Application-Layer Security

Content

- **Email Security**
- **Centralized Authentication: NTLM, Kerberos**
- **SSH**

Email Security

Outline

- Email Security
- PGP Overview
- PGP Operational Description
- PGP Key Generation and Key Rings
- PGP Public-Key Management
- MIME (& RFC 822)
- S/MIME

1. Email Security

Email (electronic mail):

- An email message is made up of a string of ASCII characters in a format specified by RFC 822.
- Then, such a message travels to the recipient via Internet.
- Email is a widely used network-based application.
- Moreover, email is the only distributed application that is widely used across all architectures and platforms.
- Email is very popular mainly due to its convenience.

1. Email Security

However, basic email has very weak security:

- **Lack of Confidentiality**
  - Sent in clear over open networks.
  - Stored on potentially insecure clients and servers.
- **Lack of Integrity**
  - Both the header and content can be modified.
- **Lack of Authentication**
  - The sender of an email is also forgeable.
- **Lack of Non-Repudiation**
  - The sender can later deny having sent an email.
  - The recipient can later deny having received the message.

1. Email Security

In this lecture, we are going to discuss email security

- **PGP: Pretty Good Privacy (https://www.openpgp.org/)**
- **S/MIME: Secure/Multipurpose Internet Mail Extensions**

2. PGP Overview

Basically, PGP provides confidentiality and authentication services to enhance the security for email transmission and storage.

- Developed by Philip Zimmermann.

- PGP and OpenPGP operations are specified in a few documents (RFC 2015, 3156, 4880).

2. PGP Overview

Summary of PGP Services

Function	Algorithms Used
Digital Signature (Authentication)	DSS/SHA or RSA/SHA
Message Encryption	CAST, IDEA, 3DES, AES, RSA, ElGamal, etc.
Compression	ZIP
E-mail Compatibility	Radix-64 conversion
Segmentation	-

3. PGP Operational Description

Operational Description

- Authentication
- Confidentiality
- Confidentiality and Authentication
- Email Compatibility
- Segmentation and Reassembly

3. PGP Operational Description

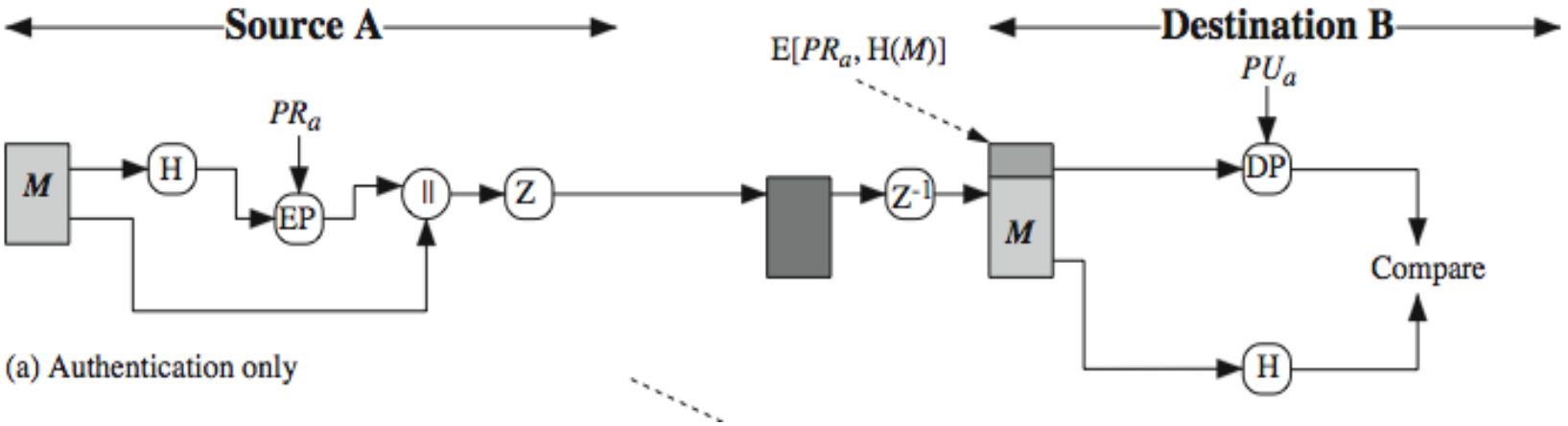
Notations

Ks: one-time session key
PRa: private key of user A
PUa: public key of user A
EP: encrypting with PU or signing with PR
DP: decrypting with PR or verifying with PU
EC: symmetric encryption
DC: symmetric decryption
H: hash function
||: concatenation
Z: compression using ZIP algorithm
R64: conversion to radix 64 ASCII format

3. PGP Operational Description

Authentication only (RSA-SHA1):

