

Distributed Systems

COMP90015 2019 SM1

Indirect Communication

Lectures by Aaron Harwood

© University of Melbourne 2019

Overview

- Group Communication
- Publish/Subscribe *发布订阅*
- Message Queues
- Shared Memory

Introduction

Indirect communication is defined as communication between entities in a distributed system through an intermediary with no direct coupling between the sender and the receiver(s).

- *Space uncoupling*: sender does not know or need to know the identity of the receiver(s)
- *Time uncoupling*: sender and receiver can have independent lifetimes, they do not need to exist at the same time

Time uncoupling is not synonymous with asynchronous communication. Asynchronous communication doesn't imply that the receiver has an independent lifetime, in other words we could consider a time coupled asynchronous system.

Group Communication

Group communication offers a space uncoupled service whereby a message is sent to a group and then this message is delivered to all members of the group. It provides more than a primitive IP multicast:

- manages group membership
- detects failures and provides reliability and ordering guarantees

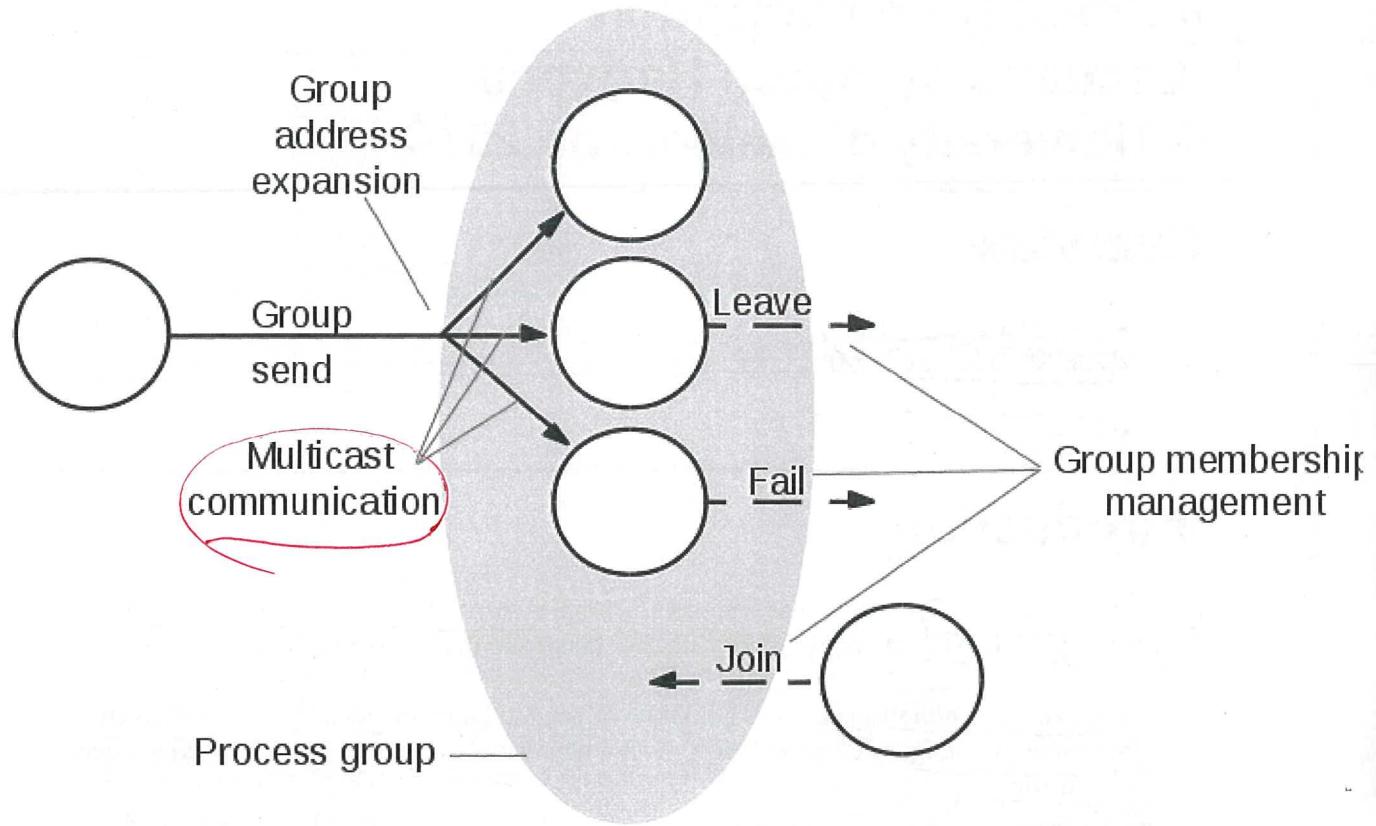
Typical aspects of an API:

- group
- group membership
- join
- leave
- multicast, broadcast

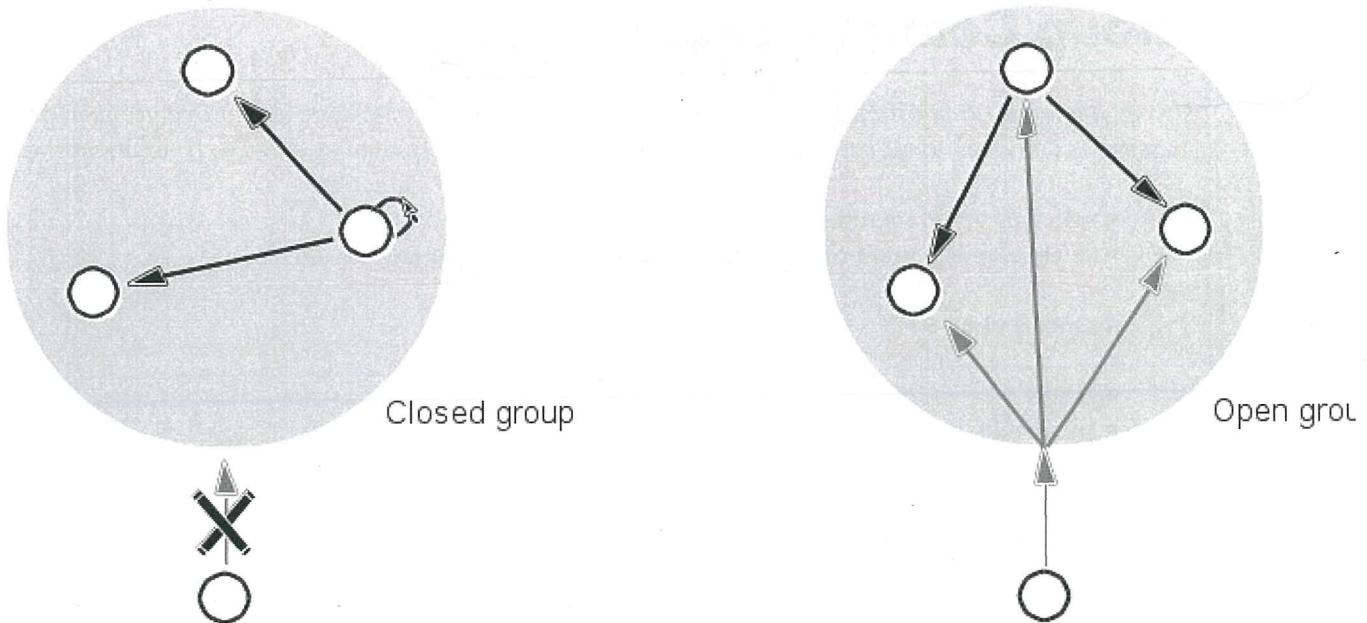
Efficient sending to multiple receivers, instead of multiple independent send operations, is an essential feature of group communication.

broadcast bandwidth.

Group Model



Group services



- 重複**
- closed groups only allow group members to multicast to it
 - overlapping groups allows entities to be members of multiple groups
 - synchronous and asynchronous variations can be considered

多組成員

Implementation issues

JGroups

- 可靠性和順序**
- reliability and ordering in multicast
 - FIFO (first in first out) ordering is concerned with preserving the order from the perspective of a sender process
 - causal ordering, a message that happens before another message will be preserved in that order in the delivery at all processes
 - total ordering, if a message is delivered before another message at one process then this is preserved at all processes
 - group membership management
 - group members leave and join
 - failed members
 - notifying members of group membership changes
 - changes to the group address

Publish/Subscribe Systems

publish / subscribe

Publish/subscribe systems are sometimes referred to as distributed event-based systems.

A publish/subscribe system is a system where publishers publish structured events to an event service and subscribers express interest in particular events through subscriptions which can be arbitrary patterns over the structured events.

- financial information systems
- live feeds of real-time data, e.g. RSS feeds
- support for cooperative working, where a number of participants need to be informed of events of shared interest
- support for ubiquitous computing, including management of events emanating from the ubiquitous infrastructure, e.g. location events
- a broad set of monitoring applications, including network monitoring in the Internet

Events and notifications

RMI and RPC support the synchronous communication model where the client invoking the call waits for the results to be returned.

Events and notifications are associated with the asynchronous communication model.

Distributed event-based systems can use the publish-subscribe communication paradigm:

- Objects that generate events publish information that are of interest to other objects.
- Objects that are interested in a particular type of event subscribe to the type of events.
- Publishers and subscriber are loosely coupled.

松散的

heterogeneity

*heterogeneity
asynchronous.
asynchronous*

Characteristics of distributed event-based systems

Heterogeneity: Allows objects that were not designed to interoperate to communicate due to the loosely coupled nature.

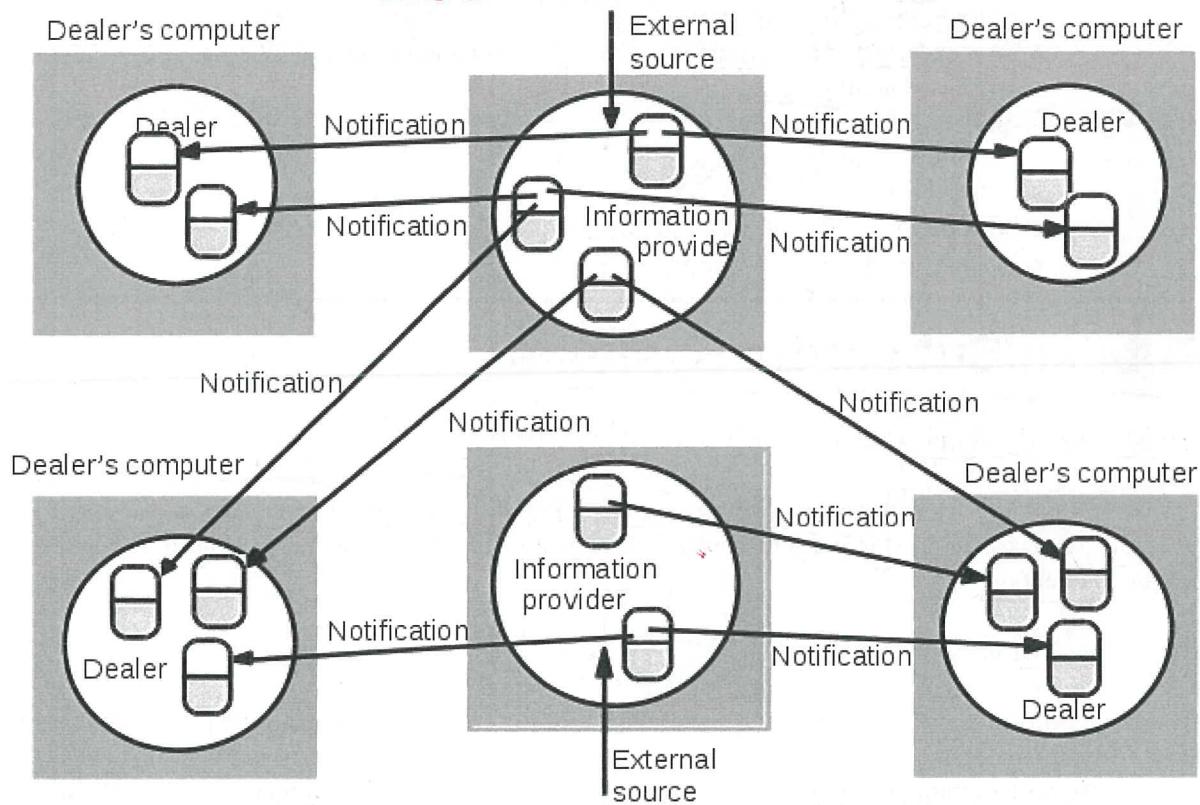
heterogeneity

Asynchronous: Communication is asynchronous and event driven.

