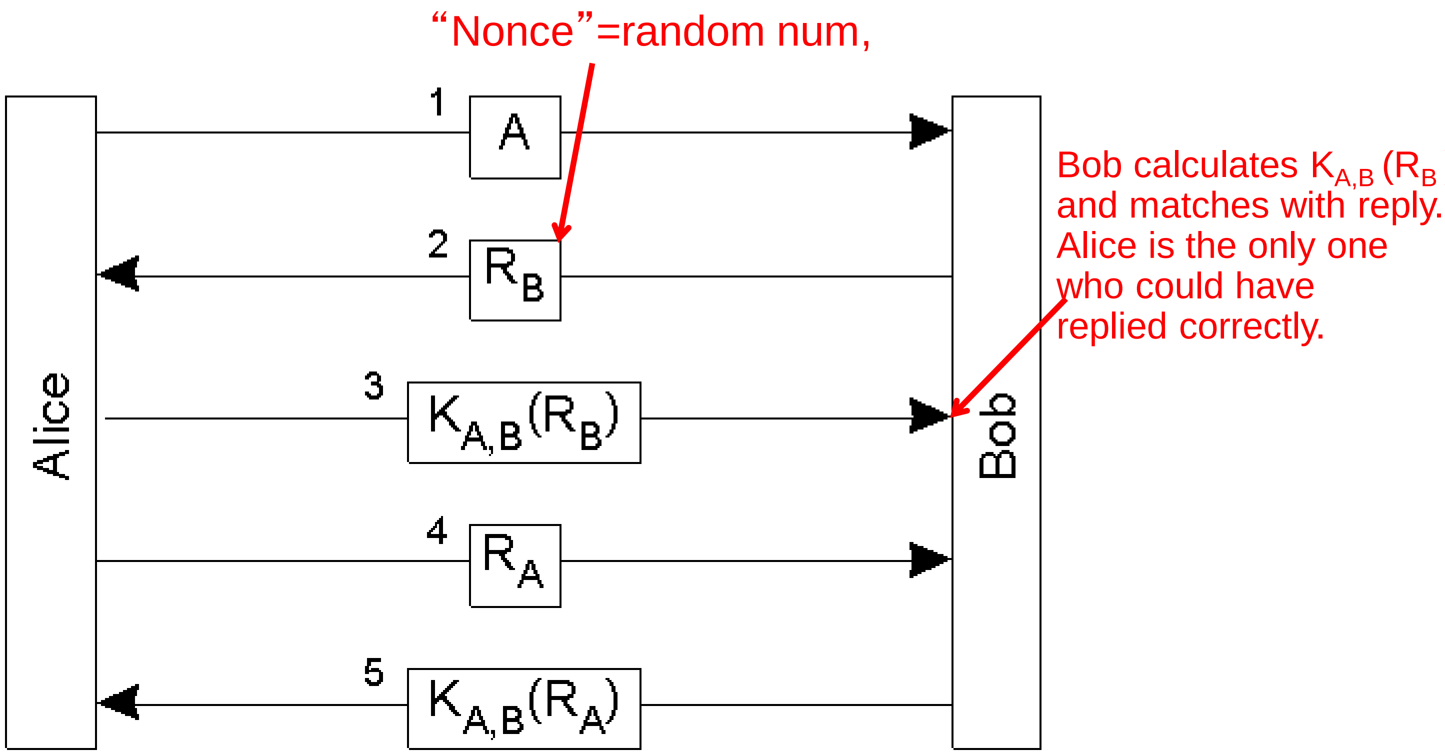
- Use of cryptography to have two principals verify each others' identities.
  - Direct authentication: the server uses a shared secret key to authenticate the client.
  - Indirect authentication: a trusted authentication server (third party) authenticates the client.
  - The authentication server knows keys of principals and generates temporary shared key (ticket) to an authenticated client. The ticket is used for messages in this session.
    - E.g., Verisign servers

Authentication

- Goal: Make sure that the sender an receiver are the ones they claim to be
- Solutions based on secret key cryptography (e.g., DES)
  - Three-way handshaking
  - Trusted third party (key distribution center, KDC)
- Solution based on public key cryptography (e.g., RSA)
  - Public key authentication

Direct Authentication: (Challenge-response protocol)