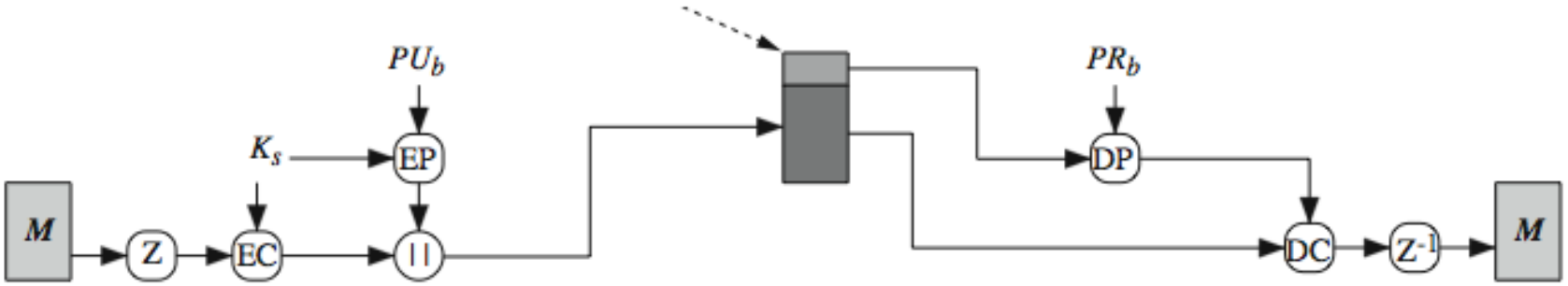
1. Sender creates a message and its SHA1 160-bit hash
2. Sender signs the hash with RSA and prepends the signature to the message
3. Receiver hashes the message and verifies the signature



3. PGP Operational Description

Confidentiality only:

1. sender generates a 128-bit random session key
2. encrypts message with session key
3. attaches session key encrypted with RSA/ElGamal
4. receiver decrypts & recovers session key
5. session key is used to decrypt message

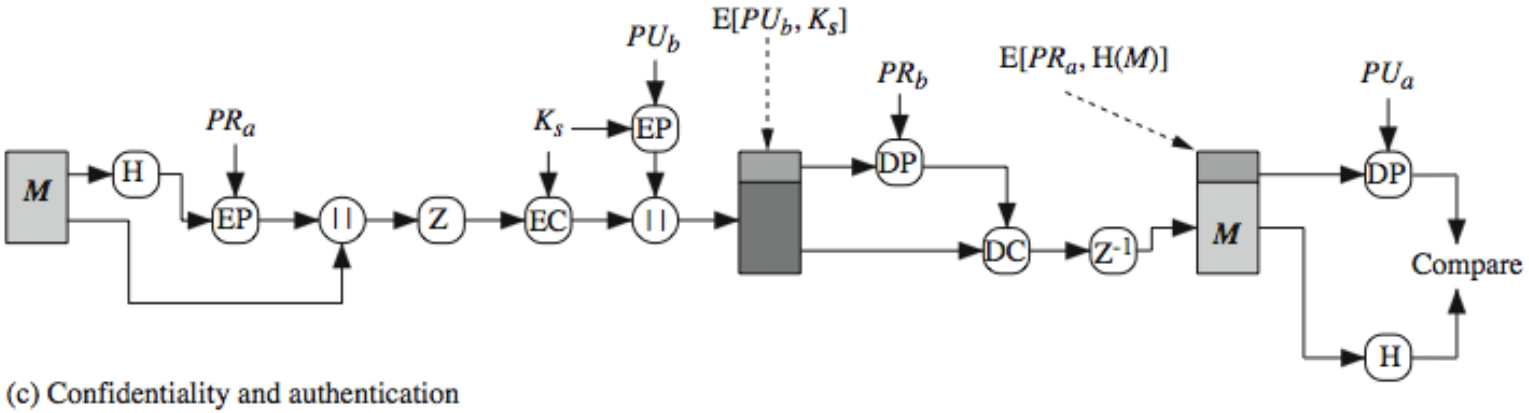


(b) Confidentiality only

3. PGP Operational Description

Confidentiality and Authentication:

- can use both services on same message
  - create signature & attach to message
  - encrypt both message & signature
  - attach RSA/ElGamal encrypted session key



(c) Confidentiality and authentication

3. PGP Operational Description

Compression: Using ZIP.

- The order of operations: sign→compress→encrypt.
- More convenient to store a signature with plain message.

Q: what about encrypt then sign?