asynchronous

Example: Simple dealing room system

交易室



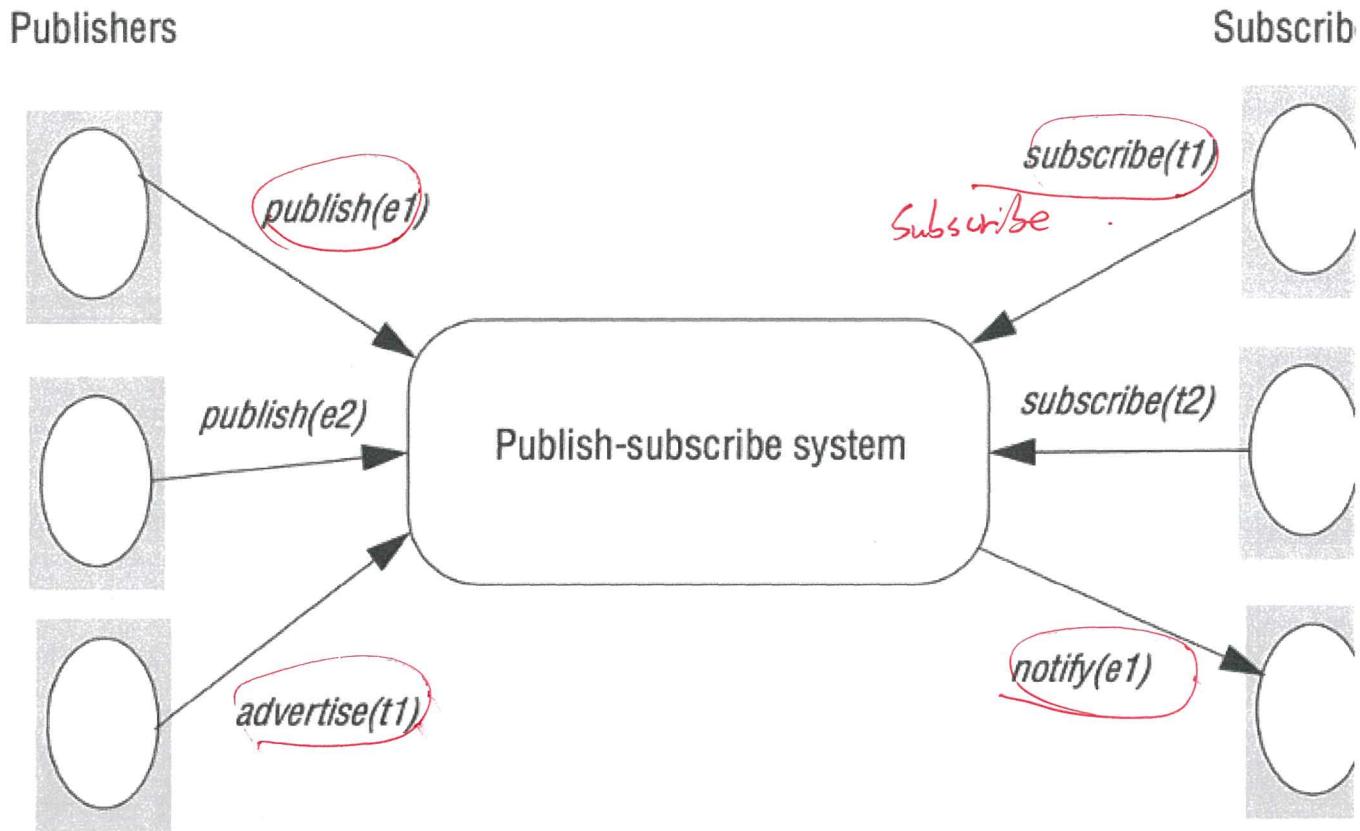
Event Types

Events sources can generate different types of events. Attributes contain informations about the event.

Types and attributes are used by subscribers when subscribing to events.

Notifications occur when event types and attributes match to that of subscriptions.

Programming model



Advertise provides an additional mechanism for publishers to declare the nature of future events, i.e. the types of events of interest that may occur.

Types of publish-subscribe systems

Channel Based: Publishers publish to named channels and subscribers subscribe to all events on a named channel.

Type Based: Subscribers register interest in types of events and notifications occur when particular types of events occur.

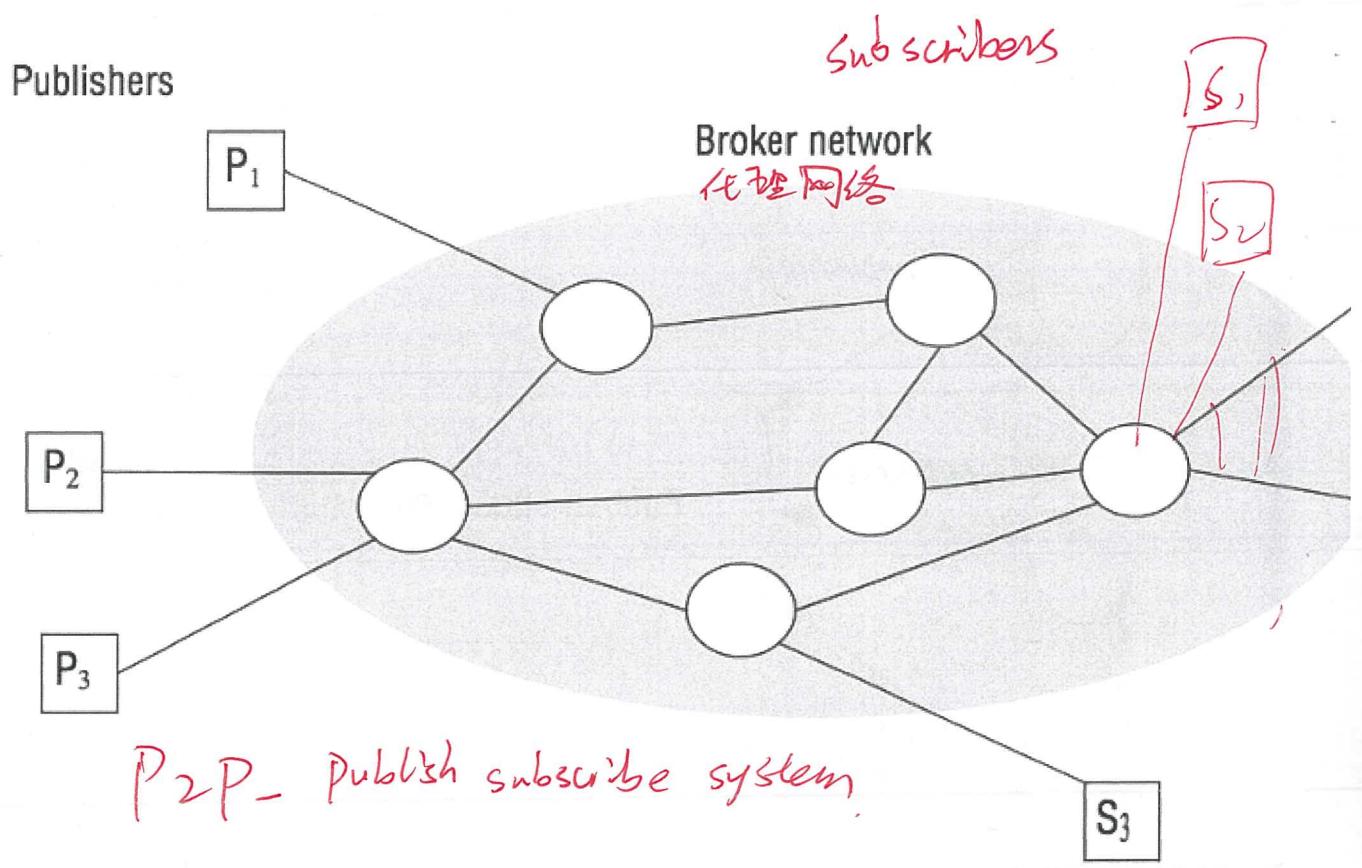
Topic Based: Subscribers register interest in particular topics and notifications occur when any information related to the topic arrives.

Content Based: This is the most flexible of the schemes. Subscribers can specify interest in particular values or ranges of values for multiple attributes. Notifications are based on matching the attribute specification criteria.

