Distributed Systems
Security

Luu Kim Hoa	00707166
Trinh Xuan Phuong	00707178

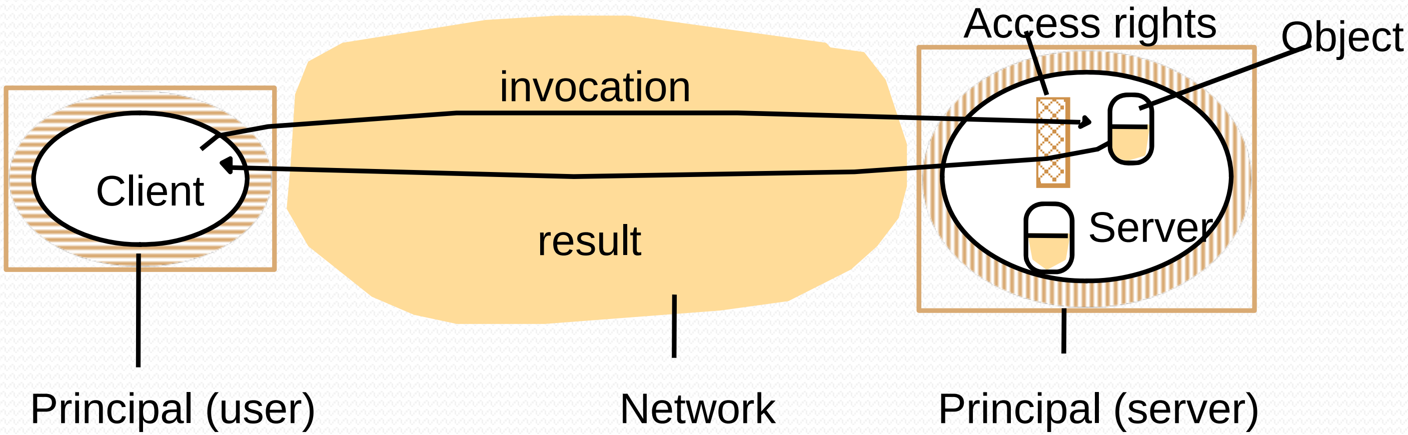
Overview

- Overview of security techniques
- Cryptographic algorithms
- Digital signatures
- Cryptography pragmatics
- Case studies

Historical context: the evolution of security needs

	1965-75	*1975-89*	*1990-99*	*Current*
Platforms	Multi-user timesharing computers	Distributed systems based on local networks	The Internet, wide-area services	The Internet + mobile devices
Shared resources	Memory, files	Local services (e.g. NFS), local networks	Email, web sites, Internet commerce	Distributed objects, mobile code
Security requirements	User identification and authentication	Protection of services	Strong security for commercial transactions	Access control for individual objects, secure mobile code
Security management environment	Single authority, single authorization database (e.g. /etc/passwd)	Single authority, delegation, replicated authorization databases (e.g. NIS)	Many authorities, no network-wide authorities	Per-activity authorities, groups with shared responsibilities

Security model



Threats and forms of attack

- Eavesdropping
  - obtaining private or secret information
- Masquerading
  - assuming the identity of another user/principal
- Message tampering
  - altering the content of messages in transit
    - man in the middle attack (tampers with the secure channel mechanism)
- Replaying
  - storing secure messages and sending them at a later date
- Denial of service
  - flooding a channel or other resource, denying access to others

Security notations

Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A server

K_A	Alice's secret key
K_B	Bob's secret key
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice's private key (known only to Alice)
K_{Apub}	Alice's public key (published by Alice for all to read)
$\{M\}_K$	Message M encrypted with key K
$[M]_K$	Message M signed with key K

Scenario 1:

Secret communication with a shared secret key

Alice and Bob share a secret key K_{AB}.

1. Alice uses K_{AB} and an agreed encryption function $E(K_{AB}, M)$ to encrypt and send any number of messages $\{M_i\}_{K_{AB}}$ to Bob.
2. Bob reads the encrypted messages using the corresponding decryption function $D(K_{AB}, M)$.

Alice and Bob can go on using K_{AB} as long as it is safe to assume that K_{AB} has not been *compromised*.

Issues:

Key distribution: How can Alice send a shared key K_{AB} to Bob securely?

Freshness of communication: How does Bob know that any $\{M_i\}$ isn't a copy of an earlier encrypted message from Alice that was captured by Mallory and replayed later?

Scenario 2:

Authenticated communication with a server

Bob is a file server; Sara is an authentication service. Sara shares secret key K_A with Alice and secret key K_B with Bob.

1. Alice sends an (unencrypted) message to Sara stating her identity and requesting a *ticket* for access to Bob. ➲
2. Sara sends a response to Alice. $\{\{\text{Ticket}\}_{KB}, K_{AB}\}_{KA}$. It is encrypted in K_A and consists of a ticket (to be sent to Bob with each request for file access) encrypted in K_B and a new secret key K_{AB}.
3. Alice uses K_A to decrypt the response.
4. Alice sends Bob a request R to access a file: $\{\text{Ticket}\}_{KB}$, Alice, R.
5. The ticket is actually $\{K_{AB}, \text{Alice}\}_{KB}$. Bob uses K_B to decrypt it, checks that Alice's name matches and then uses K_{AB} to encrypt responses to Alice.