3. PGP Operational Description

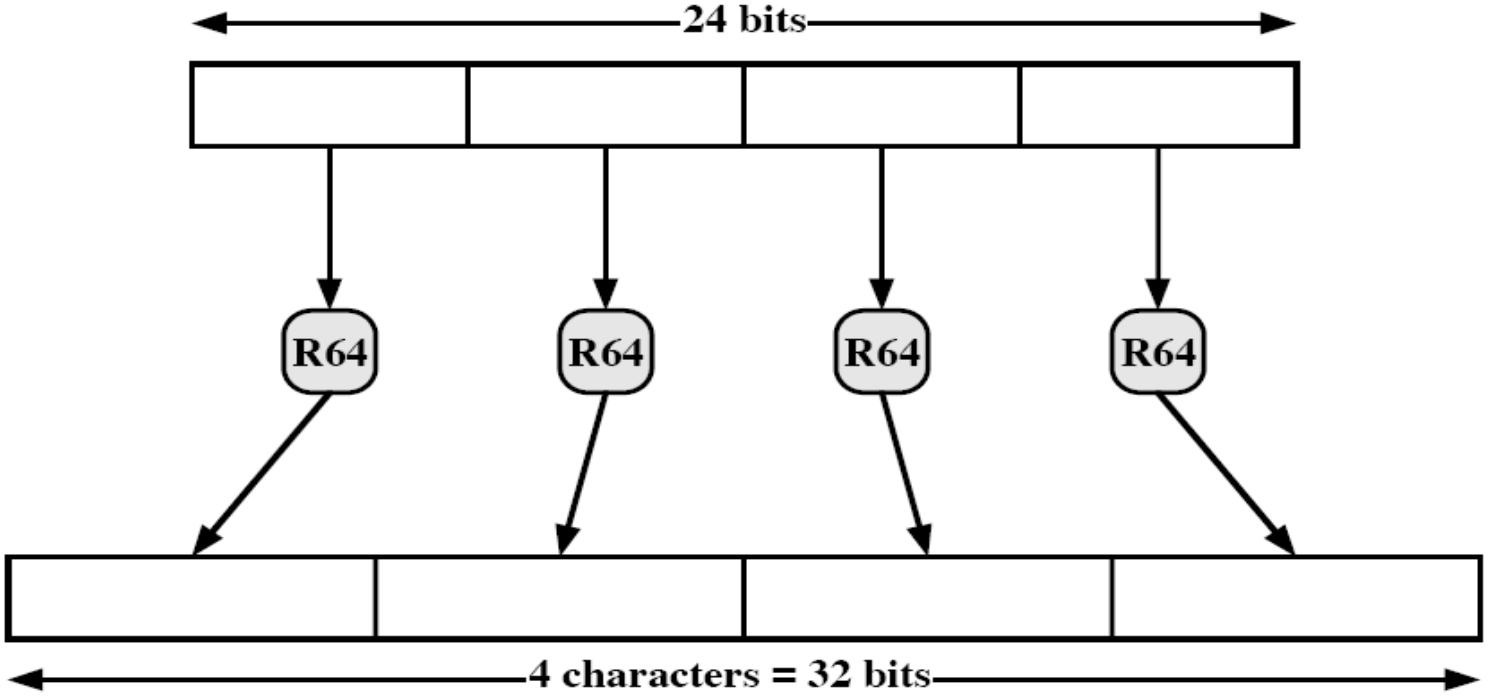
Email Compatibility:

- After the above security operations, the resulting message will contain some arbitrary octets.

- PGP needs to convert the raw 8-bit binary stream into a stream of **printable** ASCII characters.

3. PGP Operational Description

- For this purpose, the radix-64 conversion is used.
- This operation expands the message by 33%.



3. PGP Operational Description

Segmentation and Reassembly:

- Email systems often limit the size of a message up to 50,000 octets.
- So, a longer message must be broken up into segments.
- After all other operations, PGP automatically subdivides a long message into small segments.
- Once getting those emails, the receiver first strips of all email headers and reassemble the block, and then perform other processing.

4. Key Generation & Key Rings

Key Generation:

- RSA & RSA
- DSA & ElGamal
- RSA (sign only)
- DSA (sign only)
- Each session key (for encrypting the real email message) is only associated with one message.

4. Key Generation & Key Rings

Key Identifiers (Key IDs):

- One user may use multiple public/private key pairs.
- How to let the receiver know which key pair is used?
- Trivial approach
  - **Receiver tries each possible public key**
- PGP uses the **key ID** to identify a public key.
  - **Key ID = (PUa mod 2^{64}), i.e., the least significant 64 bits of the key fingerprint.**

4. Key Generation & Key Rings

Key Rings:

- Each user maintains two key rings in his/her system.
- **A private-key ring** stores the private/public key pairs owned by the user.
- **A public-key ring** stores the public keys of other users.

Next slide shows the structures of these two key rings.

4. Key Generation & Key Rings

Private Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
T_i	$PU_i \bmod 2^{64}$	PU_i	$E(H(P_i), PR_i)$	User i
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

Public Key Ring

Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
T_i	$PU_i \bmod 2^{64}$	PU_i	$trust\_flag_i$	User i	$trust\_flag_i$		
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•