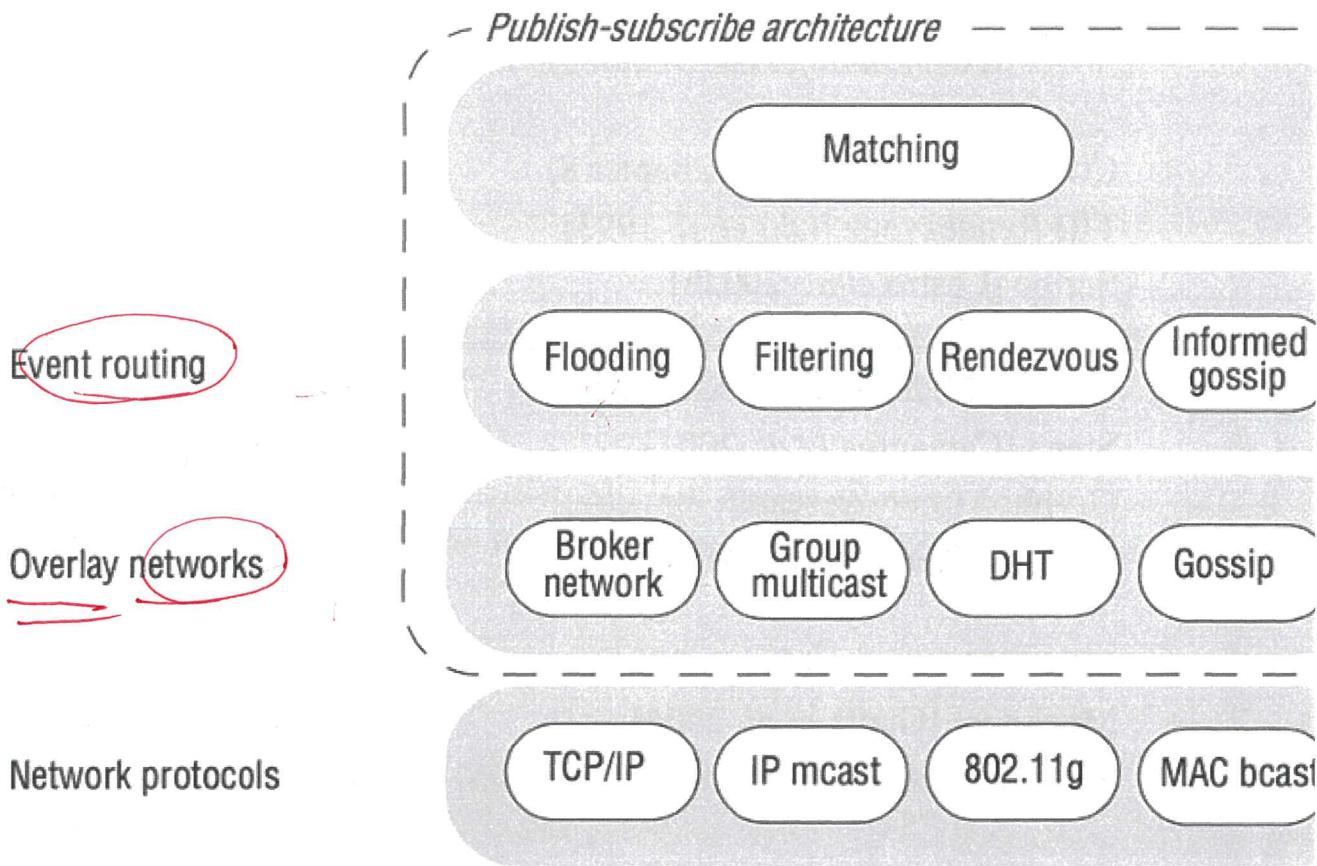
Centralized versus decentralized

集中

分布



Overall System Architecture

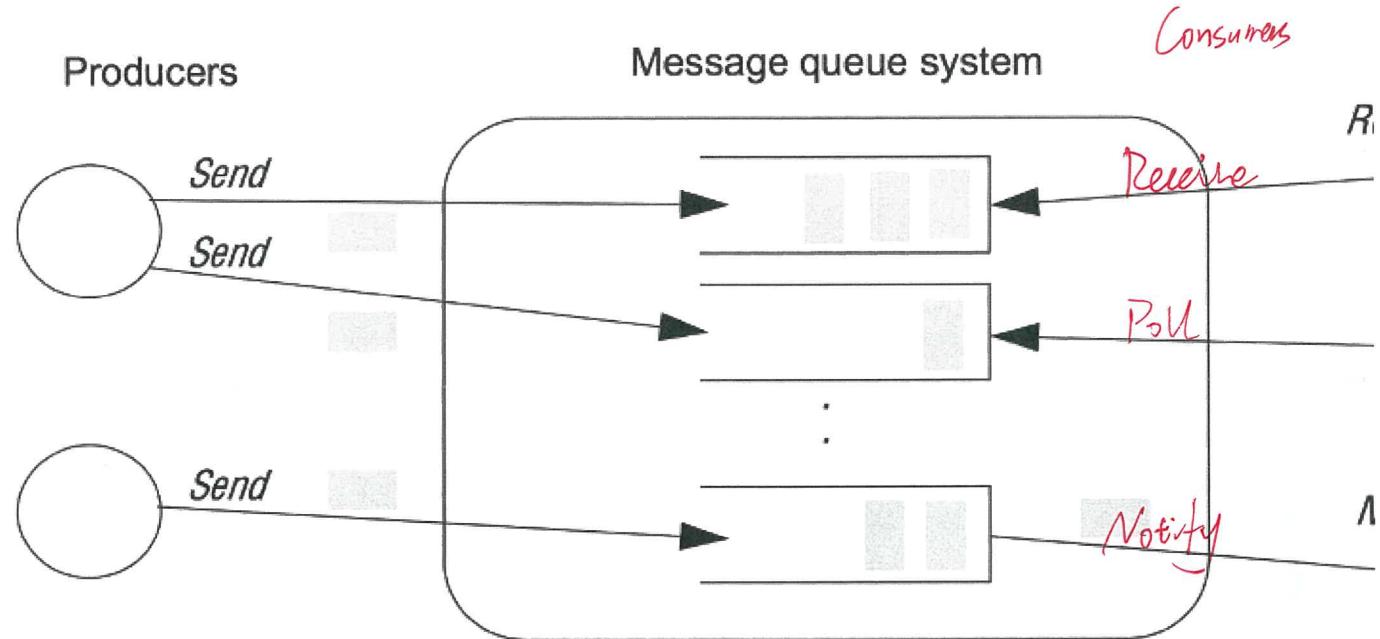


Examples of pub/sub systems

<i>System (and further reading)</i>	<i>Subscription model</i>	<i>Distribution model</i>
CORBA Event Service (Chapter 8)	Channel-based	Centralized
TIB Rendezvous [Oki <i>et al.</i> 1993]	Topic-based	Distributed
Scribe [Castro <i>et al.</i> 2002b]	Topic-based	Peer-to-peer (DHT)
TERA [Baldoni <i>et al.</i> 2007]	Topic-based	Peer-to-peer
Siena [Carzaniga <i>et al.</i> 2001]	Content-based	Distributed
Gryphon [www.research.ibm.com]	Content-based	Distributed
Hermes [Pietzuch and Bacon 2002]	Topic- and content-based	Distributed
MEDYM [Cao and Singh 2005]	Content-based	Distributed
Meghdoot [Gupta <i>et al.</i> 2004]	Content-based	Peer-to-peer
Structure-less CBR [Baldoni <i>et al.</i> 2005]	Content-based	Peer-to-peer

Message Queues

Whereas groups and publish/subscribe provide a one-to-many style of communication, message queues provide a point-to-point service using the concept of a message queue as an indirection, thus achieving the desired properties of space and time uncoupling. They are point-to-point in that the sender places the message into a queue, and it is then removed by a single process.



①持久: 可靠传递
persistent: reliable delivery

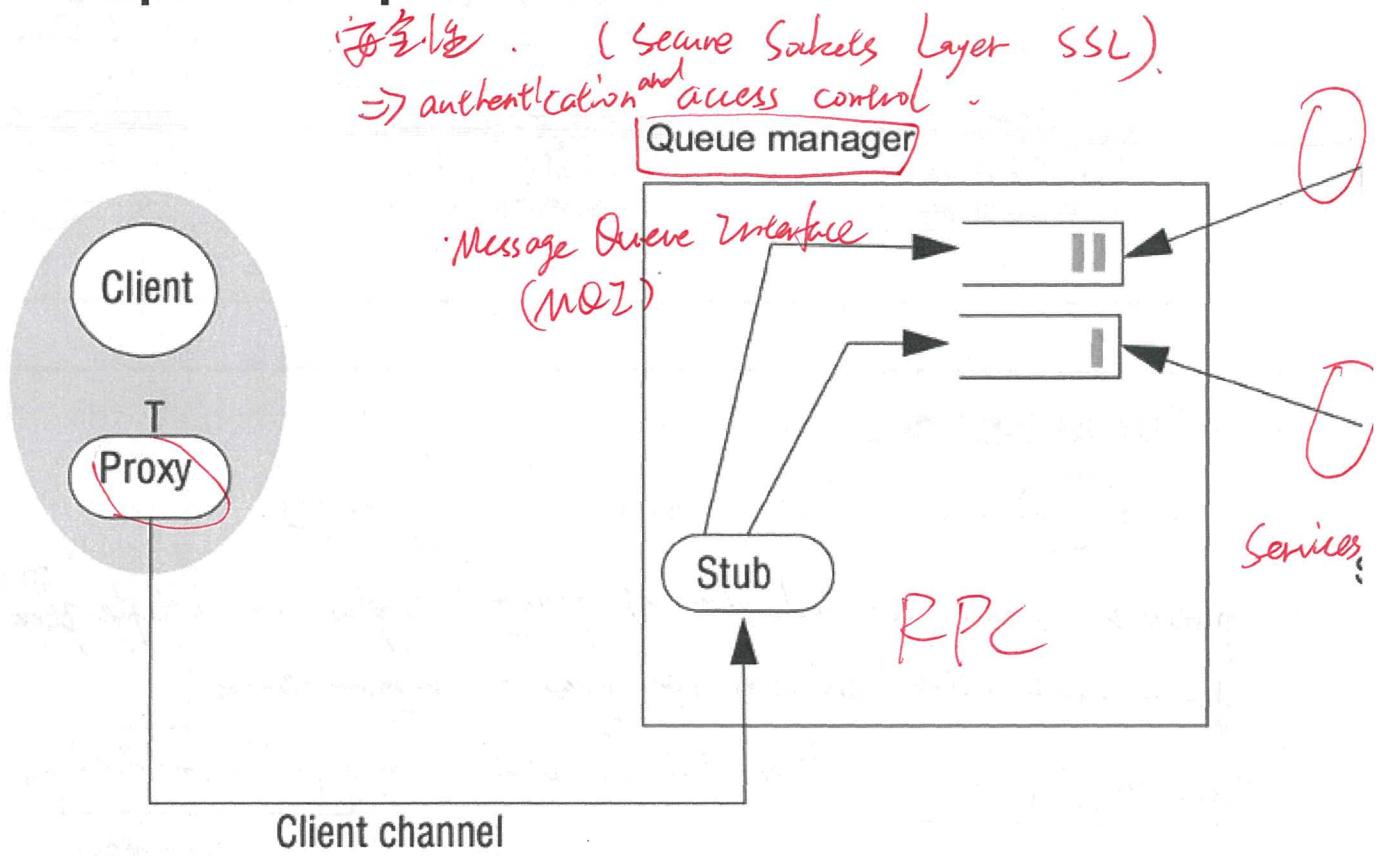
message queues will store the message until they are consumed.

Programming model

- send: producers put a message on a particular queue
- blocking receive: a consumer waits for at least one message on a queue then returns
- non-blocking receive: or poll, a consumer will check and get a message if there, otherwise it returns without a message
- notify: an event is generated at the receiver when a message is available

① FIFO
② 优先级. priority.

Example WebSphere MQ



Shared memory approaches

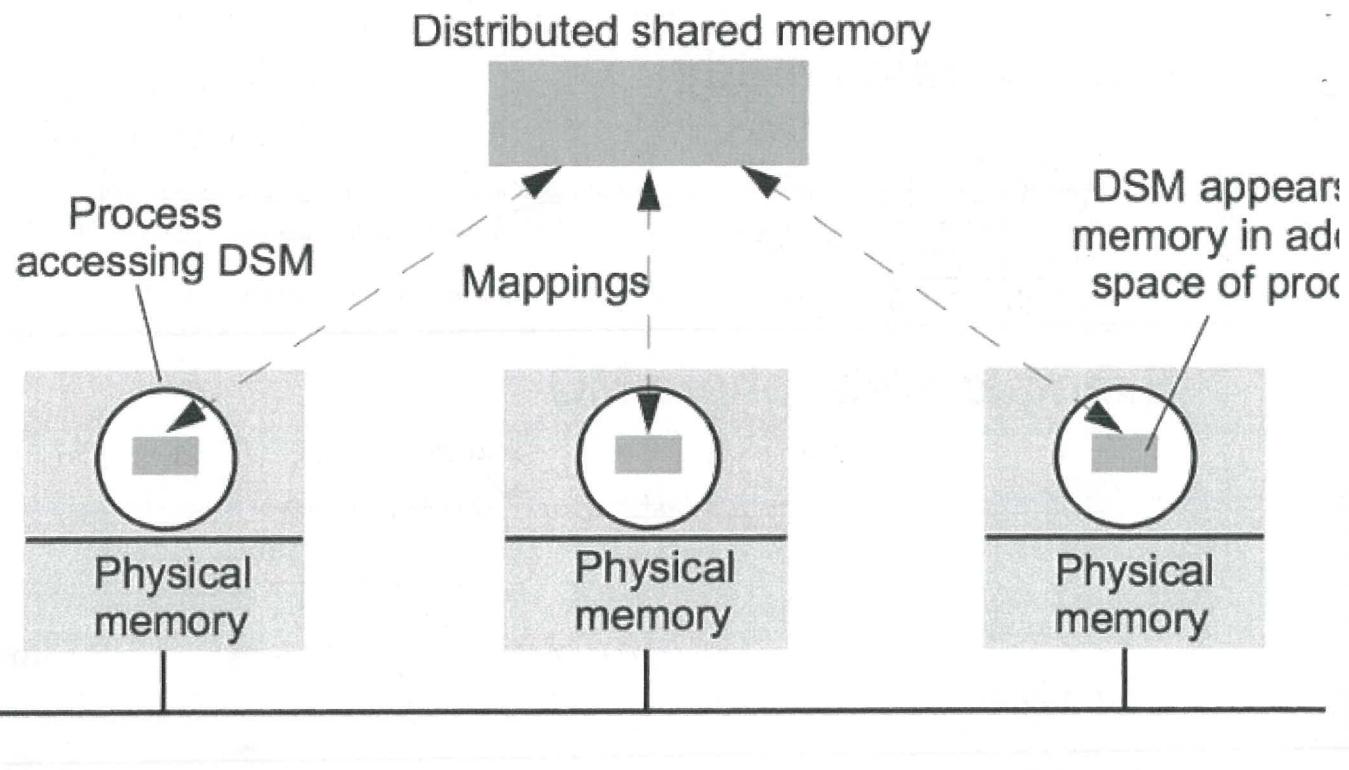
Distributed shared memory is an abstraction for sharing data between computers that do not share physical memory. Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.