- **Authentication based on a shared secret key.**

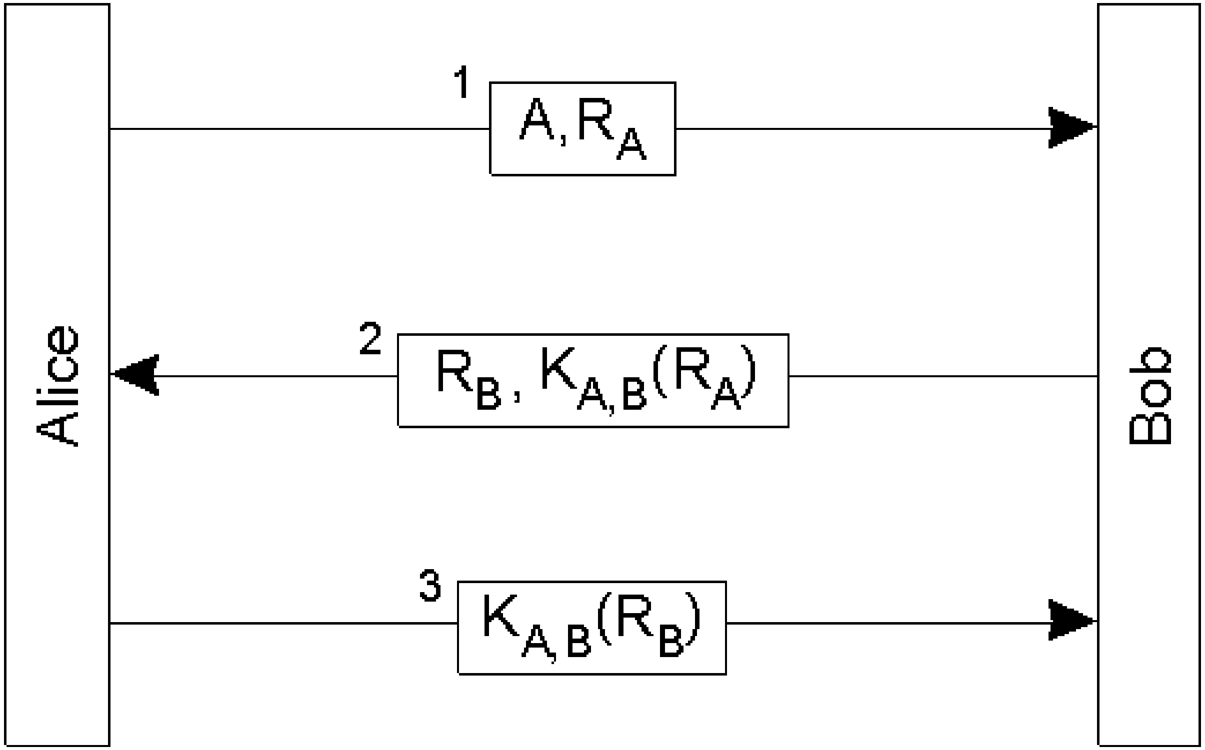


Authentication based on a shared secret key $K_{A,B}$

- R_A, R_B: random keys exchanged by A and B to verify identities

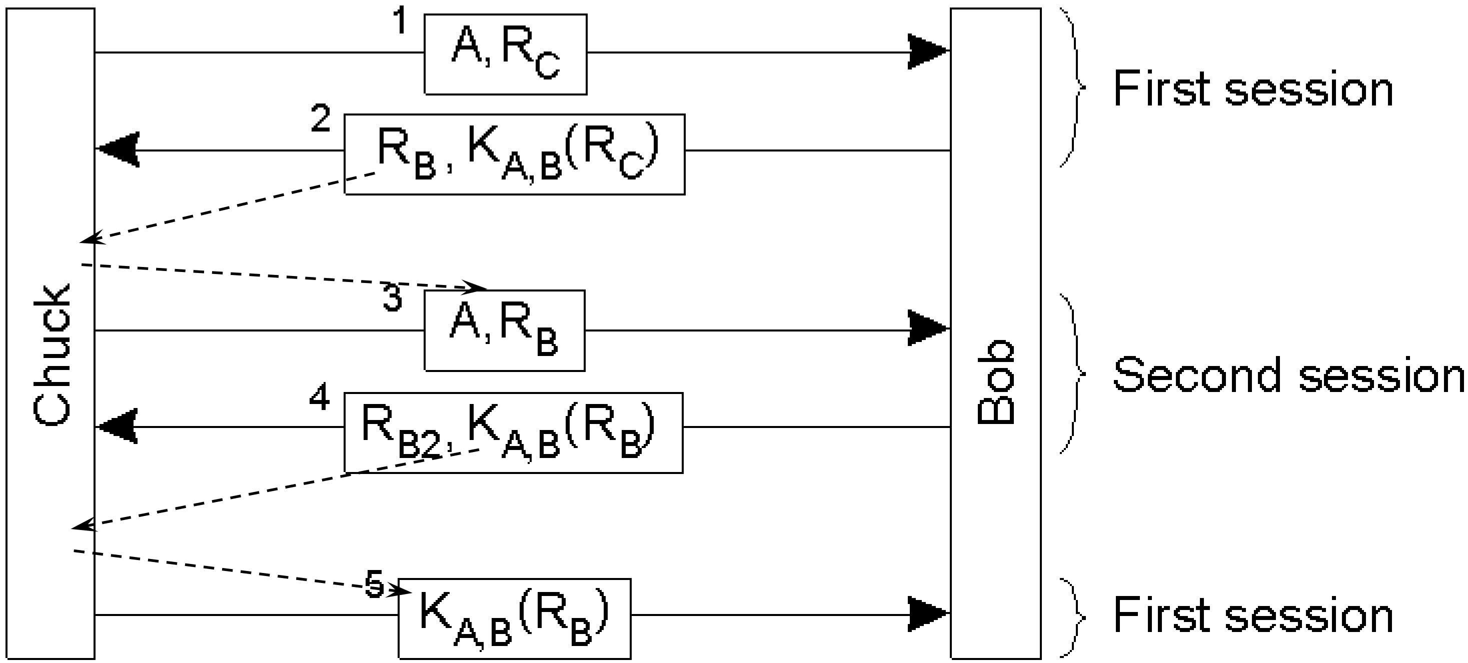
“Optimized” Direct Authentication

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



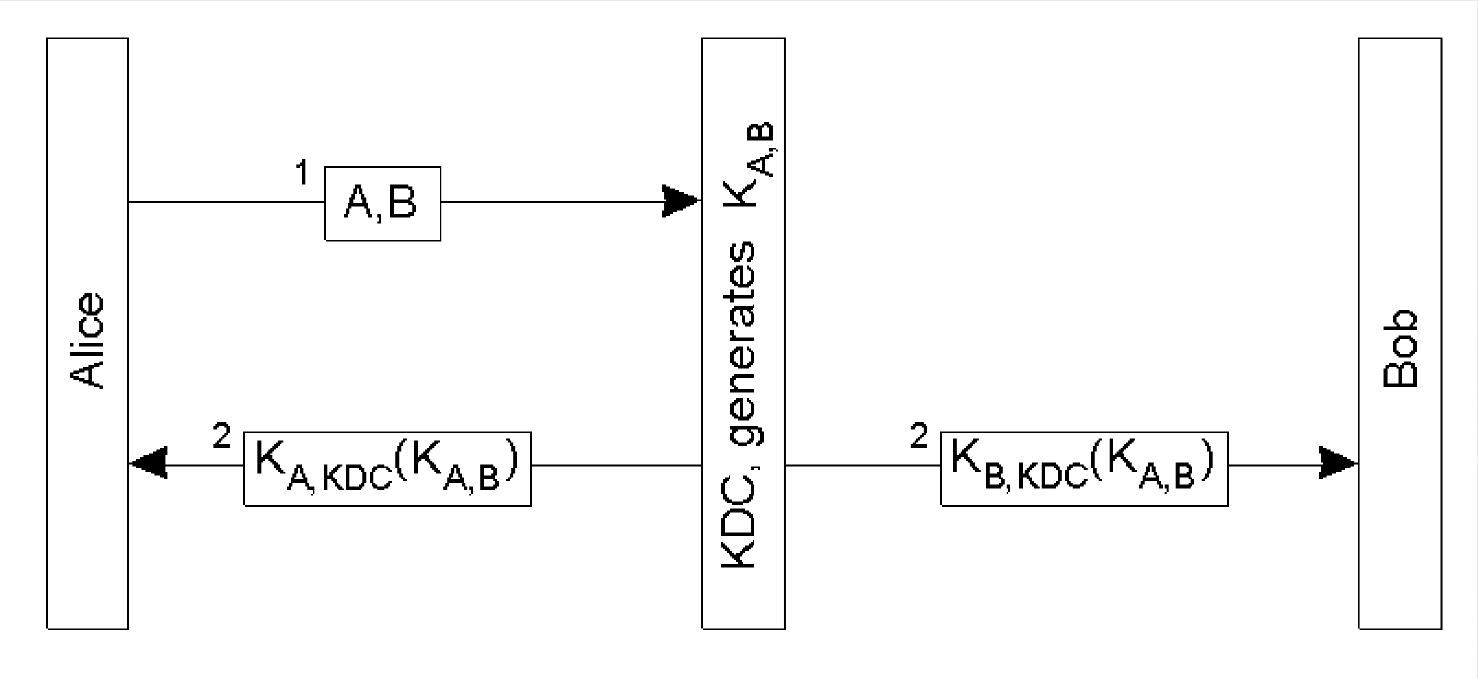
Replay/Reflection Attack (with shared keys)

Steps 1, 2, 5 -> Chuck is authenticated as Alice



The reflection attack: Bob gave away valuable information $K_{A,\ B}(R_B)$ without knowing for sure to whom he was giving it.

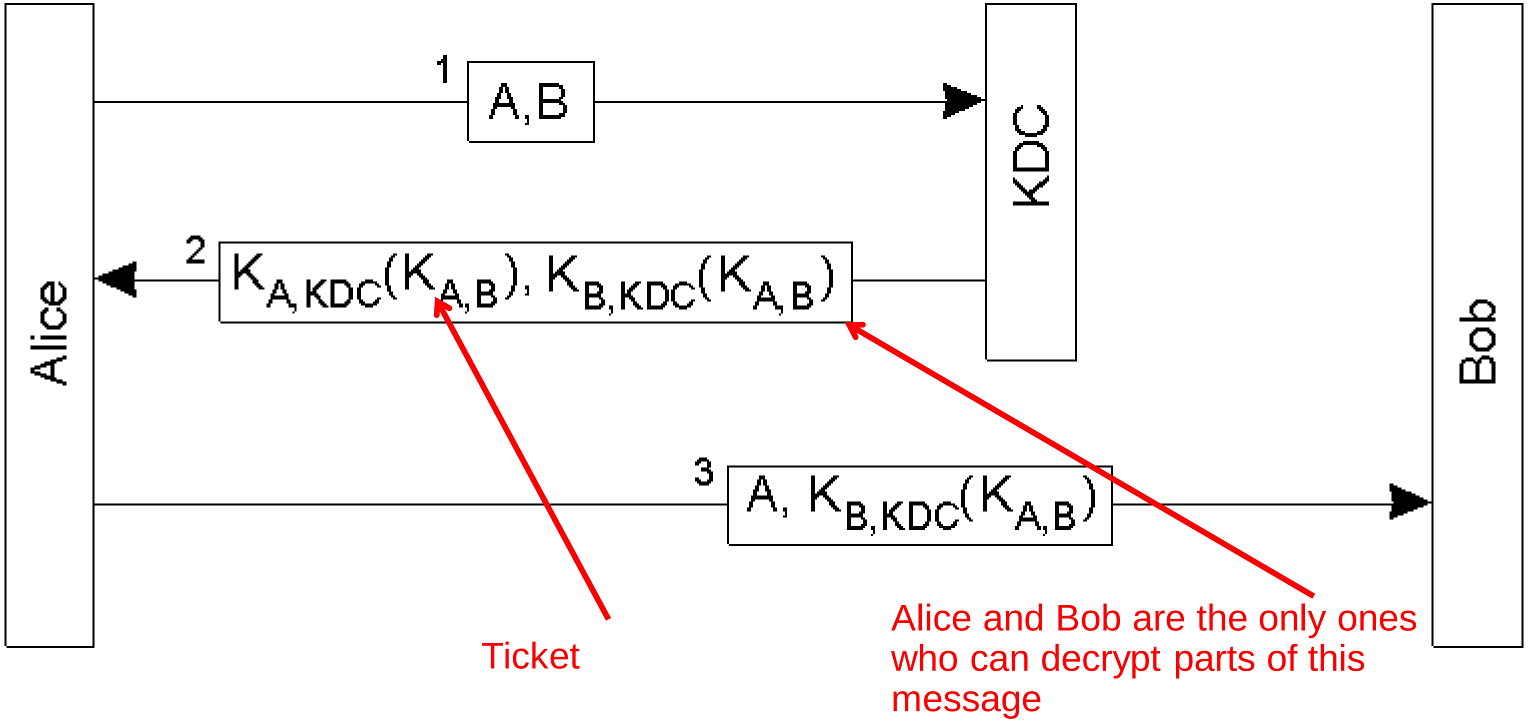
KDC based protocol (1)



- KDC shares a secret key with each of the clients.
- KDC hands out a key to both communication parties.
- Problem: A → B even before B got the key from KDC.