* = field used to index table

Figure 15.4 General Structure of Private and Public Key Rings

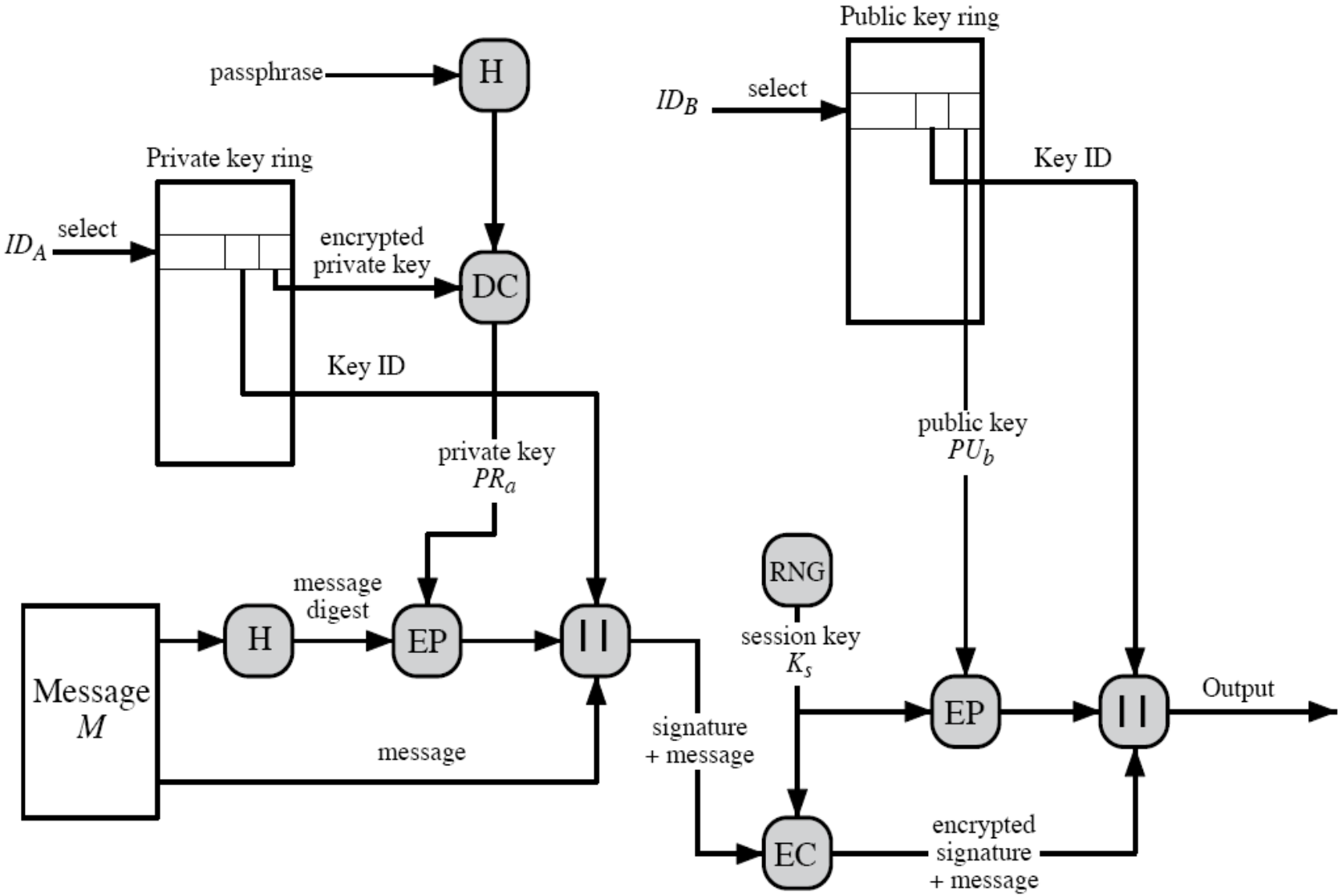
4. Key Generation & Key Rings

In the above diagram, Pi is the user's password.

- **Security of private keys thus depends on the pass-phrase security**

Next two slides showing:

- **The Procedures to Generate a PGP Message**
- **The Procedures to Receive a PGP Message**


passphrase
H
Private key ring
ID_A
select
encrypted
private key
DC
Key ID
private key
PR_a
message
digest
H
EP
Message
M
message
RNG
session key
K_s
signature
+ message
EC
Public key ring
ID_B
select
Key ID
public key
PU_b
EP
Output
encrypted
signature
+ message