Scenario 3:

Authenticated communication with public keys

Bob has a public/private key pair <K_{Bpub}, K_{Bpriv}>

1. Alice obtains a certificate that was signed by a trusted authority stating Bob's public key K_{Bpub}
2. Alice creates a new shared key K_{AB} , encrypts it using K_{Bpub} using a public-key algorithm and sends the result to Bob.
3. Bob uses the corresponding private key K_{Bpriv} to decrypt it.

(If they want to be sure that the message hasn't been tampered with, Alice can add an agreed value to it and Bob can check it.)

- Mallory might intercept Alice’s initial request to a key distribution service for Bob’s public-key certificate and send a response containing his own public key. He can then intercept all the subsequent messages.

Scenario 4:

Digital signatures with a secure digest function

Alice wants to publish a document M in such a way that anyone can verify that it is from her.

1. Alice computes a fixed-length digest of the document Digest(M).
2. Alice encrypts the digest in her private key, appends it to M and makes the resulting signed document (M, $\{\text{Digest}(M)\}_{K_{Apriv}}$) available to the intended users.
3. Bob obtains the signed document, extracts M and computes Digest(M).
4. Bob uses Alice's public key to decrypt $\{\text{Digest}(M)\}_{K_{Apriv}}$ and compares it with his computed digest. If they match, Alice's signature is verified.

Certificates

Alice’s bank account certificate

1. *Certificate type*	Account number

Certificate: a statement signed by an appropriate authority.

Certificates require:

- An agreed standard format
- Agreement on the construction of chains of trust .
- Expiry dates, so that certificates can be revoked.

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

Cryptographic Algorithms

Message M, key K, published encryption functions E, D

- Symmetric (secret key)

  $E(K, M) = \{M\}_K$ $D(K, E(K, M)) = M$

  Same key for E and D

  M must be hard (infeasible) to compute if K is not known.

  Usual form of attack is brute-force: try all possible key values for a known pair M, $\{M\}_K$. Resisted by making K sufficiently large ~ 128 bits

- Asymmetric (public key)

  Separate encryption and decryption keys: K_e, K_d

  $D(K_d. E(K_e, M)) = M$

  depends on the use of a *trap-door function* to make the keys. E has high computational cost. Very large keys > 512 bits

- Hybrid protocols - used in SSL (now called TLS)

  Uses asymmetric crypto to transmit the symmetric key that is then used to encrypt a session.

Symmetric encryption algorithms

These are all programs that perform confusion and diffusion operations on blocks of binary data

TEA: a simple but effective algorithm developed at Cambridge U (1994) for teaching and explanation. *128-bit key, 700 kbytes/sec*

DES: The US Data Encryption Standard (1977). No longer strong in its original form. *56-bit key, 350 kbytes/sec*.

Triple-DES: applies DES three times with two different keys. *112-bit key, 120 Kbytes/sec*

IDEA: International Data Encryption Algorithm (1990). Resembles TEA. *128-bit key, 700 kbytes/sec*

AES: A proposed US Advanced Encryption Standard (1997). *128/256-bit key*.

There are many other effective algorithms. See Schneier [1996].

The above speeds are for a Pentium II processor at 330 MHZ. Today's PC's (January 2002) should achieve a 5 x speedup.

Asymmetric encryption algorithms

RSA: The first practical algorithm (Rivest, Shamir and Adelman 1978) and still the most frequently used. Key length is variable, 512-2048 bits. Speed 1-7 kbytes/sec. (350 MHz PII processor)

Elliptic curve: A recently-developed method, shorter keys and faster.

Asymmetric algorithms are ~1000 x slower and are therefore not practical for bulk encryption, but their other properties make them ideal for key distribution and for authentication uses.

Digital signatures

Requirement:

- To authenticate stored document files as well as messages
- To protect against forgery
- To prevent the signer from repudiating a signed document (denying their responsibility)