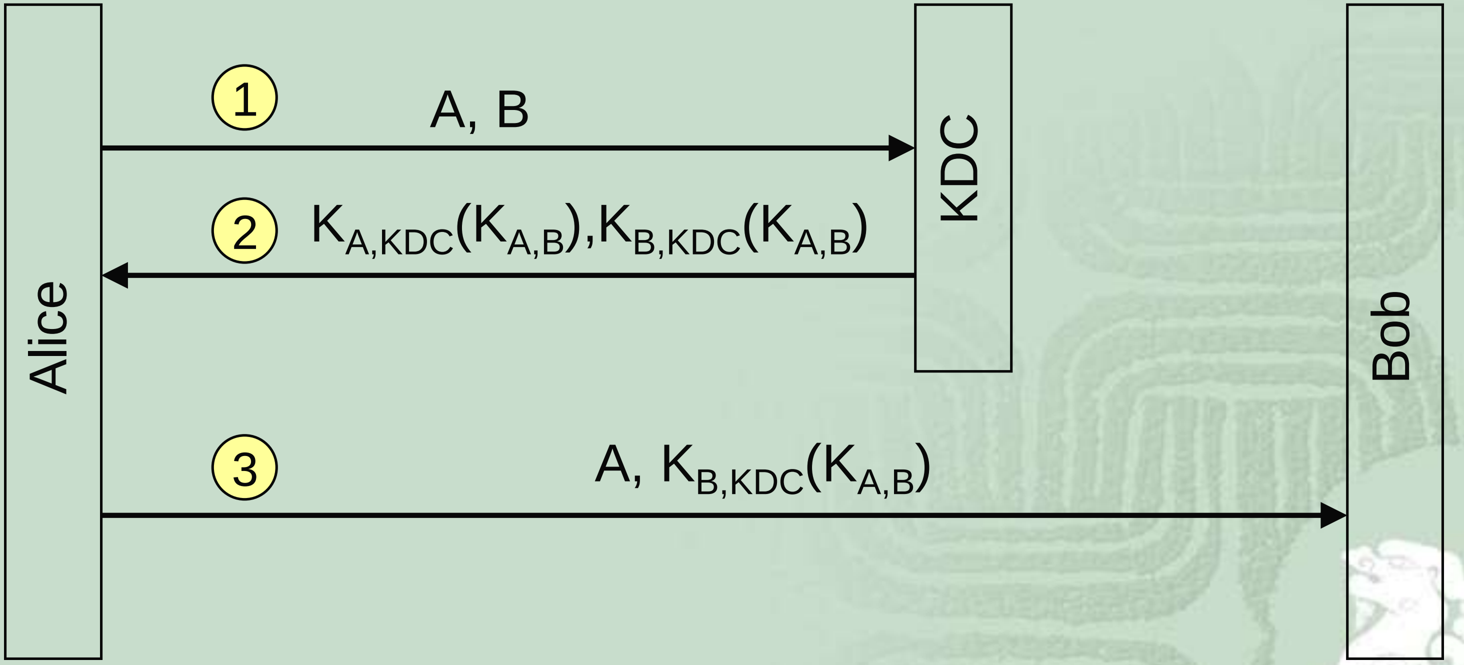
Indirect Authentication Using a Key Distribution Center (2)

- **Using a ticket and letting Alice set up a connection to Bob.**



Authentication using KDC (Ticket Based)

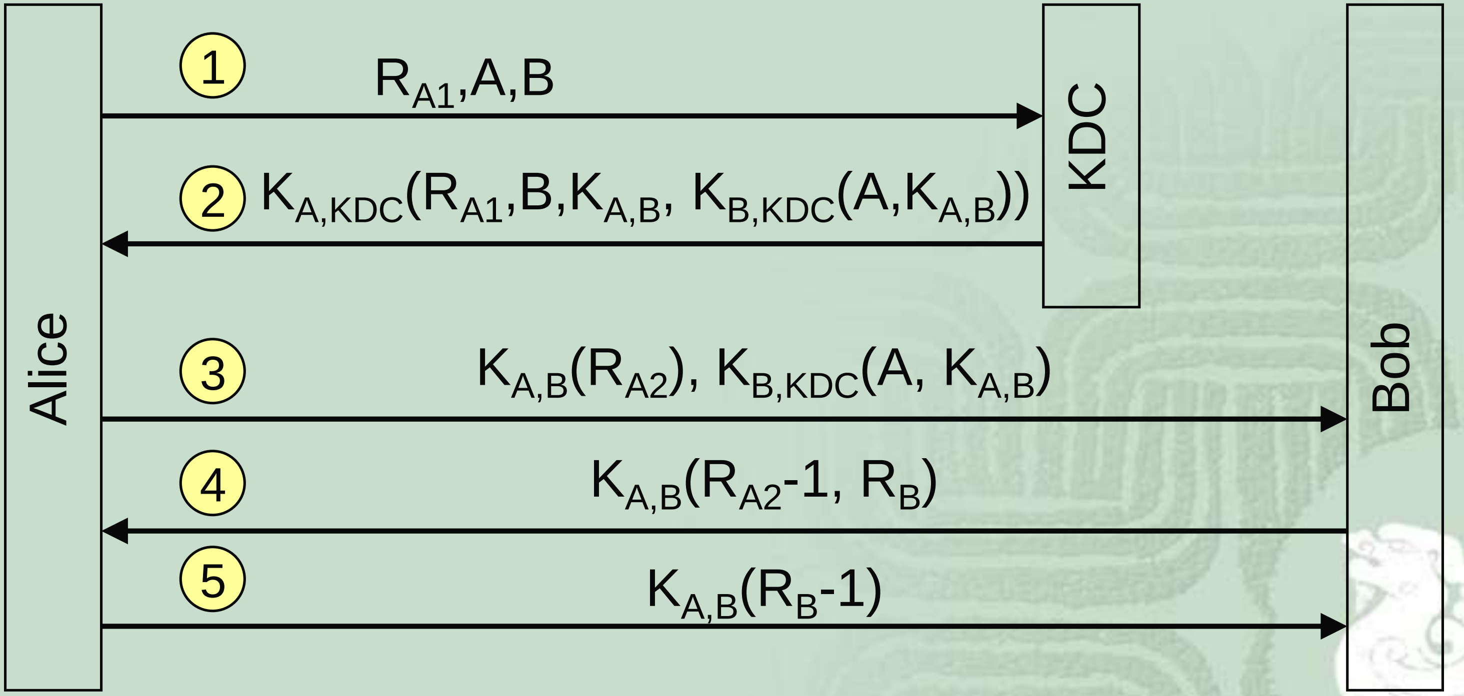
- No need for KDC to contact Bob



- Vulnerable to replay attacks if Chuck gets hold on $K_{B,KDC}^{old}$

Authentication using KDC (Needham-Schroeder Protocol)

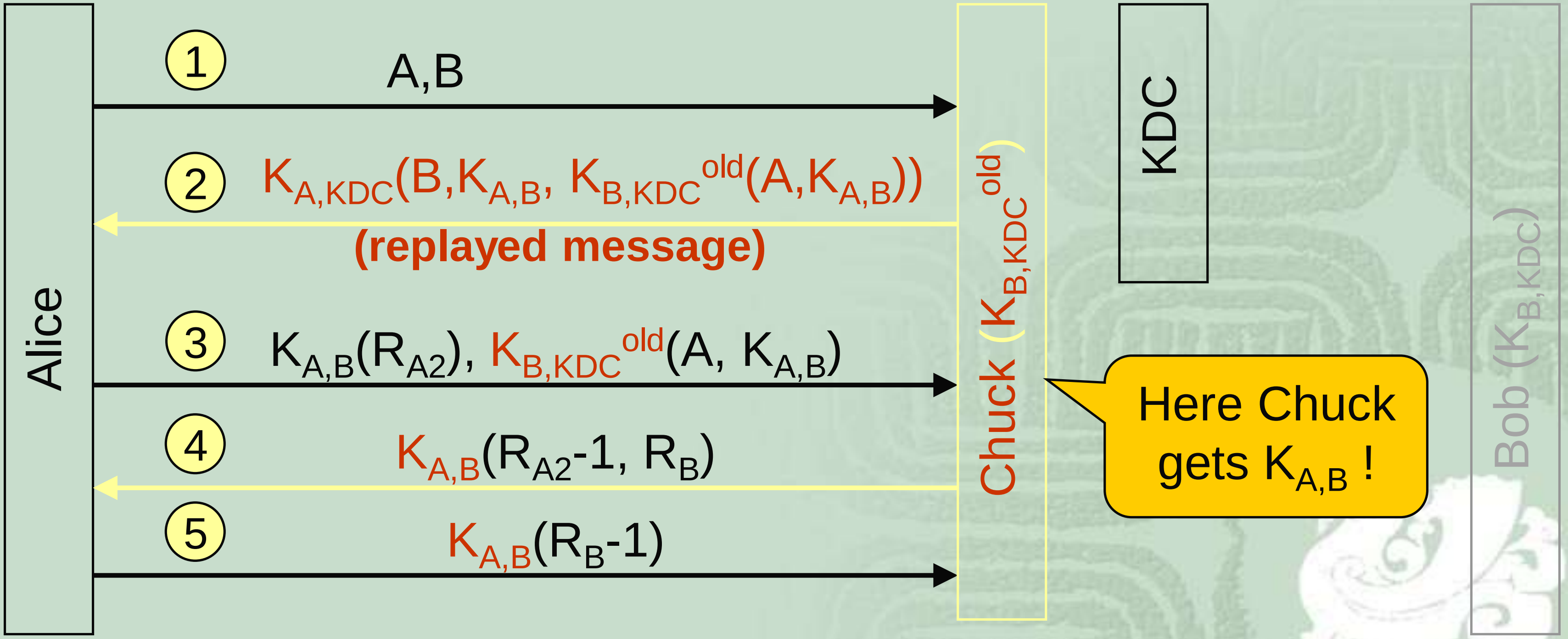
- Relate messages 1 and 2: use challenge response mechanism
- R_{A1}, R_{A2}, R_B: nonces
  - **Nonce**: random number used only once to **relate** two messages