- ① spares the programmer the concerns of message passing when writing applications.
- ② a tool for parallel applications or distributed applications.
- ③ hot C-S.

Shared memory approaches

④ a synchronous communication.

⑤ NO marshalling

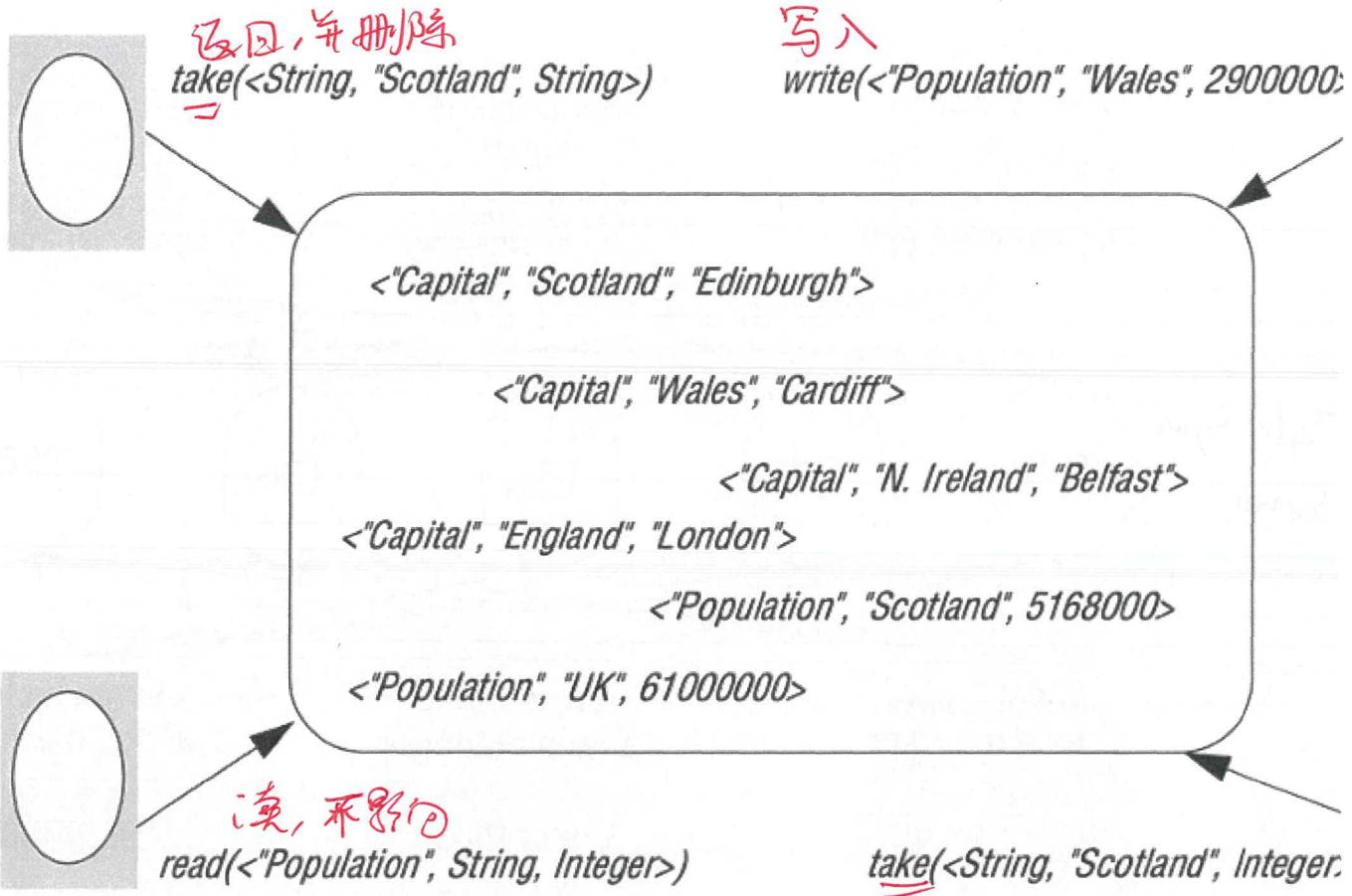


Tuple Spaces

The tuple space is a more abstract form of shared memory, compared to DSM.

Processes communicate indirectly by placing tuples in a tuple space, from which other processes can read or remove them.

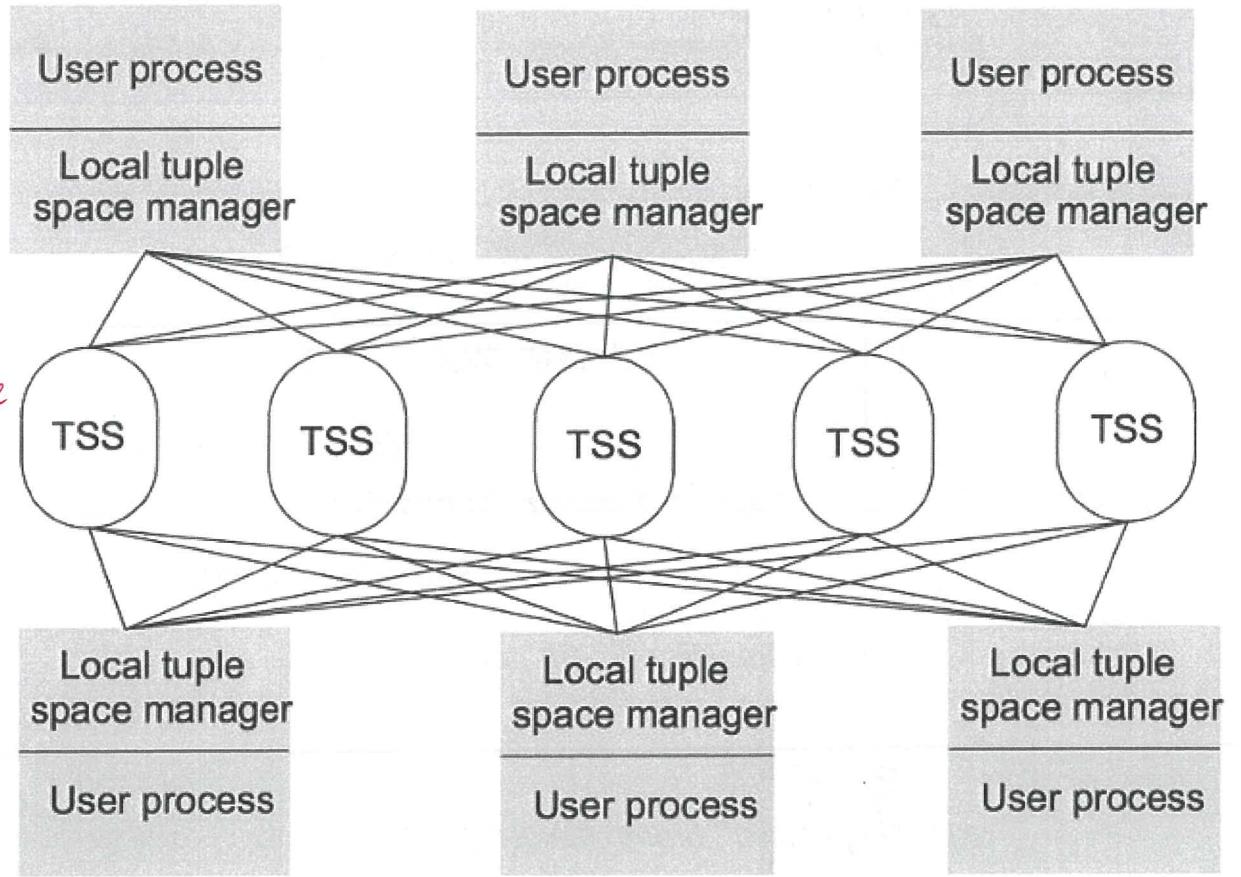
Tuple doesn't have an address but are accessed by pattern matching on content



Example York Linda Kernel

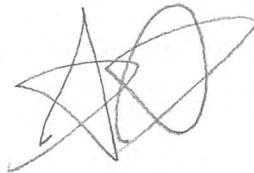
The implementation uses multiple Tuple Space Servers.

out → write
 rd → read
 in → take



Summary

	<i>Groups</i>	<i>Publish-subscribe systems</i>	<i>Message queues</i>	<i>DSM</i>	<i>Tuple spaces</i>
<i>Space-uncoupled</i>	Yes	Yes	Yes	Yes	Yes
<i>Time- uncoupled</i>	Possible	Possible	Yes	Yes	Yes
<i>Style of service</i>	Communication-based	Communication-based	Communication-based <i>Scalability very large</i>	State-based	State-based
<i>Communication pattern</i>	1-to-many	1-to-many	1-to-1 <i>Scale System</i>	1-to-many	1-1 or 1-to-m
<i>Main intent purpose</i>	Reliable distributed computing	Information dissemination or EAI; mobile and ubiquitous systems	Information dissemination or EAI; commercial transaction processing	Parallel and distributed computation	Parallel and distributed computation; mobile and ubiquitous systems
<i>Scalability</i>	Limited	Possible	Possible	Limited	Limited
<i>Associative</i>	No	Content-based publish-subscribe only	No	No	Yes
<i>关联寻址</i>					





Distributed Systems

COMP90015 2019 SM1

Security

Lectures by Aaron Harwood

© University of Melbourne 2019

Overview

- introduction
- overview of security techniques
- cryptographic algorithms
- digital signatures
- cryptography pragmatics

Introduction

The challenges of security arise as a result of the need to share or to distribute resources. If a resource is not to be shared or distributed then it can be physically isolated from external access.

A security policy provides a statement of the required integrity privacy of shared information and other limits to the allowable usage of a shared resource.

A security policy is enforced using a security mechanism.

Digital cryptography provides the basis for most computer security mechanisms, though computer security and cryptography are distinct subjects.

Threats and attacks

Some threats are obvious -- e.g., reading traffic on a shared network to gain information like a password or other personal information.

Some threats are unobvious -- e.g., pretending to be an official server.

Some threats attack the mechanism -- e.g., the attacker purchases something with their credit card and later denies that they actually did the purchase.

Security threats fall into three broad classes:

- Leakage -- the acquisition of information by unauthorized recipients;
- Tampering -- the unauthorized alteration of information;
- Vandalism -- interference with the proper operation of a system without gain to the perpetrator.

Attacks on distributed systems depend on access to an existing communication channel. A communication

① Eavesdropping: ~~watch~~
Masquerading: ~~fake~~

intercepting.

channel can be misused in different ways:

- Eavesdropping -- obtaining copies of messages without authority.
- Masquerading -- sending or receiving messages using the identity of another principal without their authority.
- Message tampering -- intercepting messages and altering their contents before passing them on (or substituting a different message in their place); e.g. the man-in-the-middle attack.
- Replaying -- storing intercepted messages and sending them at a later date.
- Denial of service -- flooding a channel or other resource with messages in order to deny access for others.

Some attacks can be arguable, e.g. to what extent is spam email considered a denial of service attack?

Threats from mobile code

Some distributed systems allow code, called mobile code, to be communicated to a remote host, to be executed by that host. In this case it is necessary to ensure that the host, including all processes and resources available at the host, is secure from any operations that the mobile code undertakes; while of course still allowing legitimate operations.