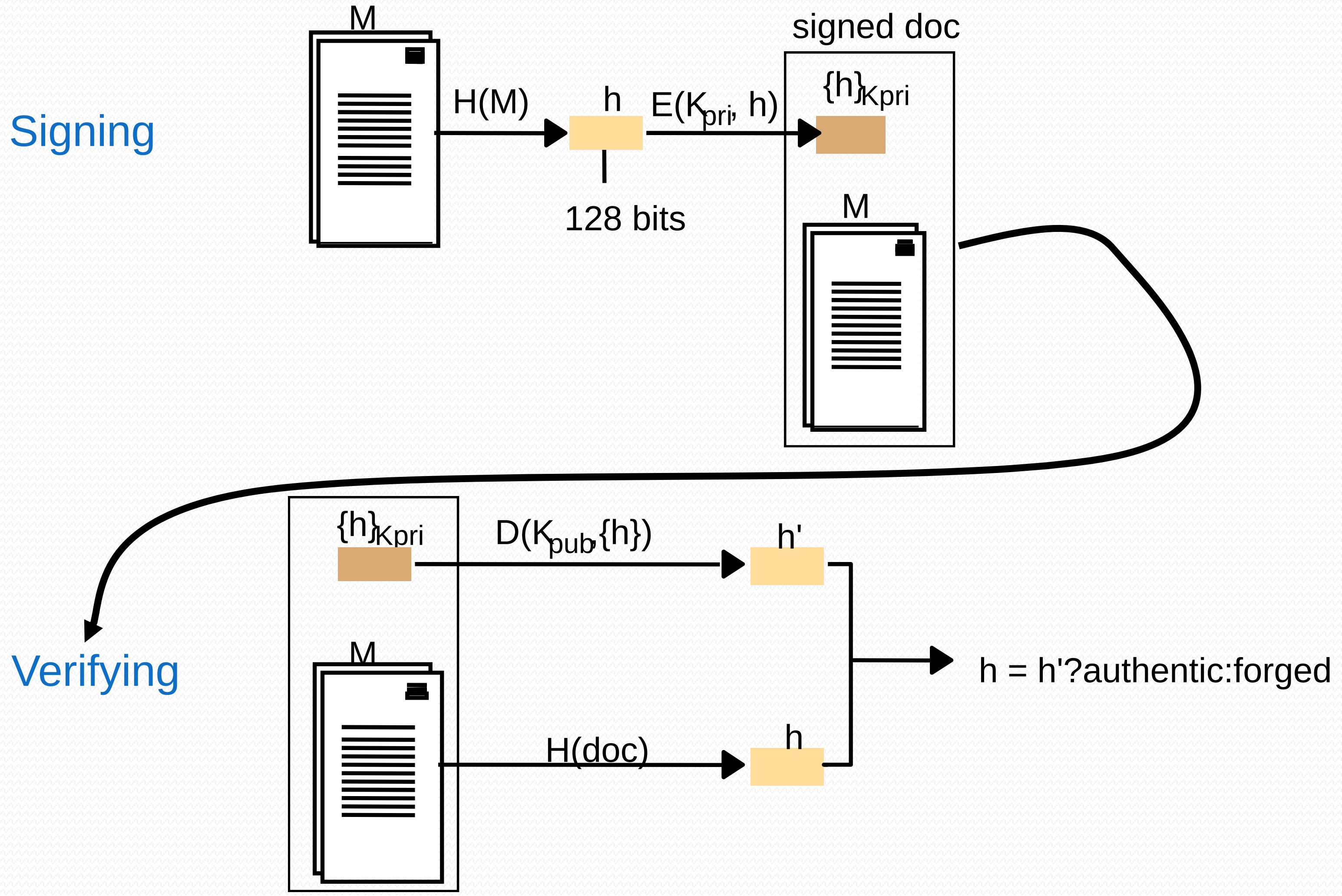
Encryption of a document in a secret key constitutes a signature

- impossible for others to perform without knowledge of the key
- strong authentication of document
- strong protection against forgery
- weak against repudiation (signer could claim key was compromised)

Secure digest functions

- Encrypted text of document makes an impractically long signature
  - so we encrypt a *secure digest* instead
  - A secure digest function computes a fixed-length hash H(M) that characterizes the document M
  - H(M) should be:
    - fast to compute
    - hard to invert - hard to compute M given H(M)
    - hard to defeat in any variant of the Birthday Attack
- **MD5**: Developed by Rivest (1992). *Computes a 128-bit digest. Speed 1740 kbytes/sec.*
- **SHA**: (1995) based on Rivest's MD4 but made more secure by producing a *160-bit digest, speed 750 kbytes/second*
- **Any symmetric encryption algorithm** can be used in CBC (cipher block chaining) mode. The last block in the chain is H(M)