- Vulnerable to replay attacks if Chuck gets hold on $K_{A,B}$

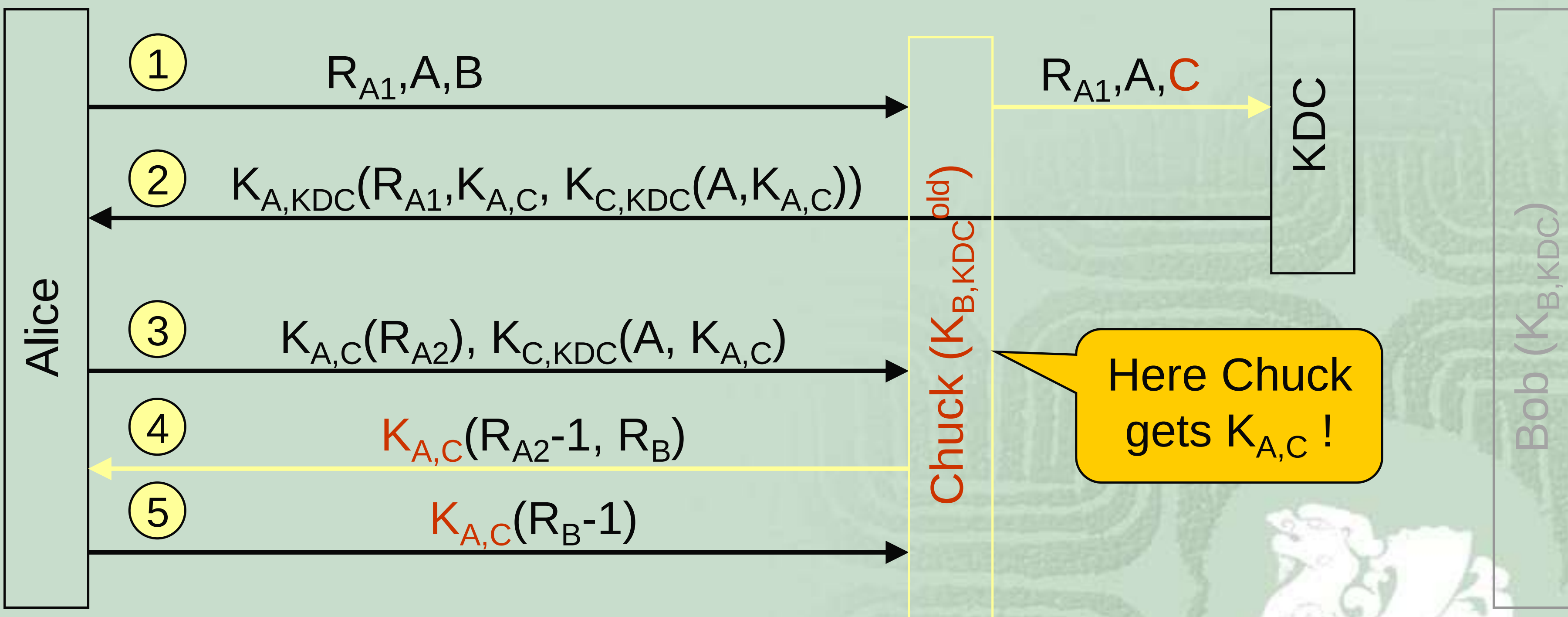
What if R_{A1} is Missing?

- Assume Chuck intercepted
  - $K_{A,KDC}(B,K_{A,B}, K_{B,KDC}^{old}(A,K_{A,B}))$
  - Knows $K_{B,KDC}^{old}$



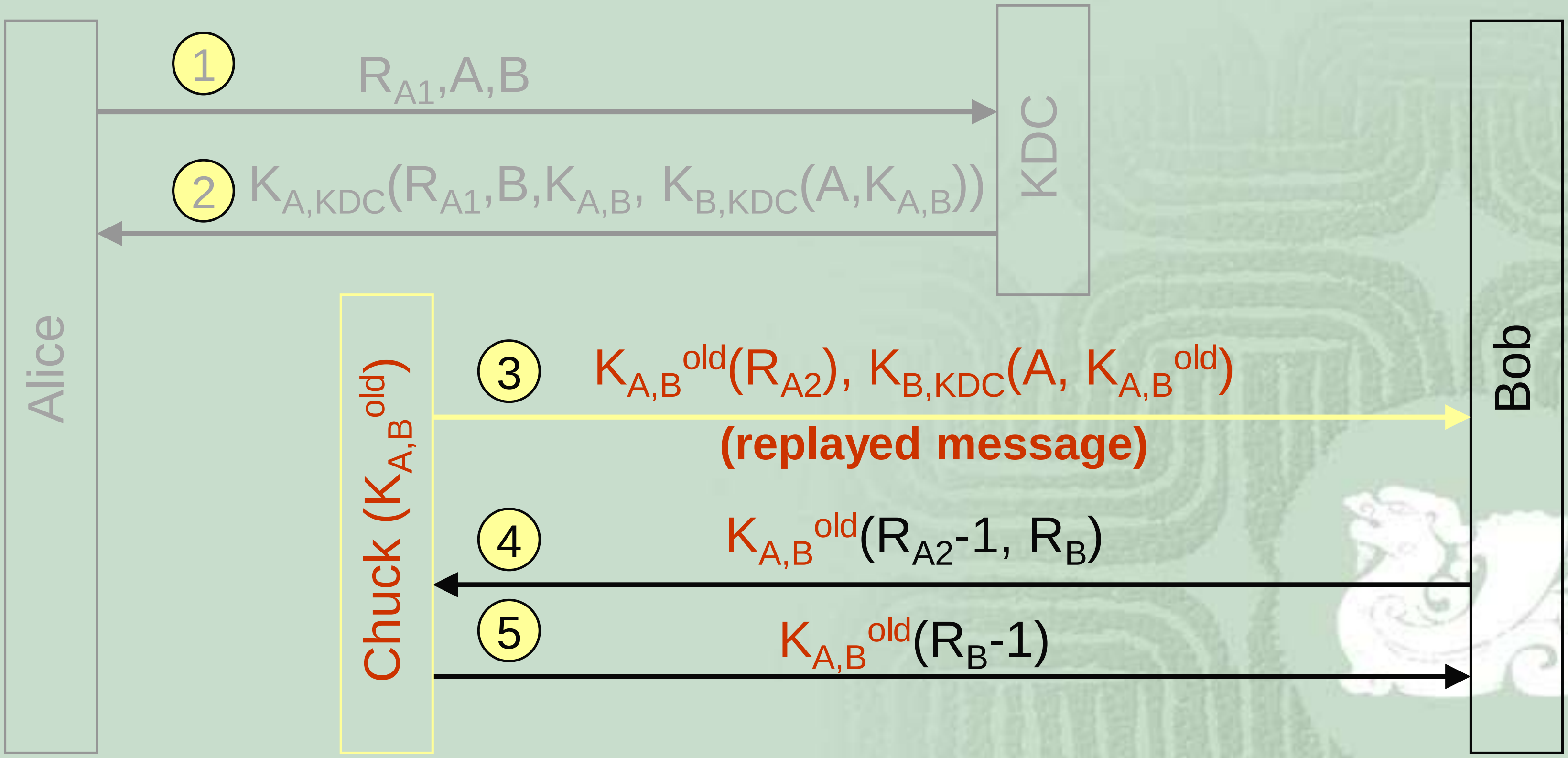
What if B is Missing from Message 2?