Similar to this is the case where code is delivered in an email or via the web browser. The operating system will typically ask the user whether the code should be trusted or not, e.g. to access files and the Internet.

The Java VM has undergone revisions to ensure that mobile code does not pose a security risk.

Construction of environments for running mobile code in a secure way is generally more difficult than providing secure channels. A different approach is to validate that the mobile code is not harmful. Trusted Network Computing works along these lines.

Information leakage

Information leakage can be particularly difficult to prevent. E.g. a flood of messages to a dealer in a particular market can be a meaningful and useful piece of information; even though the messages are themselves secure.

When the operation of a system and/or its outputs can be observed then there is the potential for information leakage.

E.g. there are many forms of anonymizing networks, where a client can communicate anonymously with a server. However if the client always makes requests on a Thursday afternoon, then the behavior may be observed and the clients' identity may be inferred.

Basically, the system must appear to be random in order for no information to be leaked.

Securing electronic transactions

There are a number of uses of the Internet that require secure transactions:

- Email -- personal information is often transmitted via email, including e.g. credit card details, and in some cases emails are used to authenticate a user, e.g. when a user is signing up to a mailing list.

- Purchase of goods and services -- payments for goods and services commonly happen via a web interface. Digital products are delivered via the Internet.
- Banking transactions -- money can be directly transferred between bank accounts and different kinds of payment services can be used, e.g. BPAY for paying bills.
- Micro-transactions -- many digital goods and services, such as per page reading of a book, usage of a CPU, a single music title, 10 minutes of an Internet radio station, etc, require very low transaction costs since the price for such services may amount to fractions of a cent.

Some example security policies for securing web purchases include:

- Authenticate the vendor to the buyer, so that the buyer is confident that the server is operated by the vendor.
- Ensure that credit card and personal details are transmitted, unaltered and privately, from the buyer to the vendor and that the details are kept private at all times.
- Responses from the vendor, including digital goods and services, should be received by the buyer without alteration or disclosure during transmission.

In this case, authenticating the buyer is not usually required since the vendor is happy so long as the money is made available.

It should be possible for a buyer to complete a secure transaction with a vendor even if there has been no previous contact between buyer and vendor and without the involvement of a third party.

Designing secure systems

Building a completely secure system is akin to building a completely bug-free system.

Known threats can be listed and the designer can show how the distributed system offers security against such threats.

Logs of sensitive system actions can be used to audit and determine if security violations have or are taking place. E.g. a log file can contain whether attempts to use supervisor resources have failed, due to incorrect password.

Costs of implementing a policy mechanism must be balanced against the threat. Costs of attack can be traded, i.e. how much does it cost the attacker in terms of time and resources.

Security should not needlessly inhibit legitimate uses.

Worst-case assumptions and design guidelines

- Interfaces are exposed -- e.g. a socket interface is open to the public, in much the same way as the front door of a house.
- Networks are insecure -- messages can be looked at, copied and falsified.
- Limit the lifetime and scope of each secret -- keys and passwords can be broken given enough time and resources.
- Algorithms and program code are available to attackers -- the bigger and more widely distributed a secret is, the greater the risk of its disclosure. Open source code is scrutinized by many more programmers than closed source code and this helps to find potential security problems before they are taken advantage of.

- Attackers may have access to large resources -- available computing power needs to be predicted into the life time of the system and systems need to be secure against some orders of magnitude beyond this.
 - Minimize the trusted base -- parts of the system that are responsible for enforcing security are trusted, the greater the number of trusted parts the greater the complexity and so the greater risk of errors and misuse.
-

Cryptography

Cryptography

Familiar names for the protagonists in security protocols:

- Alice -- First participant.
 - Bob -- Second participant.
 - Carol -- Participant in three- and four-party protocols.
 - Dave -- Participant in four-party protocols.
 - Eve -- Eavesdropper. *窃听者*
 - Mallory -- Malicious attacker. *恶意攻击者*
 - Sara -- A server.
-

There are two main classes of encryption algorithms: *shared secret key* algorithms and *public/private key* algorithms.

Some common cryptographic notation includes:

- 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 everyone to read).
 - $\{M\}_k$ -- Message M encrypted with key k .
 - $[M]_k$ -- Message M signed with key k .
-

Given an encryption algorithm, E , a decryption algorithm, D , a key, k , and a message, M , $\{M\}_k = E(M, k)$ and $M = D(\{M\}_k, k)$.

If $k = k_A$ is Alice's secret key then $\{M\}_k$ can only be decrypted by Alice using k .

If $k = k_{AB}$ is a secret key shared between Alice and Bob then $\{M\}_k$ can only be decrypted by Alice or Bob using k .

If $k = k_{Apriv}$ is Alice's private key, from a public/private key pair, then $\{M\}_k$ can be decrypted by anyone who has k_{Apub}

If $k = k_{Apub}$ is Alice's public key, from a public/private key pair, then $\{M\}_k$ can only be decrypted by Alice using k_{Apriv}

Private/secret keys should be securely maintained since their use is compromised if an attacker obtains a copy of them.

Public/private key encryption algorithms typically require 100 to 1000 times more processing power than

secret-key algorithms.

Secrecy and integrity

A fundamental policy is one of ensuring secrecy of a message. If Alice and Bob have agreed to a shared key and encryption/decryption algorithm then for a sequence of messages M_1, M_2, \dots :

1. Alice uses k_{AB} and E to encrypt message M_i and sends $\{M_i\}_{k_{AB}}$ to Bob.
2. Bob uses k_{AB} and D to decrypt message $\{M_i\}_{k_{AB}}$.

If the message makes sense when it is decrypted, or better if it contains some agreed upon value such as a checksum, then Bob can be confident that the message is from Alice and that it has not been tampered with.

Some problems:

- How can Alice send a shared key k_{AB} to Bob securely?
- How does Bob know that any $\{M_i\}$ is not a copy that was captured by Mallory and resent to Bob?

Authentication

Consider the case when Alice wants to access a resource held by Bob. Sara is an authentication server that is securely managed. Sara issues passwords to all users including Alice and Bob. Sara knows k_A and k_B because they are derived from the passwords.

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 encrypted using k_A consisting of a ticket encrypted in k_B and a new secret key k_{AB} for use when communicating with Bob: $\{\{Ticket\}_{kB}, k_{AB}\}_{kA}$.
3. Alice decrypts the response using k_A . Alice cannot tamper with the ticket before sending it, because it is encrypted with k_B .
4. Alice sends request R to Bob: $\{Ticket\}_{kB}, Alice, R$.
5. Bob receives the ticket which is actually $Ticket = \{\{k_{AB}\}, Alice\}_{kB}$. Bob decrypts the ticket using his key k_B . Alice and Bob can now communicate using the shared key or session key, k_{AB} .

The previous algorithm is a simplified version of the authentication protocol originally developed by Roger Needham and Michael Schroeder.

Kerberos overview

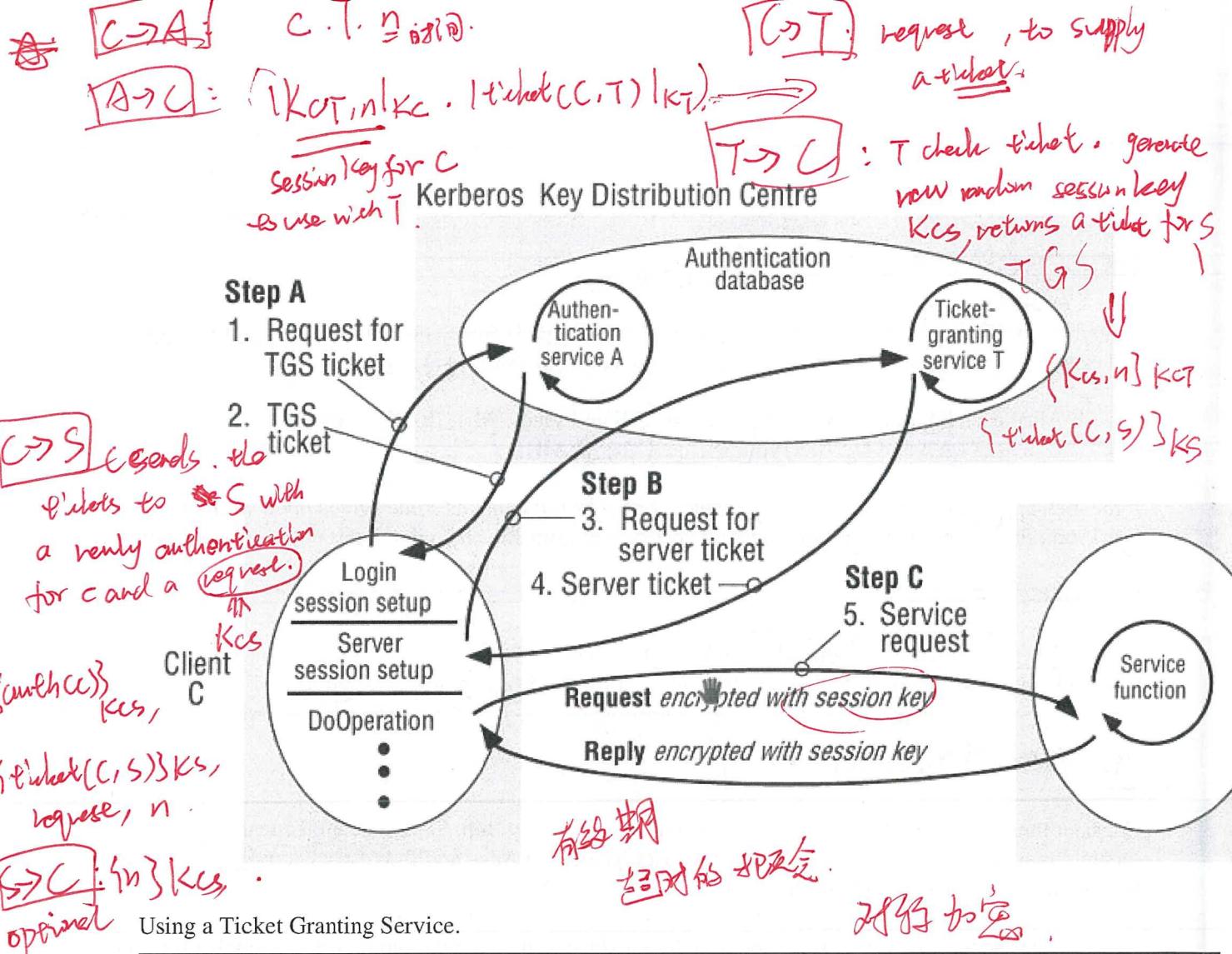
$k_A \rightarrow (k_B \xrightarrow{\text{encrypt ticket}} \text{ticket})$ and $[k_{AB}]$

authentication

$\{\{Ticket\}_{kB}, [k_{AB}]\}_{[k_A]}$

$\{\{Ticket\}_{kB}, Alice, R\}$

$\{\{Ticket\}_{kB}, [k_{AB}]\}_{[k_A]}$



Challenge Response

The use of an authentication server is practical in situations where all users are part of a single organization. It is not practical when access is required between parties that are not supervised by a single organization.

Simple systems like Telnet send passwords from client to server "in the clear". Such passwords are easily compromised by eavesdroppers.

The challenge-response technique is now widely used to avoid sending passwords in the clear. The identity of a client is established by sending the client an encrypted message that only the client should be able to decrypt, this is called a challenge message. If the client cannot decrypt the challenge message then the client cannot properly respond.

Authenticated communication with public keys

1. Alice accesses a key distribution service, Sara, to obtain a public-key certificate giving Bob's public key. The public-key certificate, Cert, is a message signed by Sara using k_{Spriv}. The key k_{Bpub} is widely known by Alice and others and is used to check the signature. Among other things the certificate contains Bob, keyname, k_{Bpub}.
 KBpub
2. Alice creates k_{AB} and encrypts it using k_{Bpub} with a public-key algorithm. She sends the result to Bob, along with a name that identifies the public/private key pair (since Bob may have several public/private keys). Alice sends {keyname, {k_{AB}} } _ {k_{Bpub} }.