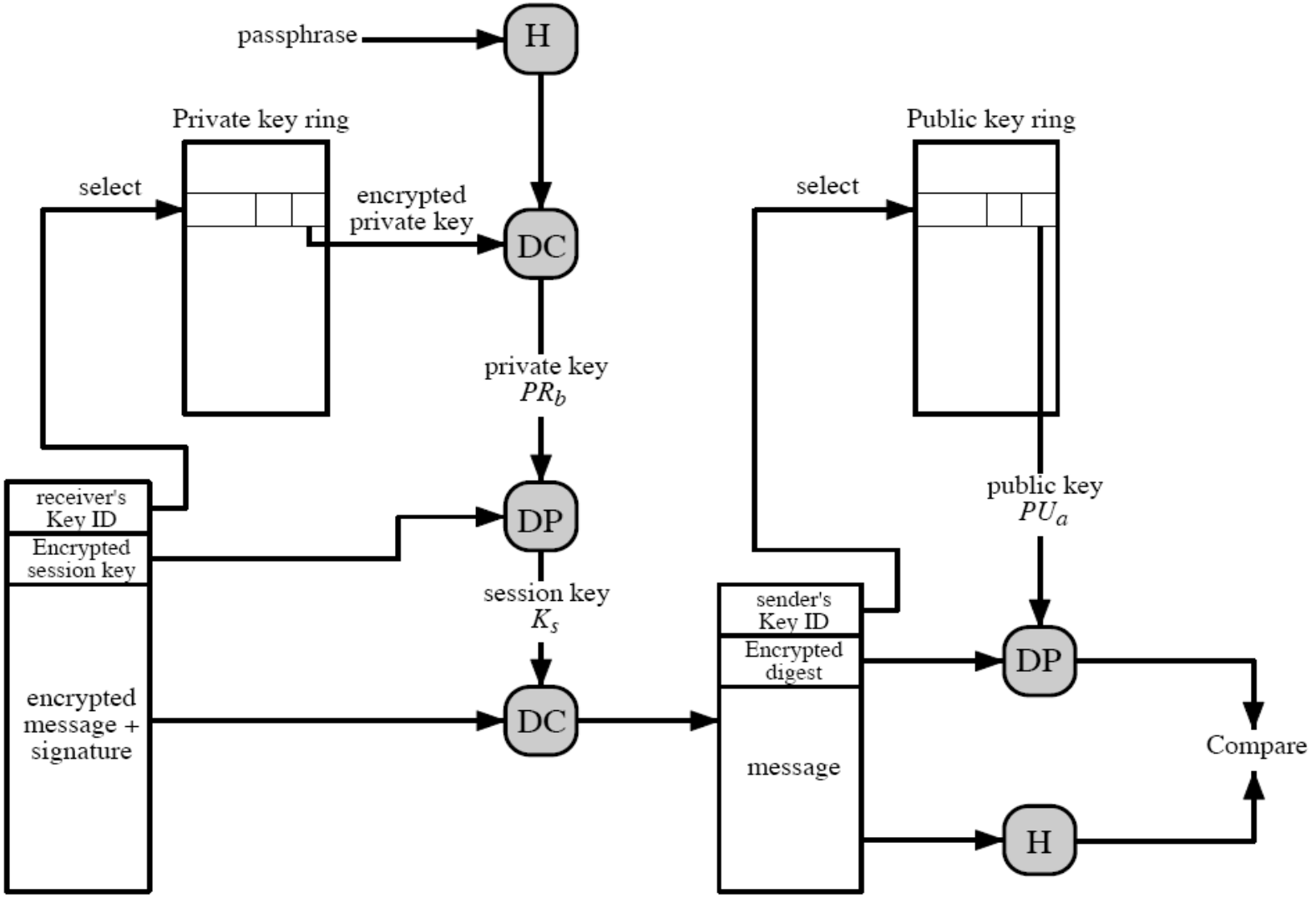


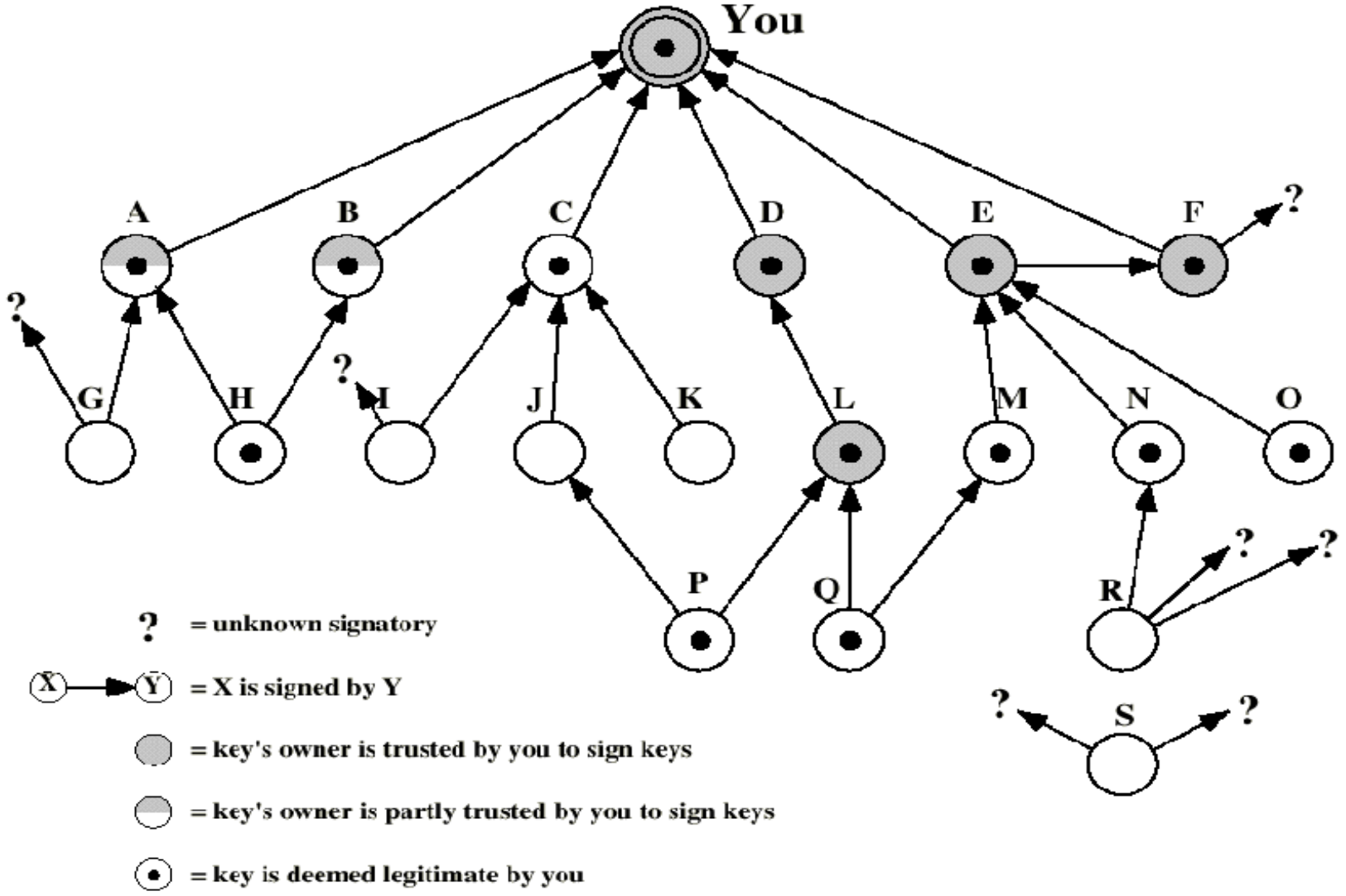
Figure 15.6 PGP Message Reception (from User A to User B; no compression or radix 64 conversion)