- Assume Chuck intercepts message 1



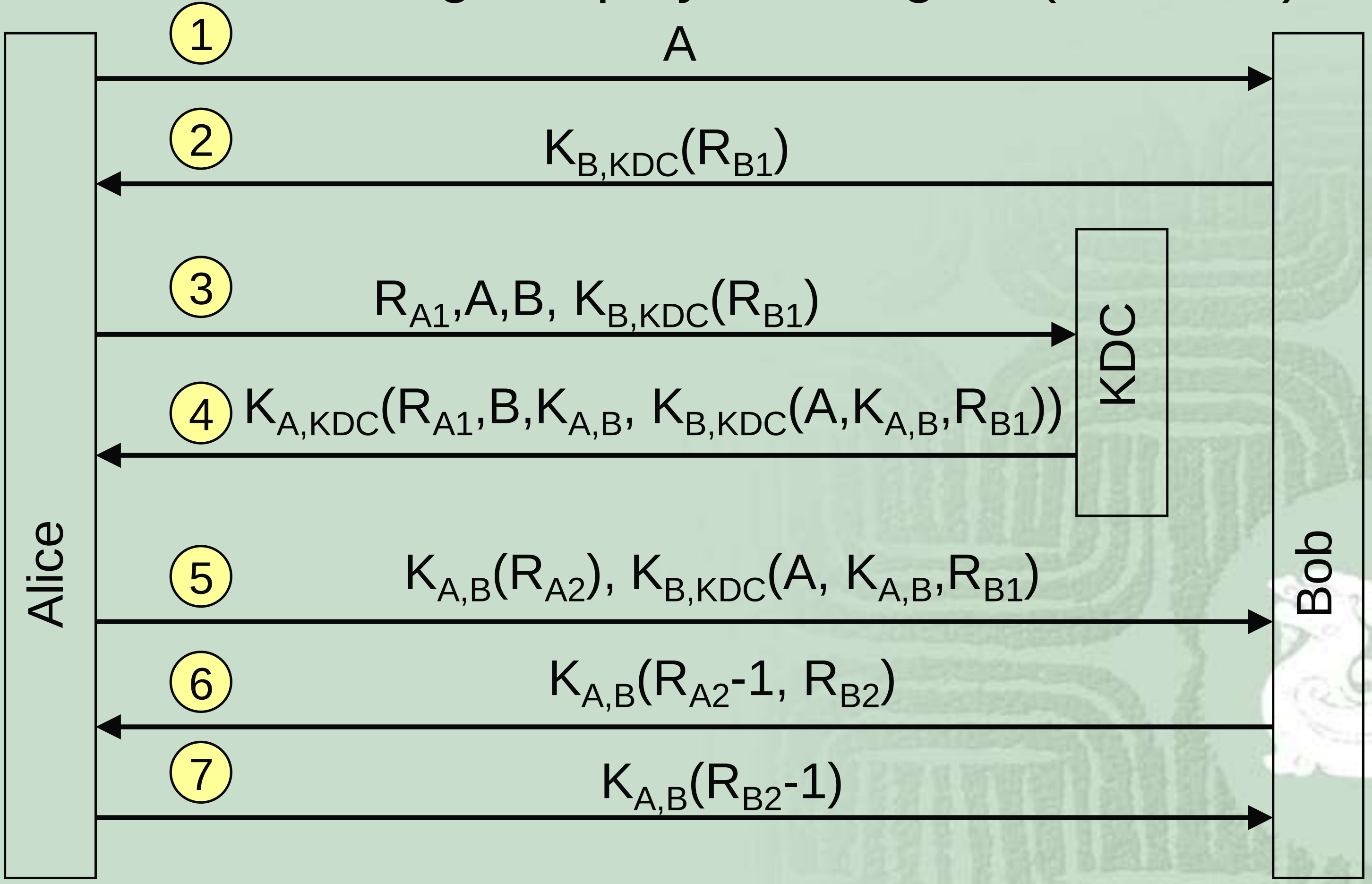
What if Chuck gets $K_{A,B}^{old}$?

- Assume Chuck intercepted
  - $K_{A,B}(R_{A2})$, $K_{B,KDC},(A,K_{A,B})$
  - Knows $K_{A,B}^{old}$

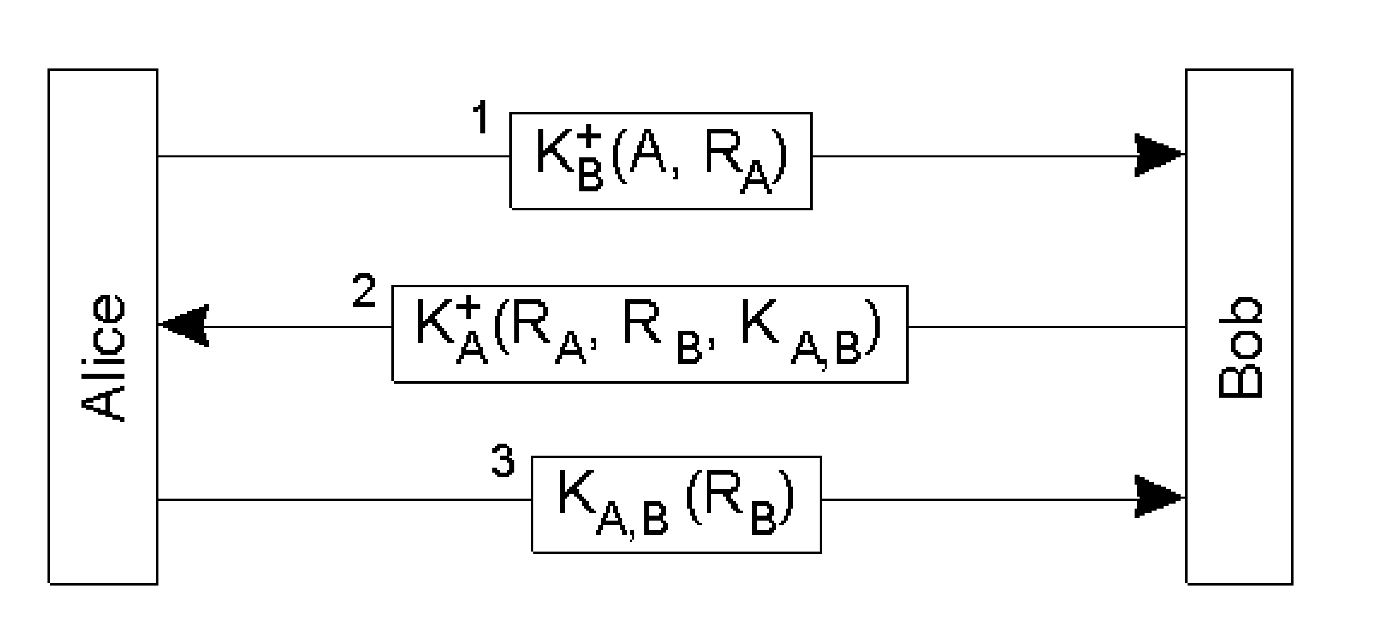


Defend Against leaking of $K_{A,B}$

- Message 5 (former 3) contains an encrypted nonce ($K_{B,KDC}(R_{B1})$) provided by Bob
- Chuck can no longer replay message 4 (former 3)



Public Key Authentication Protocol



- Mutual authentication in a public-key cryptosystem.
- It is important that Alice must trust that she got the right public key (as well as the most updated key) to Bob, and not the public key of someone impersonating Bob.