3. Bob selects the appropriate private key k_{Bpriv} and decrypts the message to obtain kAB . Alice and Bob can now securely communicate.

If the message from Alice to Bob was tampered with then the decrypted kAB will not match and messages back from Bob will not make sense. Having said this, Alice can also encrypt some additional identification in the original message, e.g. a checksum or Alice's email address, etc.

Digital signature

A digital signature serves the same role as a signature, binding an identity to a message.

In the case of public/private keys, the identity is the public/private key pair itself.

用笔名人知道的 key

A digital signature requires the use of a digest. A digest is a function, $Digest(M)$, that maps an arbitrary message, M , to a fixed length datum. The digest function must be such that a given datum is very unlikely to be mapped to from two different messages. The SHA-1 hash function is a good example of this.

If Alice wants to sign a message, M , then Alice constructs $M, \{Digest(M)\}_{kApri}$.

A receiver, Bob, decrypts the digest using $kApub$. Bob also computes the digest of M locally. If the message or the encrypted digest were tampered with then the results will not match.

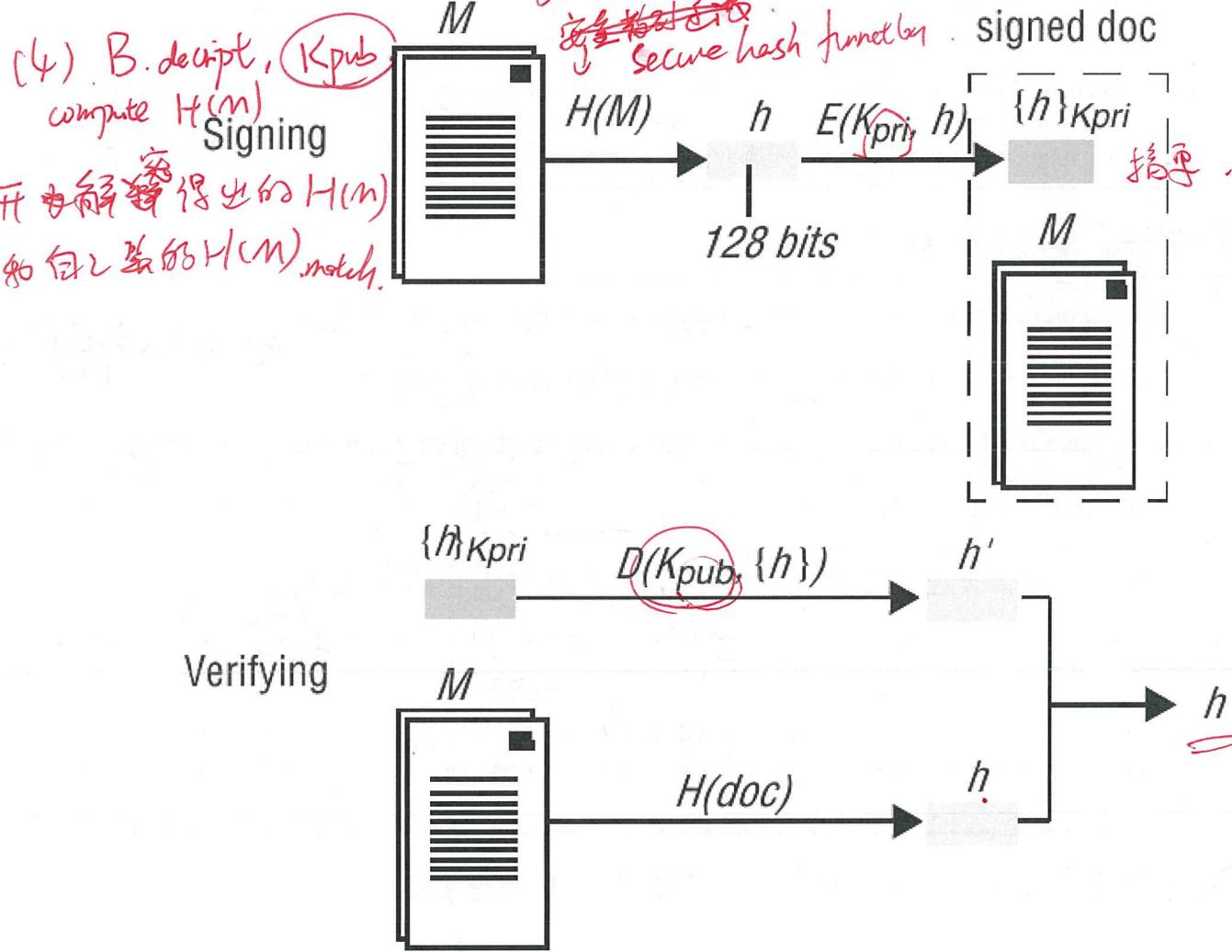
This is effectively a signature based on the identity $kApri$ since no other private key would produce that encrypted digest and no other message is likely to produce that digest. Alice cannot deny that she signed the message.

Digital Signature with pub/priv keys

(1) $A \rightarrow K_{pub}, K_{priv}$

(2) Compute digest of M $H(M)$ + K_{priv} to produce signature

(3) A send signed message to B



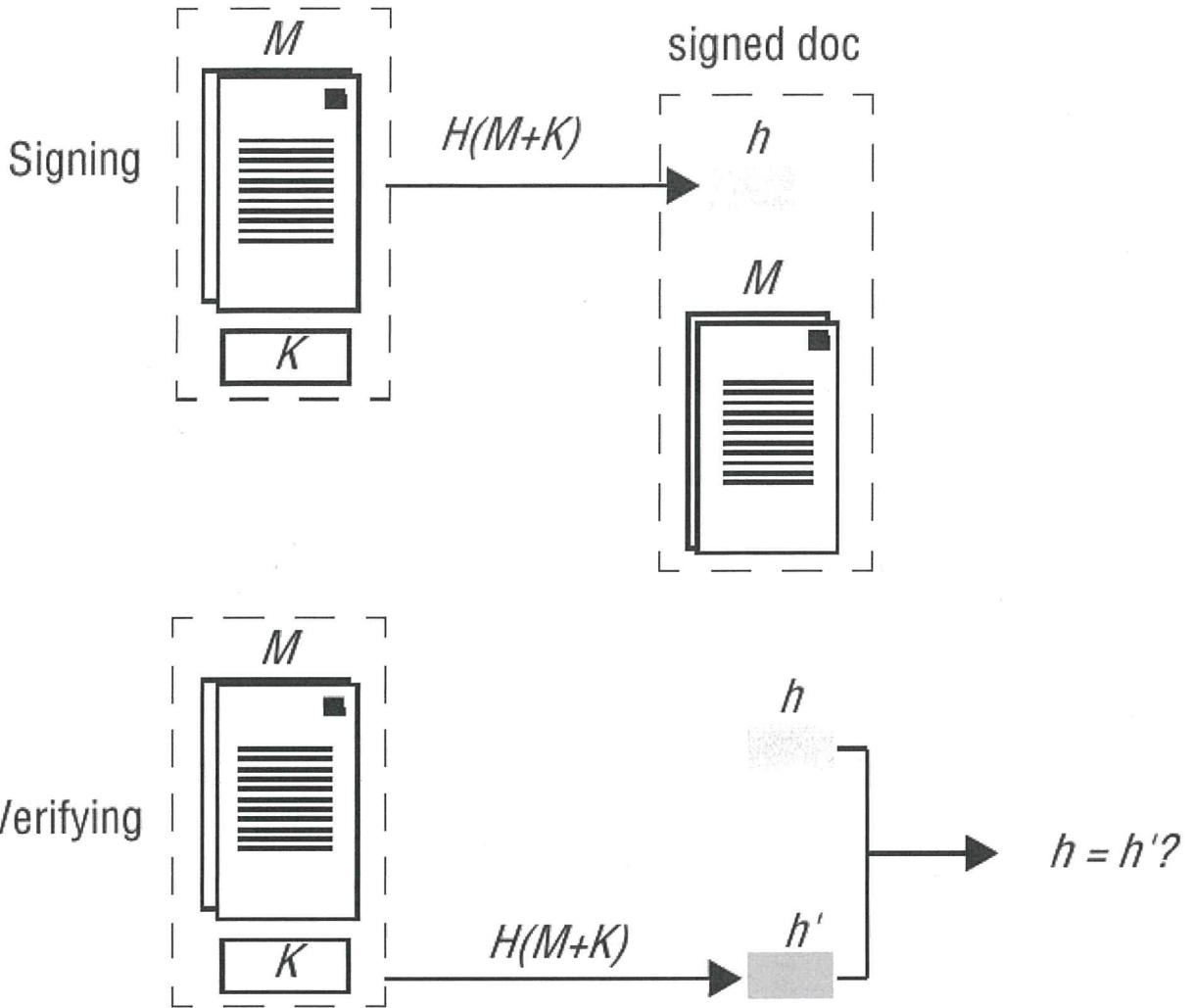
Digital Signature with shared key

MAC

Message Authentication Code

Key 共同通道分发

不涉及加密，快 3-10 倍



Certificates

A digital certificate is a document containing a statement that is signed by a principal. For the certificate to be useful, the principal's public key must be known to anyone who wants to authenticate the certificate. The principal is otherwise known as the certifying authority.

In the previous examples a certificate was used to associate a public key with an identity. In this case the certificate could be:

- Certificate type: Public key
- Name: Bob
- Public key: **kBpub**
- Certifying authority: Sara
- Signature: {Digest(field 2+field 3)}_{kSpriv}

The certificate is effectively saying, "Sara certifies that Bob's public key is **kBpub**."

Sara cannot deny that she has attested to this fact.

Certificate chains

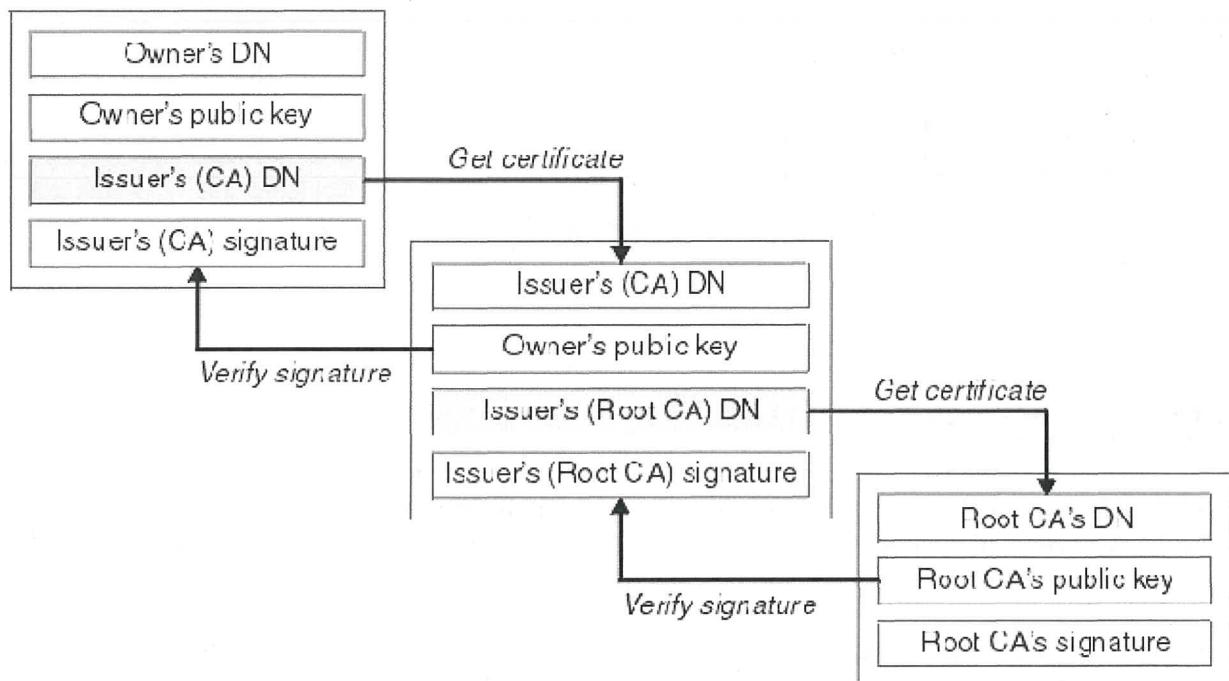
For Alice to authenticate a certificate from Sara concerning Bob's public key, Alice must first have Sara's public key. This poses a recursive problem.