5. PGP Public Key Management

- In X.509, public keys are certified by trusted CAs.
- PGP uses a completely different model – **the web of trust**.

- Each PGP user assigns **a trust level to other users (Owner Trust Field)**.
- Each user can **certify** (i.e., sign) the public keys of users he/she knows.
- In the public key ring, each entry stores a number of signatures that **certify** this public key.
- PGP automatically computes **a trust level for each public key (Key Legitimacy Field)** in the key ring.

5. PGP Public Key Management



5. PGP Public Key Management

- **(X)→(Y)** means that X's public key is signed by Y.
- **A shading circle** shows a user (owner of the key) that is trusted by you. So, you trust all public keys certified by this user.
- **A half shading circle** shows a user is partially trusted by you. A public key is also trusted if it has been certified by at least two partially trusted users.
- **A solid dot** shows that the public key is trusted by you.

RFC 822

- **S/MIME** (Secure/Multipurpose Internet Mail Extensions)
  - A security enhancement to MIME email
  - based on technology from RSA Data Security (Now, the Security Division of EMC Corporation).
  - specified by RFCs 3369, 3370, 3850 and 3851.
- **To understand S/MIME, we need first to know MIME.**

RFC 822

- RFC 822 defines a format for Internet-based text mail message.
- In RFC 822, each email is viewed as having **an envelope and content**.
- The envelope contains all information needed for email transmission and delivery.
- RFC 822 applies only to the contents.
- **The content** has two parts, separated by a blank line:
  - **The header:** Date, From,To, Subject, …
  - **The body:** containing the actual message.

6. MIME

MIME is intended to avoid a number limitations in RFC 822:

- **Extends the capabilities of RFC 822 to allow email to carry messages with non-textual content and non-ASCII character sets.**
- **Supports long message transfer.**
- **Introduces new header fields in RFC 822 email to specify the format and content of extensions.**
- **Supports a number of content types together with a number of encoding schemes.**
- **Specified in RFCs 2045-2049.**

6. MIME: Content-Transfer-Encoding

- RFC 822 emails can contain only ASCII characters.
- MIME messages are intended to transport arbitrary data.
- The Content-Transfer-Encoding field indicates how data was encoded from raw data to ASCII.
- Base64 (i.e Radix-64) is a common encoding:

  - 24 data bits (3 bytes) are encoded into 4 ASCII characters (4 bytes).

7. S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extensions):

- **A security enhancement to MIME email.**
- **Specified by RFCs 3369, 3370, 3850 and 3851.**
- **Widely supported in many email agents:**

  - MS Outlook, Mozilla, Mac Mail, Lotus Notes etc.