The Needham-Schroeder Authentication Protocol

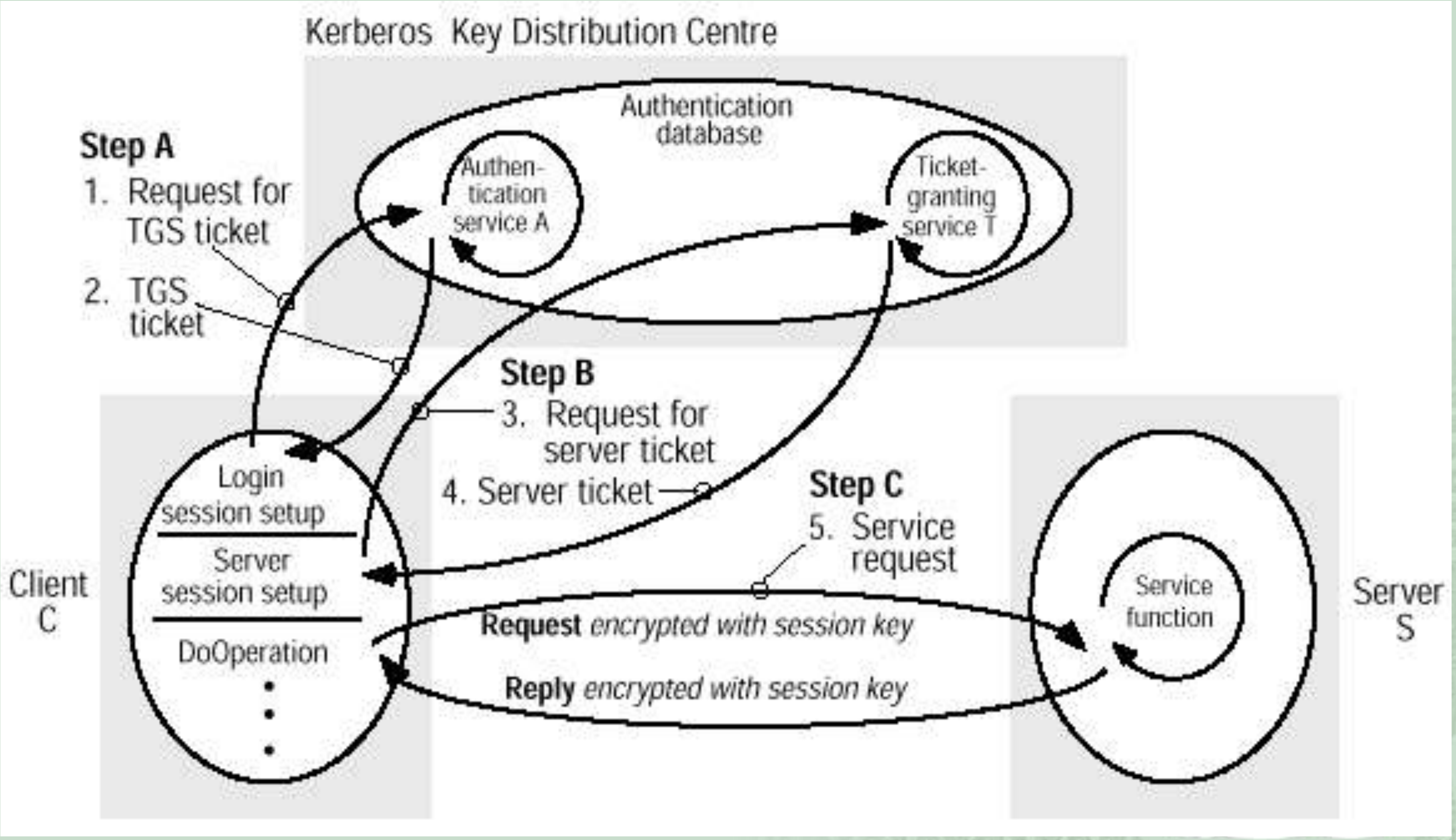
Header	*Message*	*Notes*
1. A → S:	A, B, N_A	A requests S to supply a key for communication with B.
2. S → A:	$\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$	S returns a message encrypted in A's secret key, containing a newly generated key K_{AB}, and a 'ticket' encrypted in B's secret key. The nonce N_A demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A's secret key.
3. A → B:	$\{K_{AB}, A\}_{K_B}$	A sends the 'ticket' to B.
4. B → A:	$\{N_B\}_{K_{AB}}$	B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B.
5. A → B:	$\{N_B - 1\}_{K_{AB}}$	A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B.

Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

Kerberos

- Developed at MIT
- For protecting networked services
- Based on the Needham-Schroeder protocol
- Current version: Kerberos Version 5
- Source code available
- Also used in OSF DCE, Windows 2000, ...

Kerberos Architecture



Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

The Kerberos Protocol

A. Obtain Kerberos session key and TGS ticket, once per login session

Header	*Message*	*Notes*
1. C → A: Request for TGS ticket	C, T, n	Client C requests the Kerberos authentication server A to supply a ticket for communication with the ticket-granting service T.
2. A → C: TGS session key and ticket	$\{K_{CT}, n\}_{K_C}, \{ticket(C,T)\}_{K_T}$ containing C, T, t_1, t_2, K_{CT}	A returns a message containing a ticket encrypted in its secret key and a session key for C to use with T. The inclusion of the nonce n encrypted in K_C shows that the message comes from the recipient of message 1, who must know K_C.

B. Obtain ticket for a server S, once per client-server session

3. C → T: Request ticket for service S	$\{auth(C)\}_{K_{CT}}, \{ticket(C,T)\}_{K_T}, S, n$	C requests the ticket-granting server T to supply a ticket for communication with another server S.
4. T → C: Service ticket	$\{K_{CS}, n\}_{K_{CT}}, \{ticket(C,S)\}_{K_S}$	T checks the ticket. If it is valid T generates a new random session key K_{CS} and returns it with a ticket for S (encrypted in the server's secret key K_S).

Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

The Kerberos Protocol (cont.)

C.	*Issue a server request with a ticket*	
5. $C \rightarrow S$: Service request	$\{auth(C)\}_{K_{CS}}$, $\{ticket(C,S)\}_{K_S}$, *request, n*	C sends the ticket to S with a newly generated authenticator for C and a request. The request would be encrypted in K_{CS} if secrecy of the data is required.

D.	*Authenticate server (optional)*	
6. $S \rightarrow C$: Server authentication	$\{n\}_{K_{CS}}$	(Optional): S sends the nonce to C, encrypted in K_{CS}.

$auth(C)$ contains C,t.

$ticket(C,S)$ contains C,S,t_1,t_2,K_{CS}.

Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

The Secure Sockets Layer (SSL)

- Originated by Netscape, now a nonproprietary standard (SSLv3)
- Provides secure end-to-end communications
- Operates between TCP/IP (or any other reliable transport protocol) and the application
- Built into most browsers and servers

Internet Security Protocols: SSL

An extended version of SSL has been adopted as Internet standard, Transport Layer security (TSL) [RFC 2246]

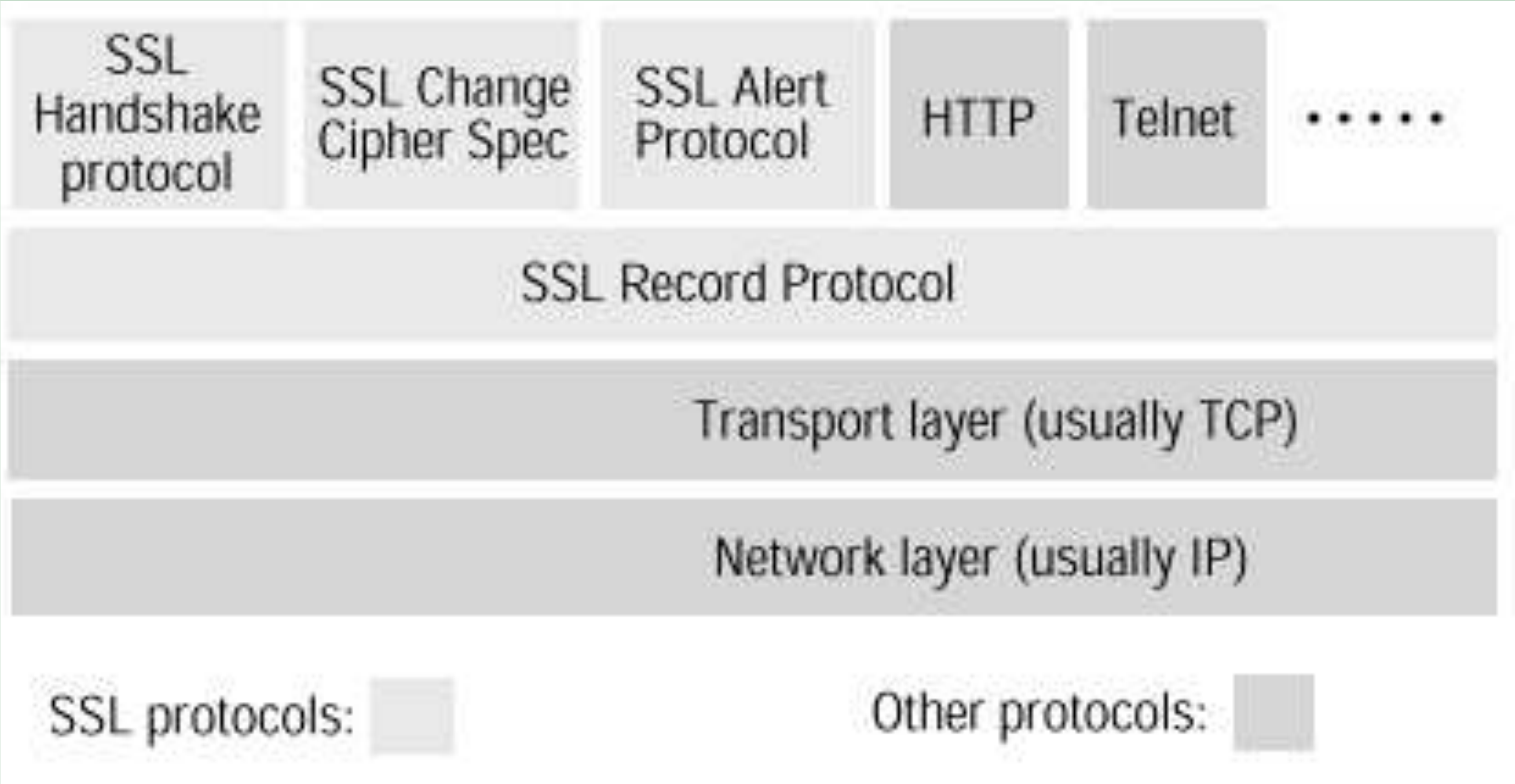
SSL features:

Negotiable encryption and authentication algorithms

- different client can use different protocols
- set up during initial connection establishment

Bootstrapped security communication

The SSL Protocol Stack



Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

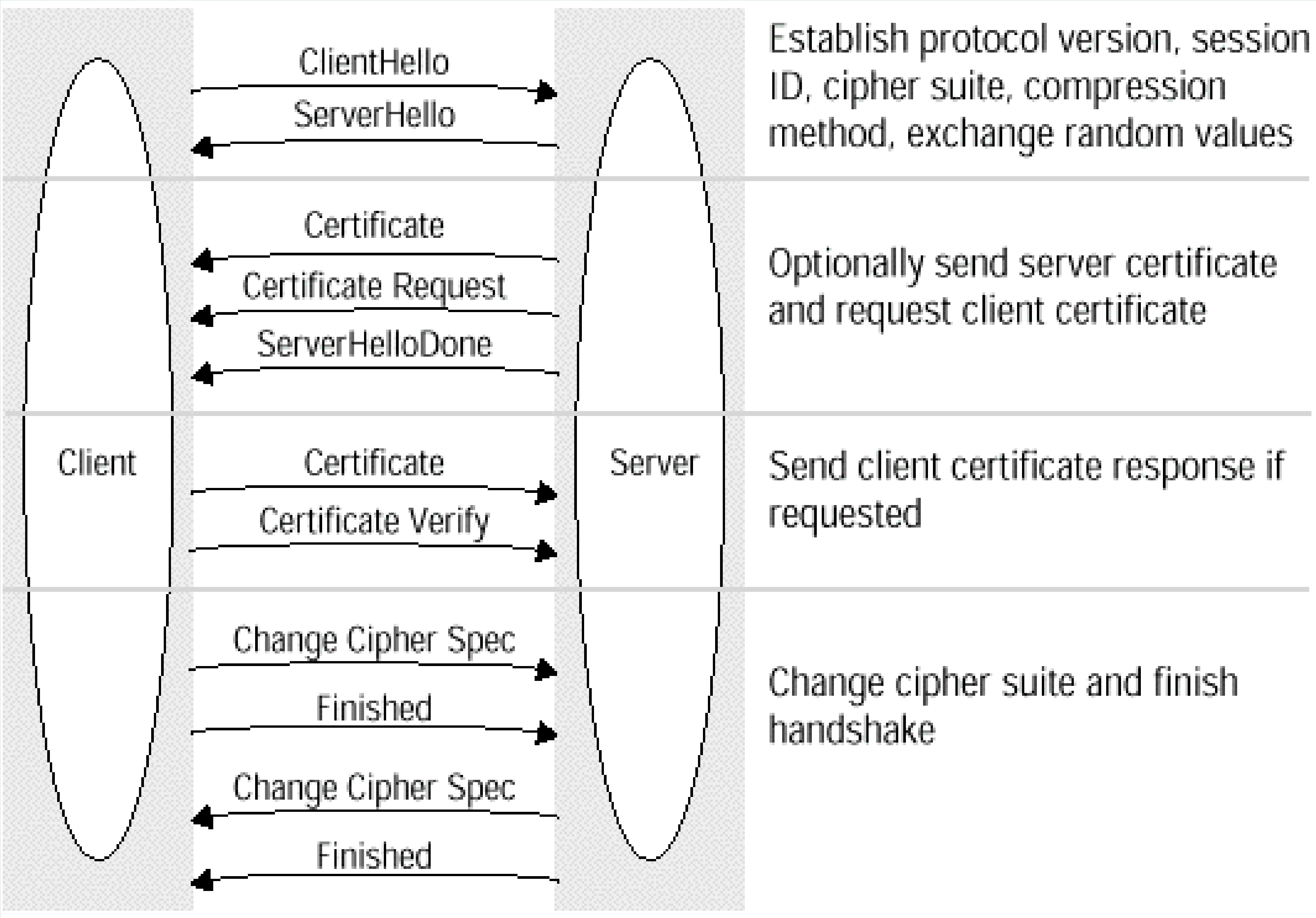
How SSL Works

- Sessions between a client and a server are established by the Handshake Protocol
- A session defines a set of security parameters, including peer certificate, cipher spec, and master secret
- Multiple connections can be established within a session, each defining further security parameters such as keys for encryption and authentication
- Security parameters dictate how application data are processed by the SSL Record Protocol into TCP segments

Security Functions of SSL

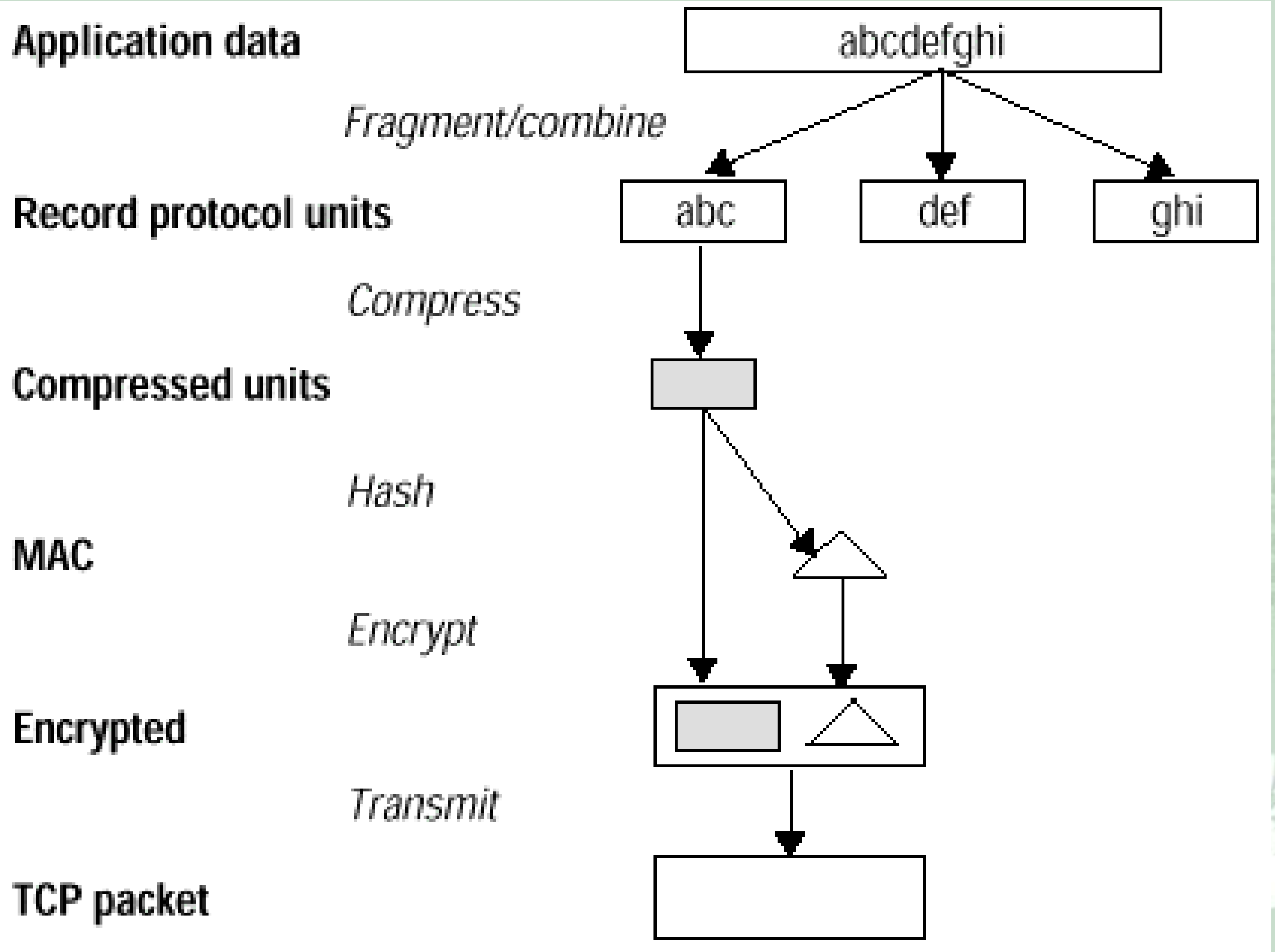
- Confidentiality: using one of DES, Triple DES, IDEA, RC2, RC4, …
- Integrity: using MAC with MD5 or SHA-1
- Authentication: using X.509v3 digital certificates

The SSL Handshake Protocol



Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

The SSL Record Protocol



Source: G. Coulouris et al., *Distributed Systems: Concepts and Design, Third Edition.*