Digital signatures with public keys



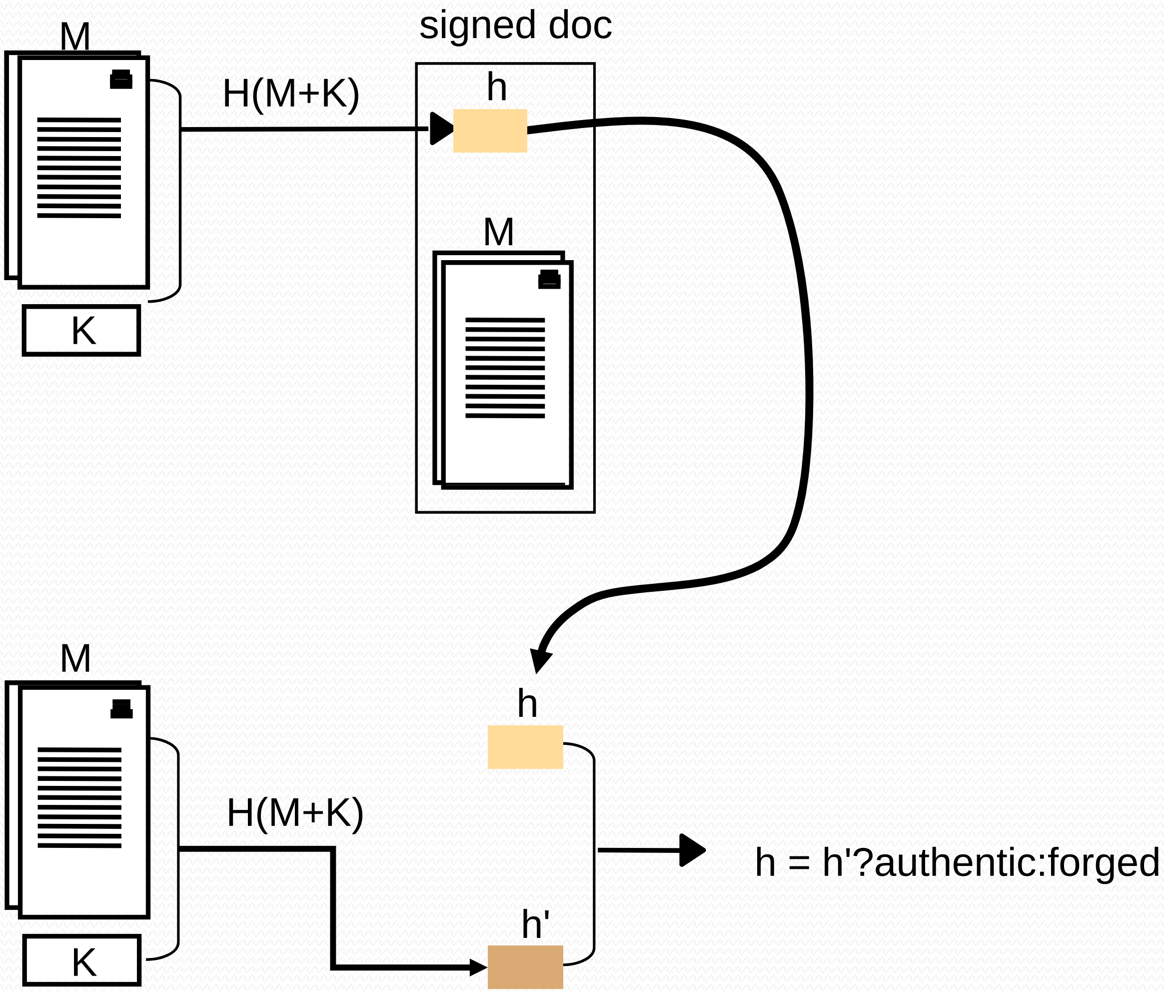
MACs: Low-cost signatures with a shared secret key

MAC: Message Authentication Code

Signing

Signer and verifier share a secret key K

Verifying



M

K

H(M+K)

signed doc

h

M

M

K

H(M+K)

h

h'

h = h'?authentic:forged

Case studies

- Needham - Schroeder protocol
- Kerberos protocol
- Secure Socket Layer (SSL) protocol

Needham - Schroeder protocol

In early distributed systems (1974-1984) it was difficult to protect the servers

- E.g. against masquerading attacks on a file server
- Because there was no mechanism for authenticating the origins of requests
- Public-key cryptography was not yet available or practical

Needham and Schroeder therefore developed an authentication and key-distribution protocol for use in a local network

- An early example of the care required to design a safe security protocol
- Introduced several design ideas including the use of *nonces*.