In the simplest case, Sara creates a *self-signed* certificate, which is attesting to her own public key. The self-signed certificate is widely publicized (e.g. by distributing with operating system or browser installation). This certificate is trusted. The private key must be closely guarded in order to maintain the integrity of all certificates that are signed with it.

However, assume that Carol has signed a certificate attesting to Bob's public key and that Sara has signed a certificate attesting to Carol's public key. This is an example of a *certificate chain*. If Alice trusts Carol's certificate then she can authenticate Bob's identity. Otherwise Alice must first authenticate Carol's identity using Sara's certificate.

Revoking a certificate is usually by using predefined expiry dates. Otherwise anyone who may make use of the certificate must be told that the certificate is to be revoked.

From <http://publib.boulder.ibm.com>.



A standard for digital certificates is X.509. From Wikipedia, the structure of an X.509 version 3 certificate is:

- Certificate
 - ◆ Version
 - ◆ Serial Number
 - ◆ Algorithm ID
 - ◆ Issuer
 - ◆ Validity
 - ◊ Not Before
 - ◊ Not After

- ◆ Subject
- ◆ Subject Public Key Info
 - ◊ Public Key Algorithm
 - ◊ Subject Public Key

cont...

- (cont...)
 - ◆ Issuer Unique Identifier (Optional)
 - ◆ Subject Unique Identifier (Optional)
 - ◆ Extensions (Optional)
 - Certificate Signature Algorithm
 - Certificate Signature
-

```

Certificate:
Data:
Version: 3 (0x2)
Serial Number: 1 (0x1)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
        OU=Certification Services Division,
        CN=Thawte Server CA/Email=server-certs@thawte.com
Validity
    Not Before: Aug 1 00:00:00 1996 GMT
    Not After : Dec 31 23:59:59 2020 GMT
Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
        OU=Certification Services Division,
        CN=Thawte Server CA/Email=server-certs@thawte.com
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
        Modulus (1024 bit):
            00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c:
            68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da:
            85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06:
            6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2:
            6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b:
            29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90:
            6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f:
            5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36:
            3a:c2:b5:66:22:12:d6:87:0d
        Exponent: 65537 (0x10001)

```

cont...

```

X509v3 extensions:
X509v3 Basic Constraints: critical
    CA:TRUE
Signature Algorithm: md5WithRSAEncryption
    07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9:
    a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48:
    3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88:
    4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9:
    8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5:
    e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9:
    b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e:
    70:47

```

Cryptographic algorithms

symmetric

Secret key cryptography is often referred to as symmetric cryptography. Public/private key cryptography is often referred to as asymmetric.

Block ciphers operate on fixed-size blocks of data; 64 bits is a popular size for the blocks. A message is subdivided into blocks and the last block is padded to the standard length. Each block is encrypted independently. A block is transmitted as soon as it is encrypted.

块技术 = 顺序发送

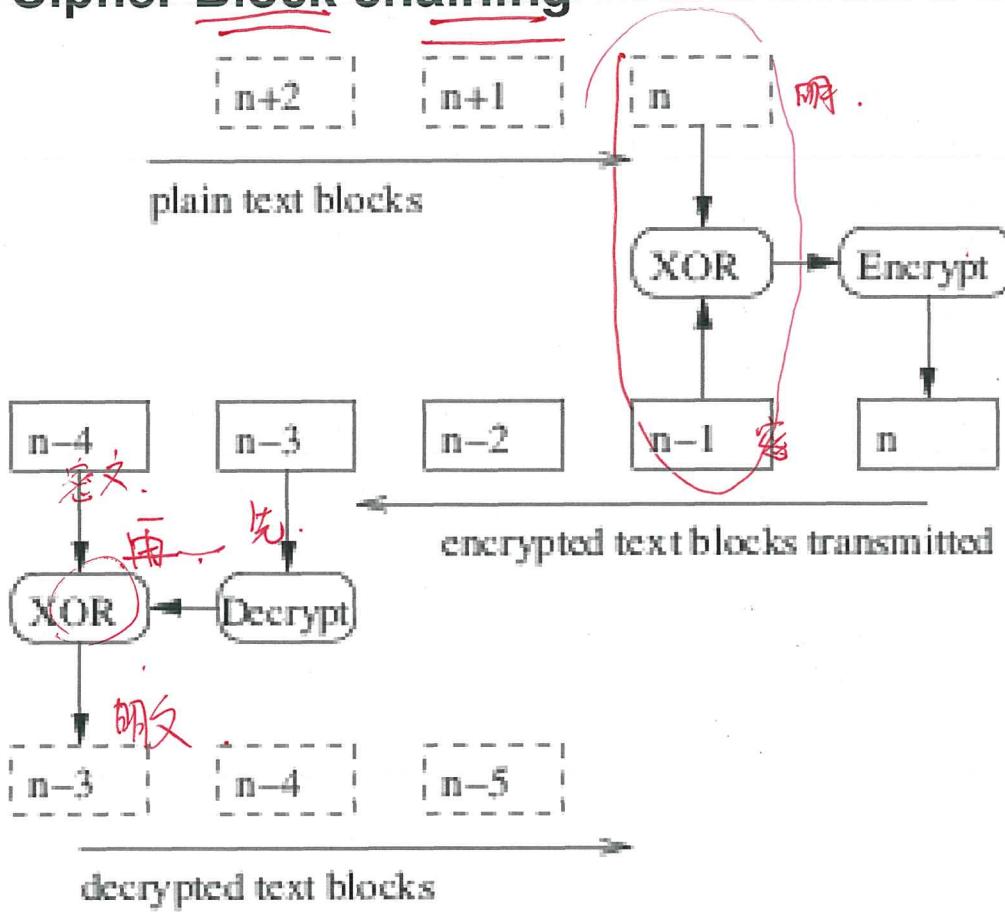
Cipher block chaining avoids the problem of identical plain text blocks encrypted to identical encrypted blocks. However, if the same message is sent to different recipients then it will still look the same and this poses an information leakage weakness. To guard against this a block called an initialization vector is used to start each message in a different way.

XOR

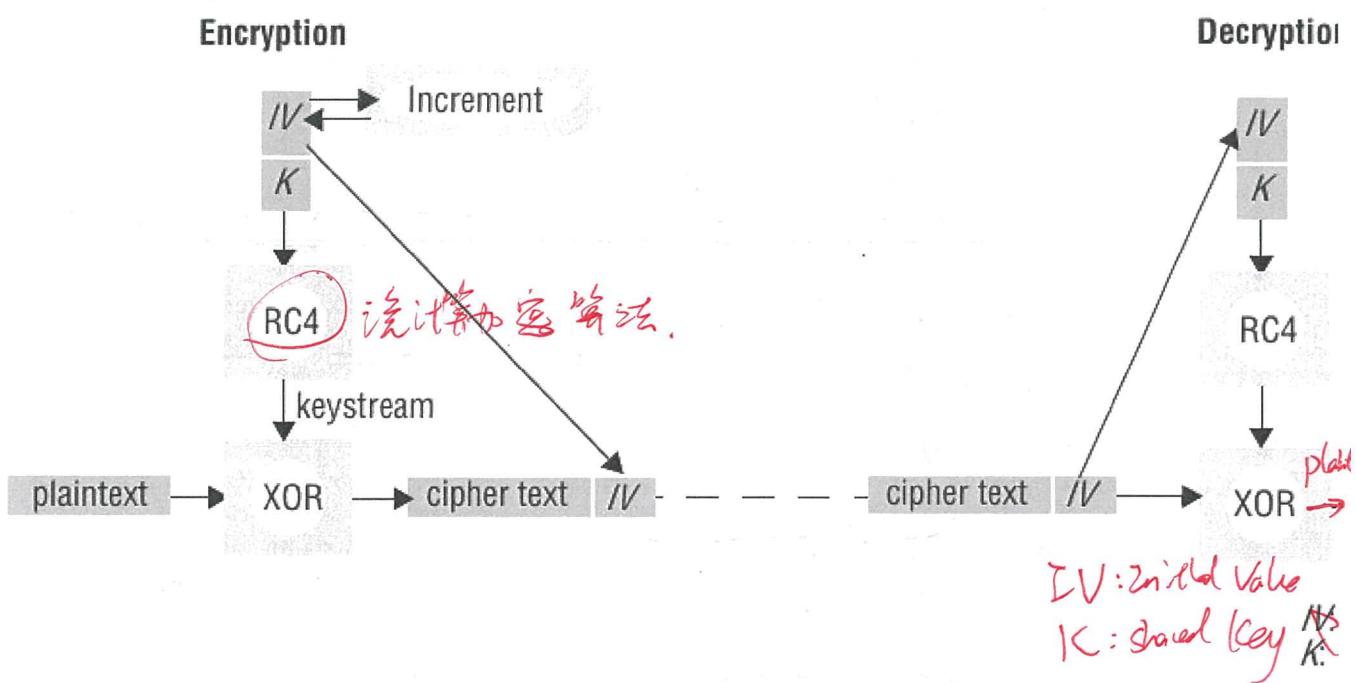
必须可逆的传输，不能丢失

Stream ciphers are used when the data cannot be easily divided into blocks. In this case, an agreed upon key stream (such as from a random number generator with known seed) is encrypted and the output is XOR'ed with the data stream.

Cipher Block chaining



Streaming Cipher in IEEE 802.11 WEP



WEP (Wired Equivalent Privacy) security is seriously flawed and everyone should now be using at least WPA2 (WiFi Protected Access version 2).

Some symmetric algorithms include:

- **TEA**: Tiny Encryption Algorithm, and subsequently the Extended (XTEA) version that guards against some minor weaknesses. The algorithm uses 128 bit keys to encrypt 64 bit blocks. The algorithm consists of only a few lines of code (hence the name), is secure and reasonably fast.
- **DES**: Data Encryption Standard uses a 56 bit key to encrypt a 64 bit block. This algorithm is now considered obsolete because it is too weak. The triple DES version, 3DES, is stronger but takes a long time to encrypt.
- **IDEA**: International Data Encryption Algorithm uses a 128 bit key to encrypt 64 bit blocks. It is a faster and more secure successor to DES.
- **RC4**: A stream cipher that uses keys of any length up to 256 bytes. About 10 times as fast as DES and was widely used in WiFi networks until a weakness was exposed.
- **AES**: Advanced Encryption Standard has variable block length and key length with specifications for keys with a length of 128, 192 or 256 bits to encrypt blocks with length of 128, 192 or 256 bits. Block and key lengths can be extended by multiples of 32 bits.

The most widely known asymmetric algorithm is RSA or the Rivest, Shamir and Adelman algorithm.

RSA is based on the use of the product of two very large prime numbers (greater than 10^{100}). Its strength comes from the fact that the determination of the prime factors of such large numbers is very computationally expensive.

There are no known flaws in RSA.

One potential weakness of all public-key algorithms is that an attacker, having the public key, can try to decrypt a message by encrypting all possible messages and finding the one that matches the target message. Such an attack can be defended against by only encrypting messages that have at least as many bits as the key, hence the attack is no better than trying all possible keys.

Secure socket layer

SSL

The Secure Socket Layer and its successor the Transport Layer Security (TLS) protocol are intended to provide a flexible means for clients and server to communicate using a secure channel.