7. S/MIME

S/MIME

- Functions
- Algorithms
- Processing
- Certificate management

7. S/MIME: Functions

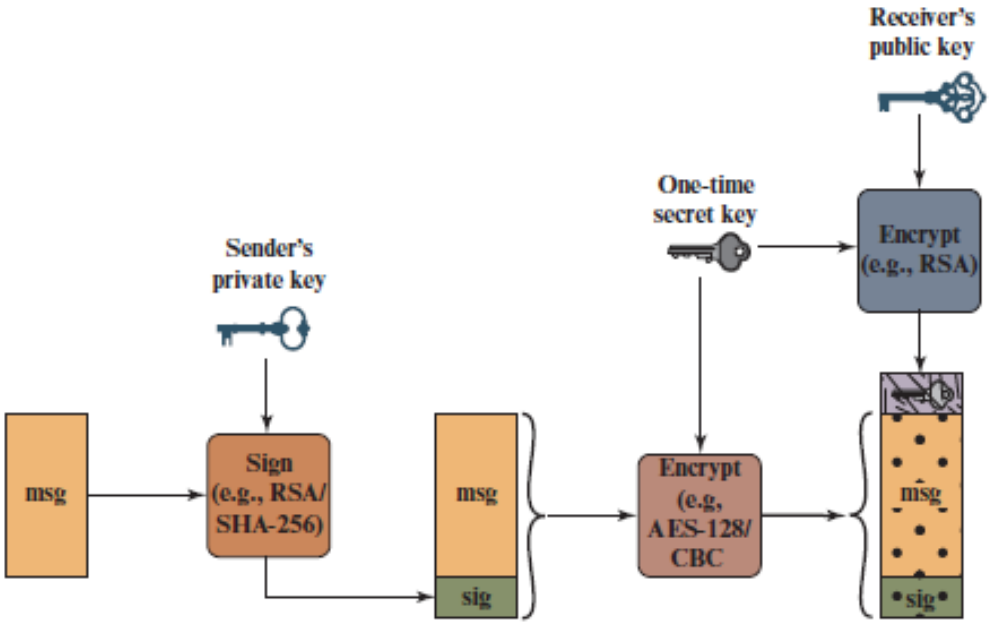
Similar to PGP, S/MIME provides the following functions to secure email:

- **Enveloped Data:** encrypted-only.
- **Signed Data:** signed-only.
- **Signed and Enveloped:** nesting of signed and encrypted entities.

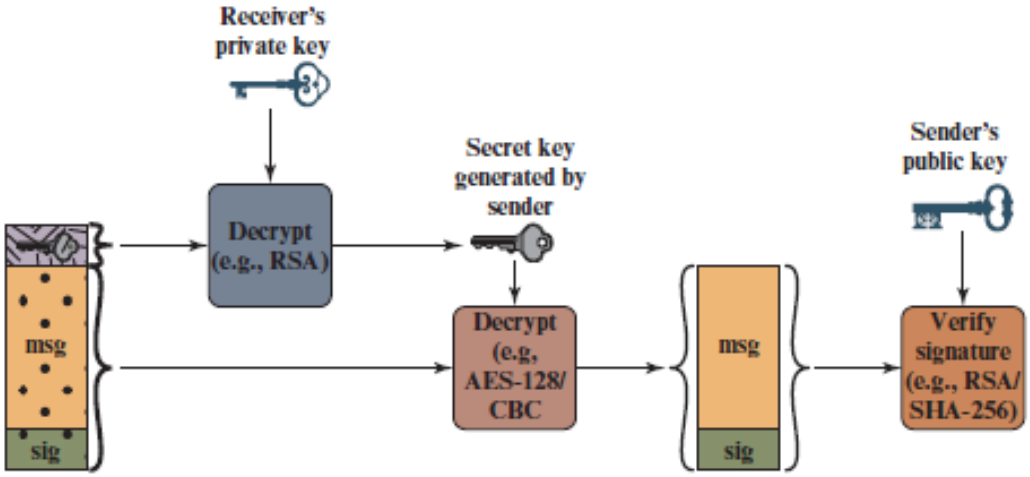
7. S/MIME: Algorithms

S/MIME supports the following algorithms.

- digital signatures: DSS & RSA
- session key encryption: ElGamal & RSA
- message encryption: AES, Triple-DES, and others
- MAC: HMAC with SHA