The Needham–Schroeder secret-key authentication protocol

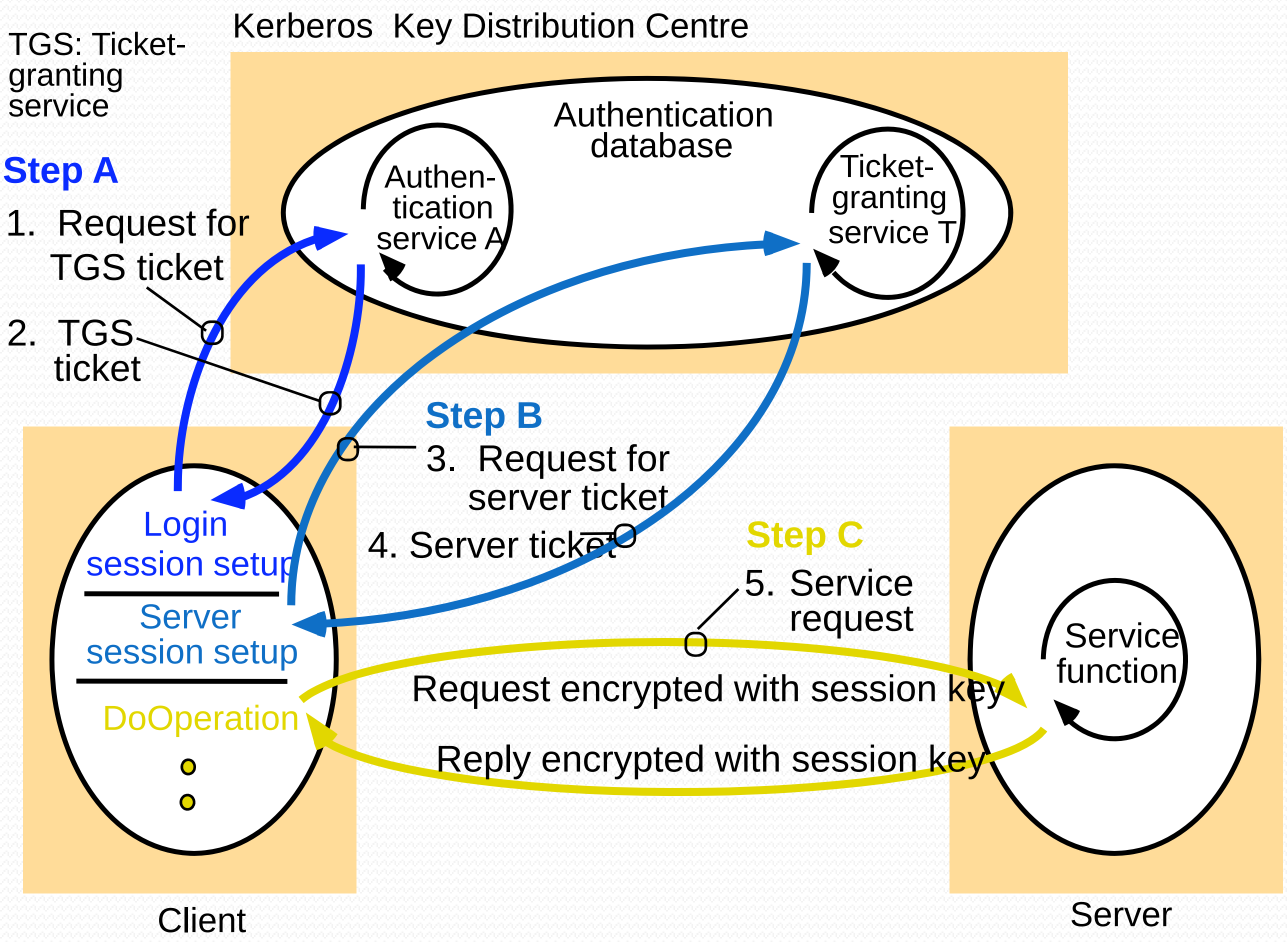
Weakness: Message 3 might not be fresh - and K_{AB} could have been compromised in the store of A's computer. Kerberos addresses this by adding a timestamp or a nonce to message 3.

Header	*Message*	*Notes*
1. A->S:	A, B, N_A	A requests S to supply a key for communication with B.
		nse
3. A->B:	$\{K_{AB}, A\}_{KB}$	
4. B->A:	$\{N_B\}_{KAB}$	B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B.
5. A->B:	$\{N_B - 1\}_{KAB}$	A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B.

Kerberos authentication and key distribution service

- Secures communication with servers on a local network
  - Developed at MIT in the 1980s to provide security across a large campus network > 5000 users
  - Based on Needham - Schroeder protocol
- Standardized and now included in many operating systems
  - Internet RFC 1510
  - BSD UNIX, Linux, Windows 2000, NT, XP, etc.
  - Available from MIT
- Kerberos server creates a shared secret key for any required server and sends it (encrypted) to the user's computer
- User's password is the initial secret shared with Kerberos

System architecture of Kerberos



TGS: Ticket-granting service

Kerberos Key Distribution Centre

Authentication database

Authen-tication service A

Ticket-granting service T

Step A

1. Request for TGS ticket
2. TGS ticket

Step B

3. Request for server ticket
4. Server ticket

Step C

5. Service request

Request encrypted with session key

Reply encrypted with session key

Login session setup

Server session setup

DoOperation

Client

Service function

Server

Needham - Schroeder protocol

1. A->S: A, B, N_A
2. S->A: $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{KB}\}_{KA}$
3. A->B: $\{K_{AB}, A\}_{KB}$
4. B->A: $\{N_B\}_{KAB}$
5. A->B: $\{N_B - 1\}_{KAB}$