Access Control

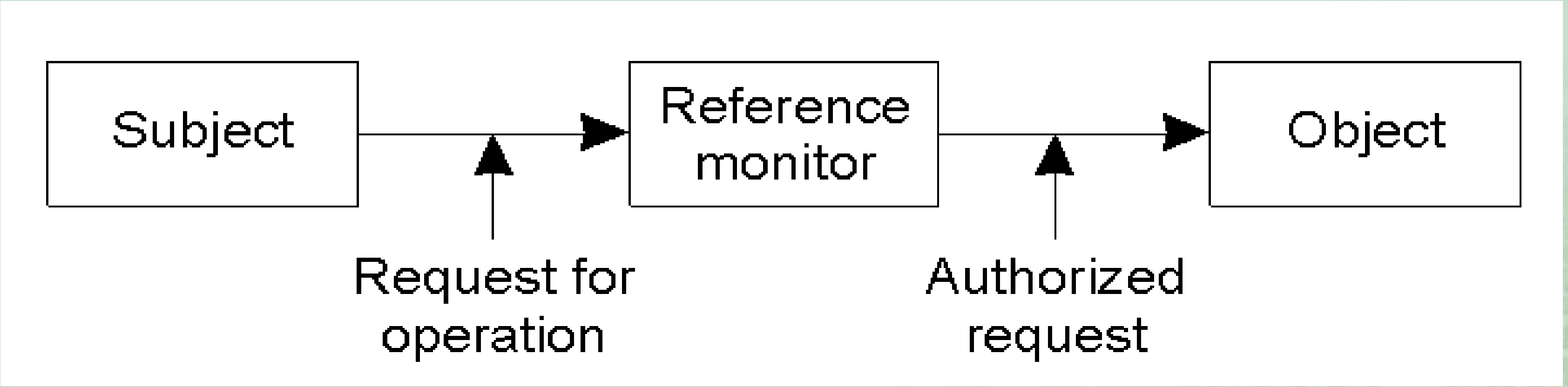
- A request from a client can be carried out only if the client has sufficient access rights for that requested operation.
- Verifying access rights is called access control, whereas authorization is about granting access rights.
- Many access control models:

  Access Control Matrix

  Access Control List (Capability List)

  Firewalls

General Issues in Access Control



General model of controlling access to objects

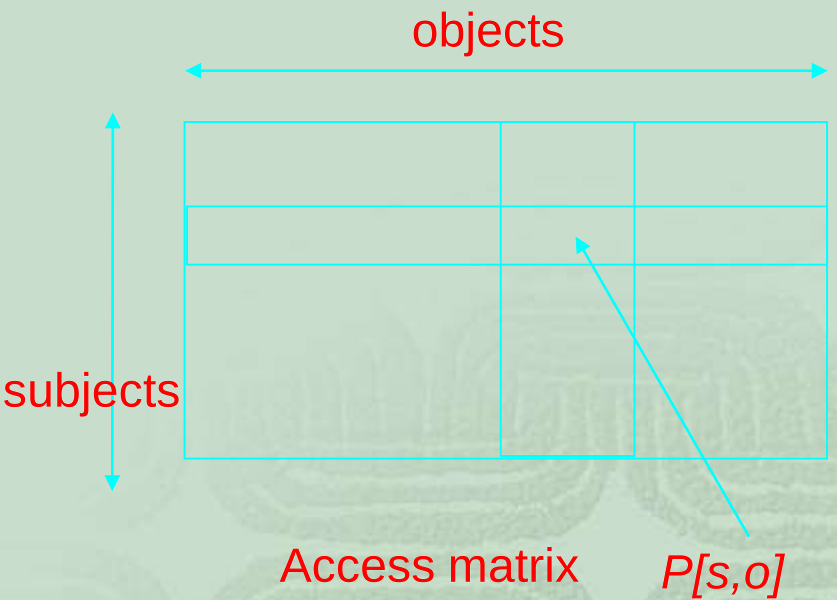
Theoretical model is based on Lampson's work on Access Control Matrix

Access Control Matrix

- Theoretical model:
  - Current objects *O*: finite set of entities to which access is to be controlled. Ex. Files
  - Current subjects *S:* finite set of entities that access current object. Ex. Processes
  - Generic rights, $R = \{r_1, r_2, \ldots, r_m\}$ give various rights that subjects have over objects. Ex. *r-w-x* in UNIX
- Protection state of a system
  - Protection state = (S, O, P), where P is a matrix, known as Access Control Matrix with subjects in the row and objects in the column and entries are the access rights

Access control matrix

- $P[s,o] \in R$, and denotes the access rights which subject s has on object o.
- Enforcing a security policy:
  - s requests an access α to o
  - protection system presents (s,α,o) to the monitor of o
  - The monitor looks into the access rights of s to o. If $\alpha \in P[s,o]$, then the access is permitted else denied



	o1	o2	s1	s2	s3
s1	*read,write*	*own,delete*	*own*	*sendmail*	*recmail*
s2	*execute*	*copy*	*recmail*	*own*	*block,wkup*
s3	*own*	*read,write*	*sendmail*	*block,wkup*	*own*

Access Control Matrix

Sub/Obj	file 1	file 2	file 3	file 4
user 1	owner	R/W	Exec	owner
user 2	--	R	owner	R/W
user 3	Copy/R	owner	--	--

(a) Resource ACM

Sub/Obj	process 1	process 2	process 3
process 1	--	send	Unblock send
process 2	receive	--	receive
process 3	Block receive	send	--

(b) Process communication ACM

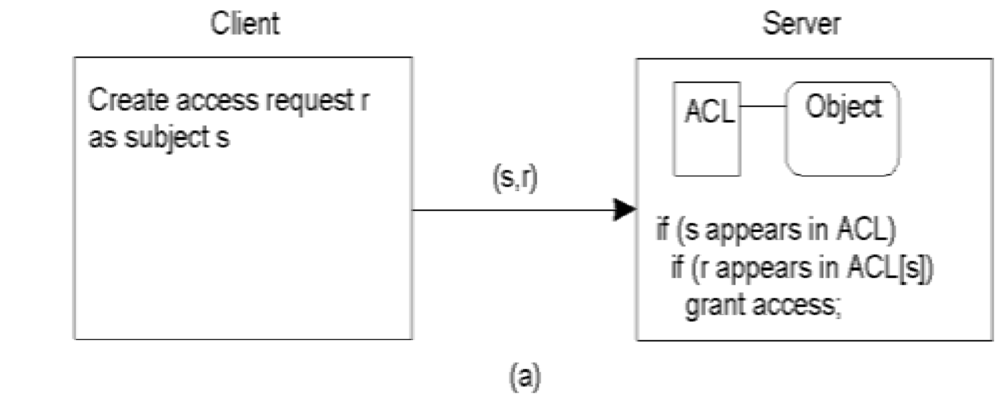
Sub/Obj	domain A	domain B	domain C
domain A	--	enter	--
domain B	--	--	enter
domain C	enter	--	--

(c) Domain communication ACM

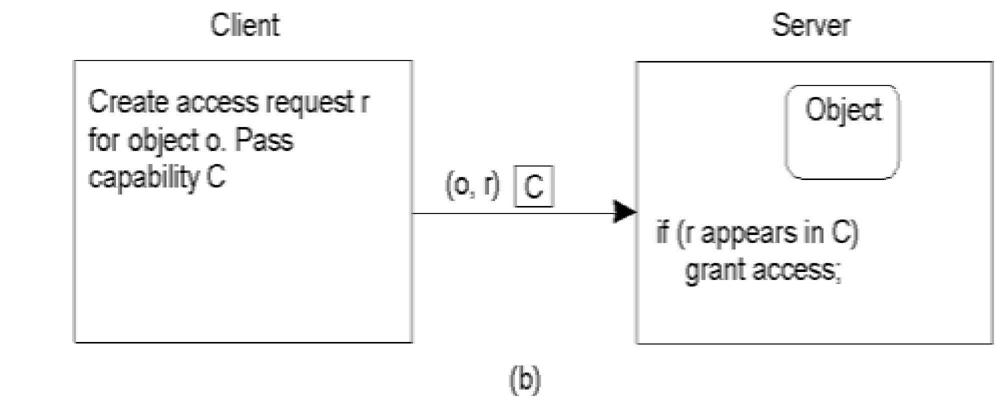
Access Control List

- ACM is simple and straightforward, but if a system supports thousands of users and millions of objects, the ACM will be a very sparse matrix.
- An ACL (Access Control List) is a column of ACM with empty entries removed, each object is assumed to have its own associated ACL.
- Another approach is to distribute the matrix row-wise by giving each subject a list of CL (Capability List).

Comparison between ACL and CL



ACL is associated with Object



CL is associated with Subject

Firewalls

- A Firewall is a special kind reference monitor to control external access to any part of a distributed system.
- A Firewall disconnects any part of a distributed system from outside world, all outgoing and incoming packets must be routed through the firewall.
- A firewall itself should be heavily protected against any kind of security threads.
- Models of firewall:
    - Packet-filtering gateway
    - Proxy:
        - Application-level Proxy
        - Circuit-level Proxy