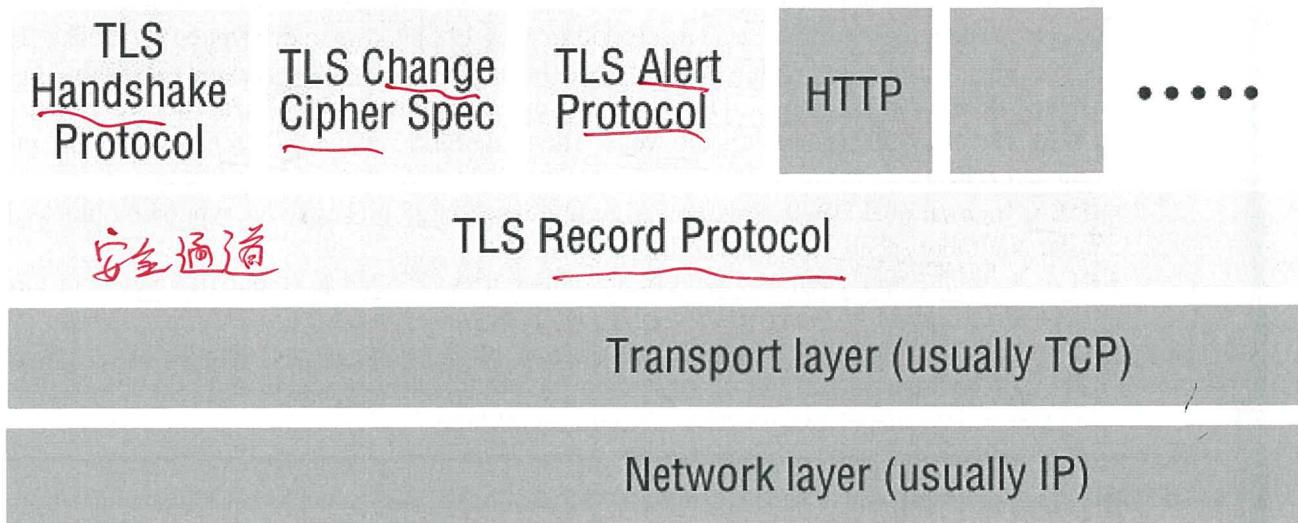
In typical use, the server is authenticated while the client remains unauthenticated. The protocols allow client/server applications to communicate in a way designed to prevent eavesdropping, tampering and message forgery.

There are three basic phases:

- Peer negotiation for algorithm support.
- Public key encryption-based key exchange and certificate-based authentication.
- Symmetric cipher-based traffic encryption.

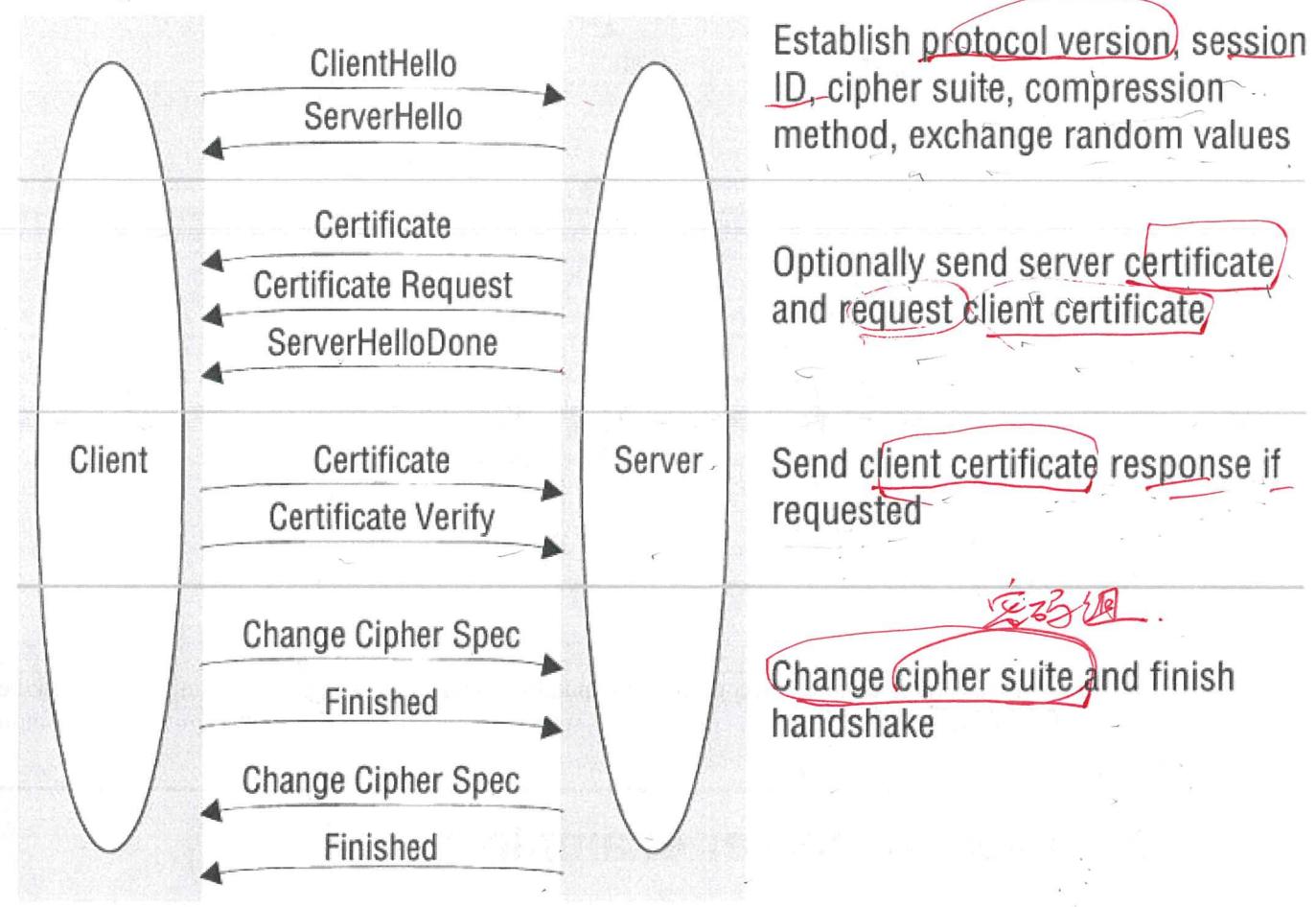
Generally the protocol uses a Record Protocol layer that exchanges records; each record can be optionally compressed, encrypted and packed with a message authentication code (a signature that uses a shared secret key). Each record has a type that specifies an upper level protocol including Handshake, Change Cipher Spec and Alert Protocol.

TLS protocol stack



TLS record protocol

TLS handshake



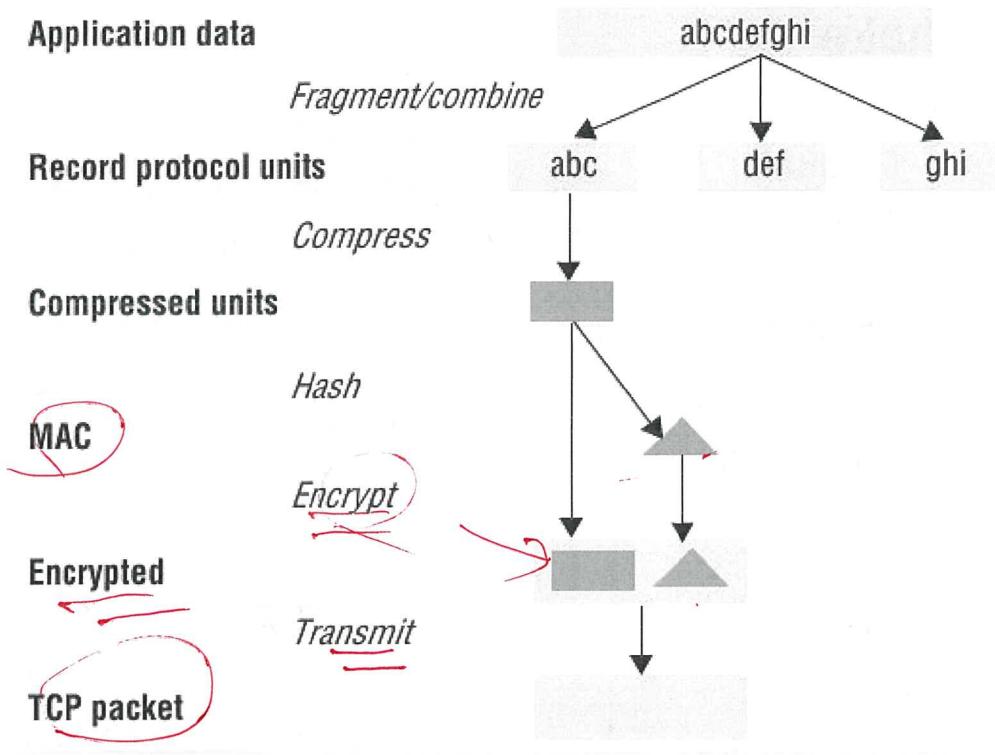
The first phase allows the client and server to establish which cipher, compression method and other connection parameters are to be used.

The second phase exchanges certificates. A master secret or common secret is negotiated and this is used to generate all other key data for the purpose of encryption.

Security measures include:

- Numbering all records and including the sequence numbers in the signatures.
- The message that ends the handshake sends a hash of all the exchanged data seen by both parties.
- Hashing is done by combining (XORing) the results of both MD5 and SHA, in case one is found to be vulnerable.

TLS record protocol



Optional compression is included because the computations can share work with the encryption and thereby save work overall; i.e. it is faster than compressing separately and then securely transmitting the compressed data.

Java secure server example

When using Java secure server (and client) a keystore file is needed, that contains the certificates and other data required for SSL.

```

/*
 * from www.javaworld.com */
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.SSLSocketFactory;

public class EchoServer
{
    public static void main(String [] args)
    {
        try
        {
            SSLSocketFactory sslsocketfactory =
                (SSLSocketFactory)
                SSLSocketFactory.getDefault();
            SSLSocket sslsocket =
                (SSLSocket)
                sslsocketfactory.createServerSocket(9999);
            SSLServerSocket sslsocket = (SSLServerSocket)

```

```
sslserversocket.accept();

---



```
InputStream inputstream = sslsocket.getInputStream();
InputStreamReader inputstreamreader =
 new InputStreamReader(inputstream);
BufferedReader bufferedreader =
 new BufferedReader(inputstreamreader);
String string = null;
while ((string = bufferedreader.readLine()) != null)
{
 System.out.println(string);
 System.out.flush();
}
}
catch (Exception exception)
{
 exception.printStackTrace();
}
}
}

import java.io.InputStream;
import java.io.OutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;

import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;

public class EchoClient
{
 public static void main(String [] arstring)
 {
 try
 {
 SSLSocketFactory sslsocketfactory =
 (SSLSocketFactory)SSLSocketFactory.getDefault();
 SSLSocket sslsocket =
 (SSLSocket)
 sslsocketfactory.createSocket("localhost", 9999);

 InputStream inputstream = System.in;
 InputStreamReader inputstreamreader =
 new InputStreamReader(inputstream);
 BufferedReader bufferedreader =
 new BufferedReader(inputstreamreader);

 OutputStream outputstream = sslsocket.getOutputStream();
 OutputStreamWriter outputstreamwriter =
 new OutputStreamWriter(outputstream);
 BufferedWriter bufferedwriter =
 new BufferedWriter(outputstreamwriter);

 String string = null;
 while ((string = bufferedreader.readLine()) != null)
 {
 bufferedwriter.write(string + '\n');
 bufferedwriter.flush();
 }
 }
 }
}
```



---


```

```
        }
    catch (Exception exception)
    {
        exception.printStackTrace();
    }
}
```