Step A once per login session

Step B once per server session

Step C once per server transaction

Advances and Weakness

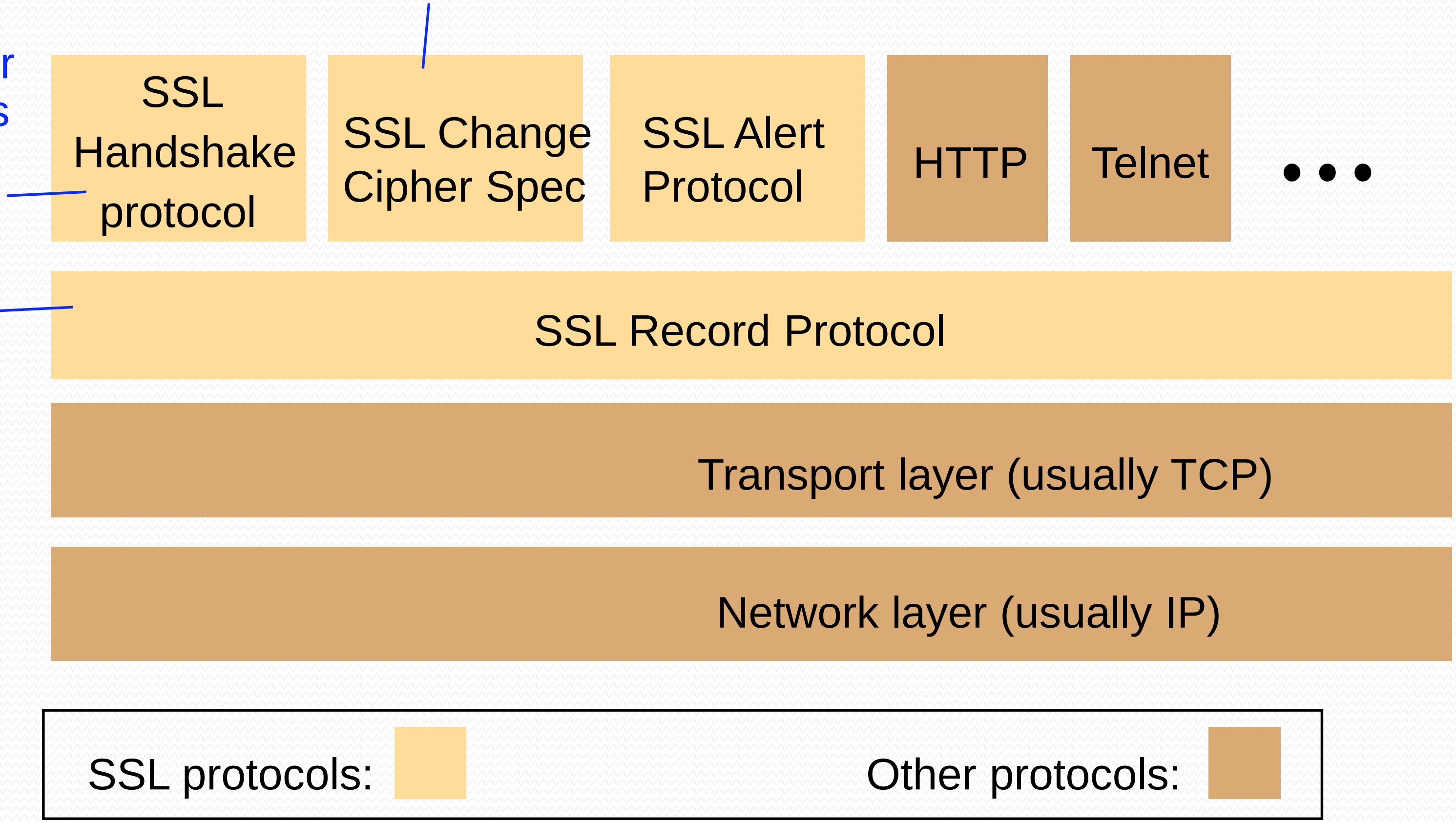
- Advances
  - Secures communication
  - Single sign on
  - Mutual authentication
  - Don’t send clear user’s password on a insecure network

- Weakness
  - KDC
  - User’s experiences

The Secure Socket Layer (SSL)

- Key distribution and secure channels for internet commerce
  - Hybrid protocol; depends on public-key cryptography
  - Originally developed by Netscape Corporation (1994)
  - Extended and adopted as an Internet standard with the name Transport Level Security (TLS)
  - Provides the security in all web servers and browsers and in secure versions of Telnet, FTP and other network applications
- Design requirements
  - Secure communication without prior negotation or help from 3rd parties
  - Free choice of crypto algorithms by client and server
  - Communication in each direction can be authenticated, encrypted or both

SSL protocol stack



changes the secure channel to a new spec

negotiates cipher suite, exchanges certificates and key masters

implements the secure channel

SSL Handshake protocol

SSL Change Cipher Spec

SSL Alert Protocol

HTTP

Telnet

• • •

SSL Record Protocol

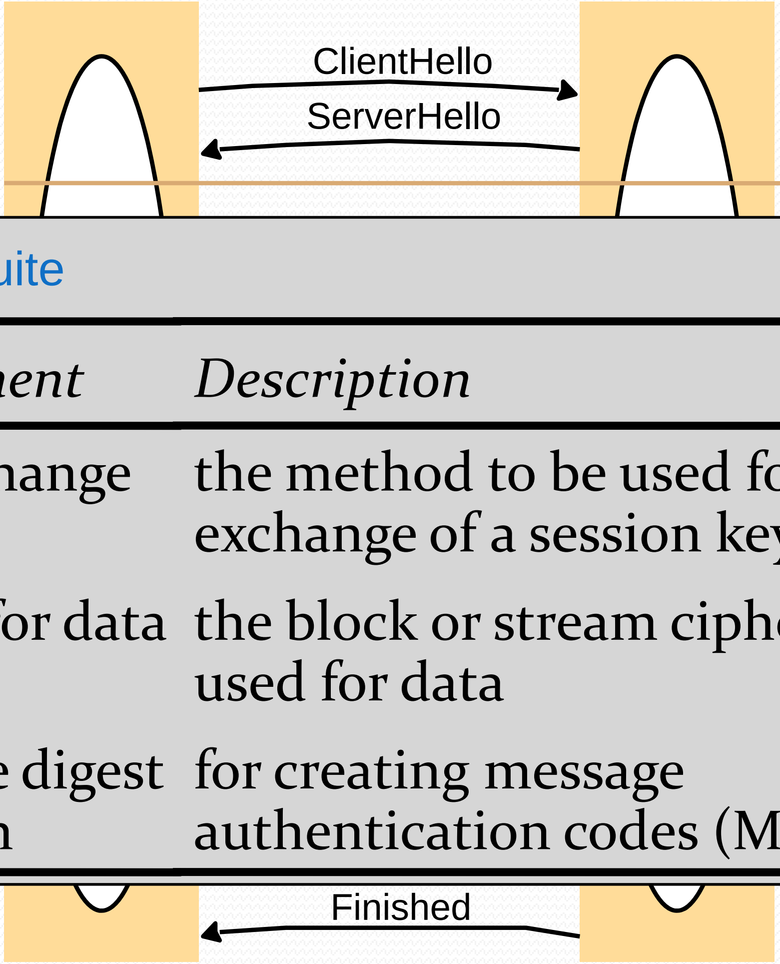
Transport layer (usually TCP)

Network layer (usually IP)

SSL protocols: 🟨

Other protocols: 🟫

SSL handshake protocol



Establish protocol version, session ID, cipher suite, compression method, exchange random start values

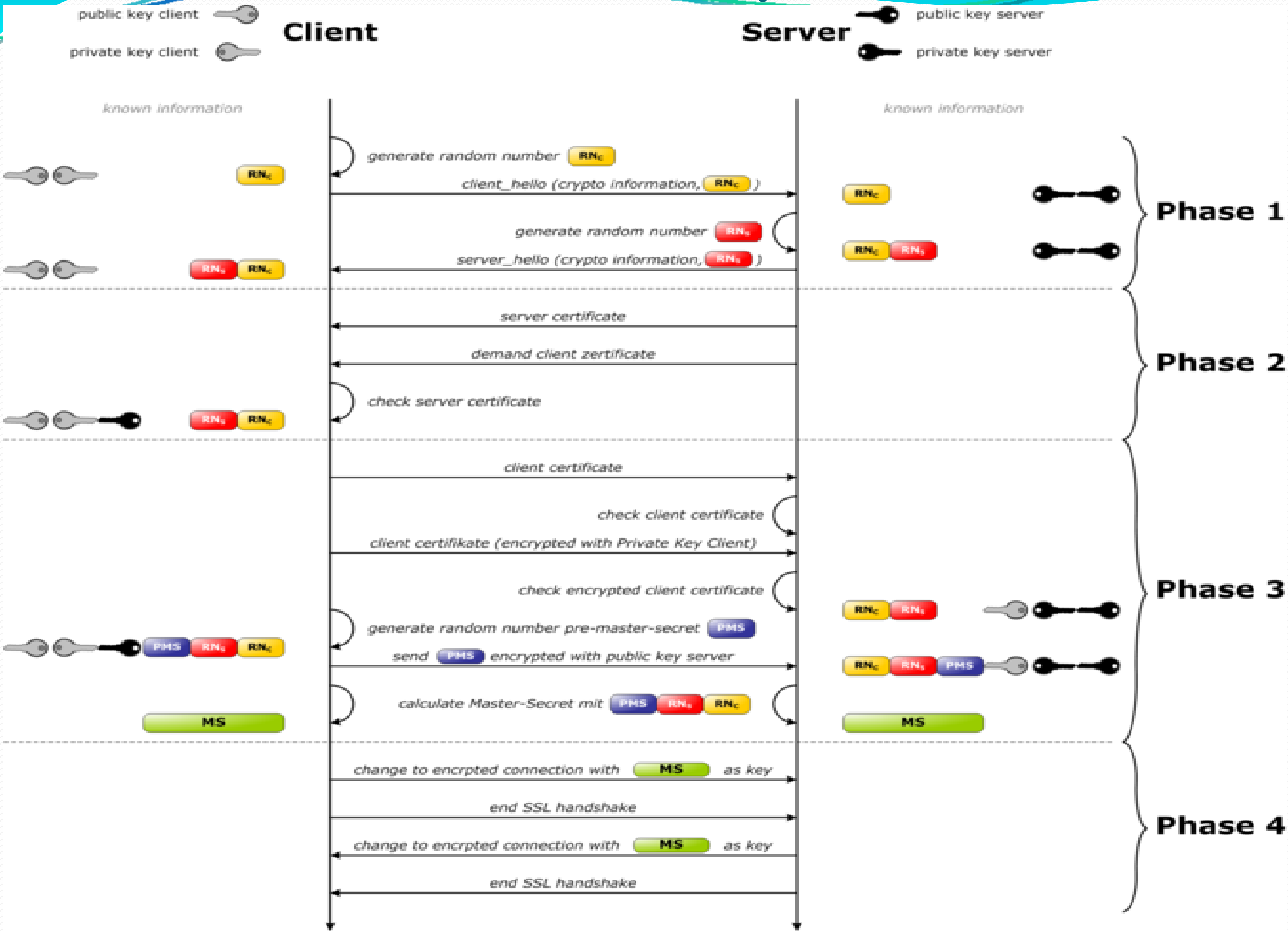
Cipher suite

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA

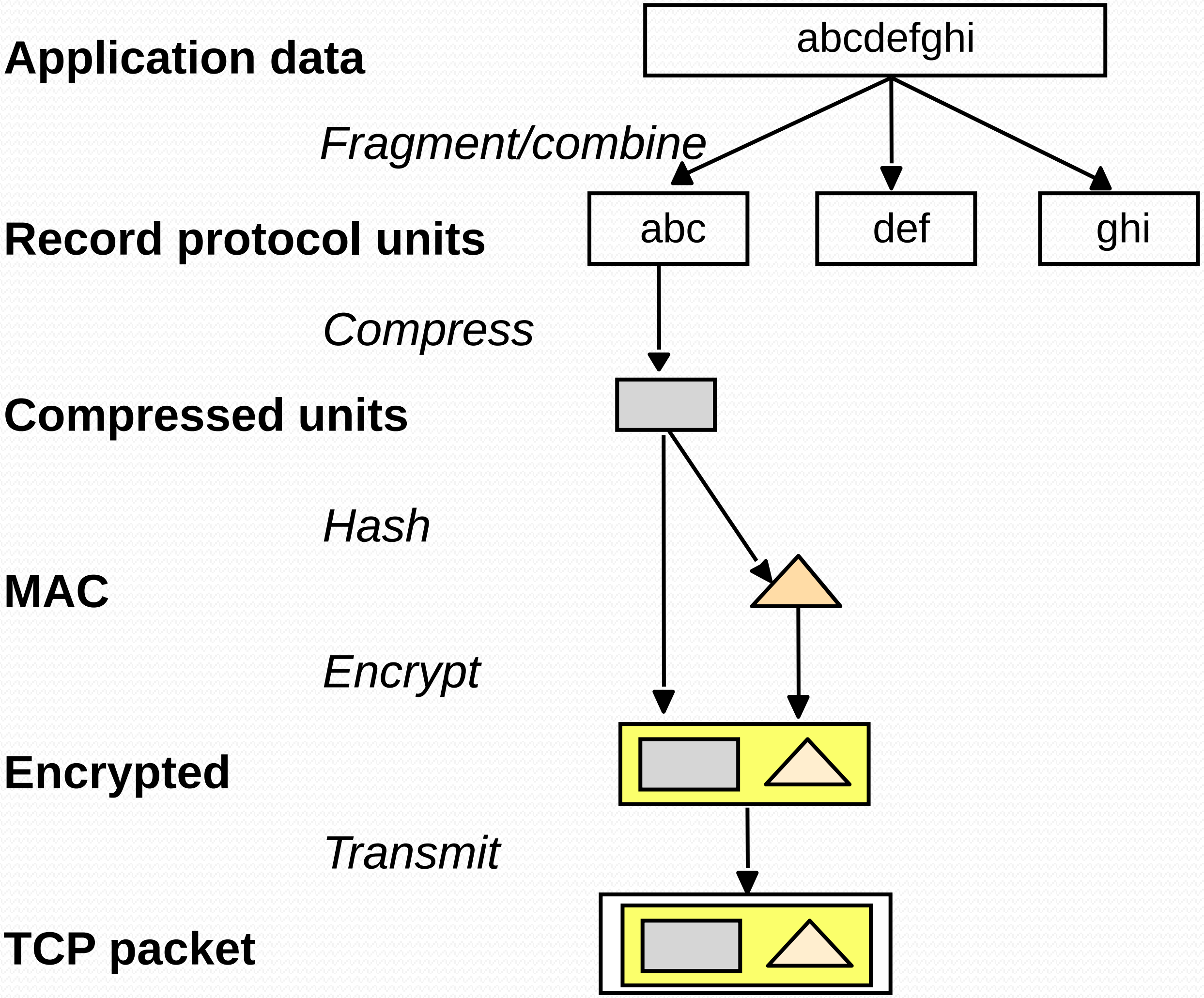
to generate:

2 session keys	*2 MAC keys*
K_{AB}	M_{AB}
K_{BA}	M_{BA}

SSL handshake protocol..



SSL record protocol



Application data

abcdefghi

Fragment/combine

Record protocol units

abc

def

ghi

Compress

Compressed units

Hash

MAC

Encrypt

Encrypted

Transmit

TCP packet

Summary

- It is essential to protect the resources, communication channels and interfaces of distributed systems and applications against attacks.
- This is achieved by the use of access control mechanisms and secure channels.
- Public-key and secret-key cryptography provide the basis for authentication and for secure communication.
- Kerberos and SSL are widely-used system components that support secure and authenticated communication.

References

- [1] G. Coulouris, J. Dollimore and T. Kindberg, “Distributed Systems, Concepts and Design”, Addison Wesley, 2001.
- [2] Jason Garman, “Kerberos: The Definitive Guide”, O’Reilly, 08/2003.
- [3] http://en.wikipedia.org

Thank you!!!

Questions & Answer!!!