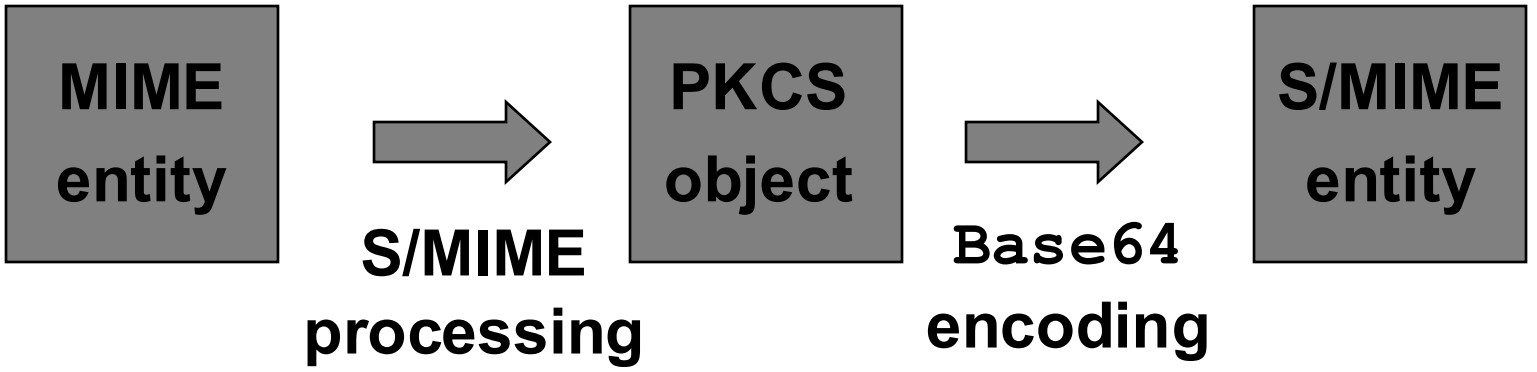
(a) Sender signs, then encrypts message



(b) Receiver decrypts message, then verifies sender's signature

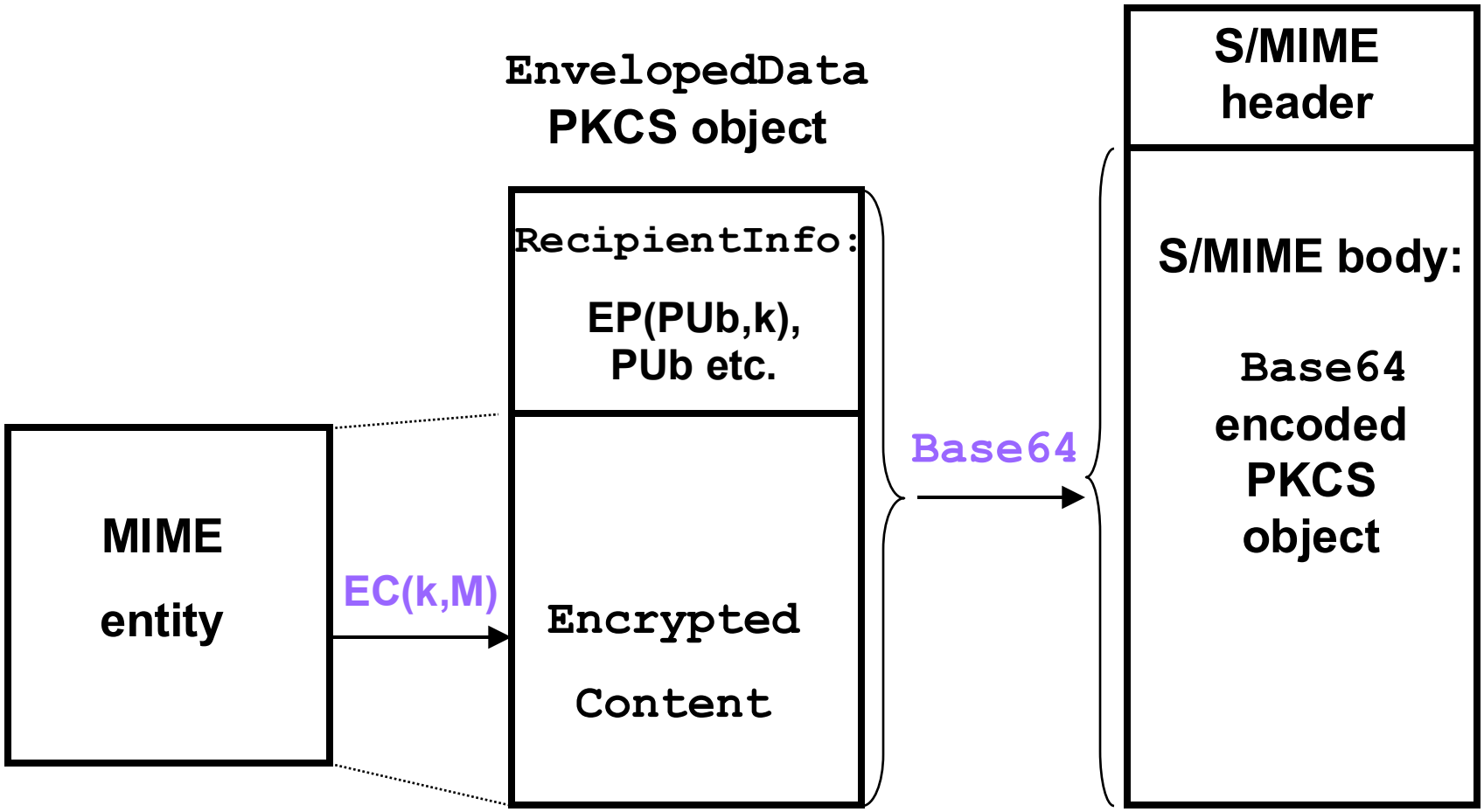
Figure 19.3 Simplified S/MIME Functional Flow

7. S/MIME: Processing

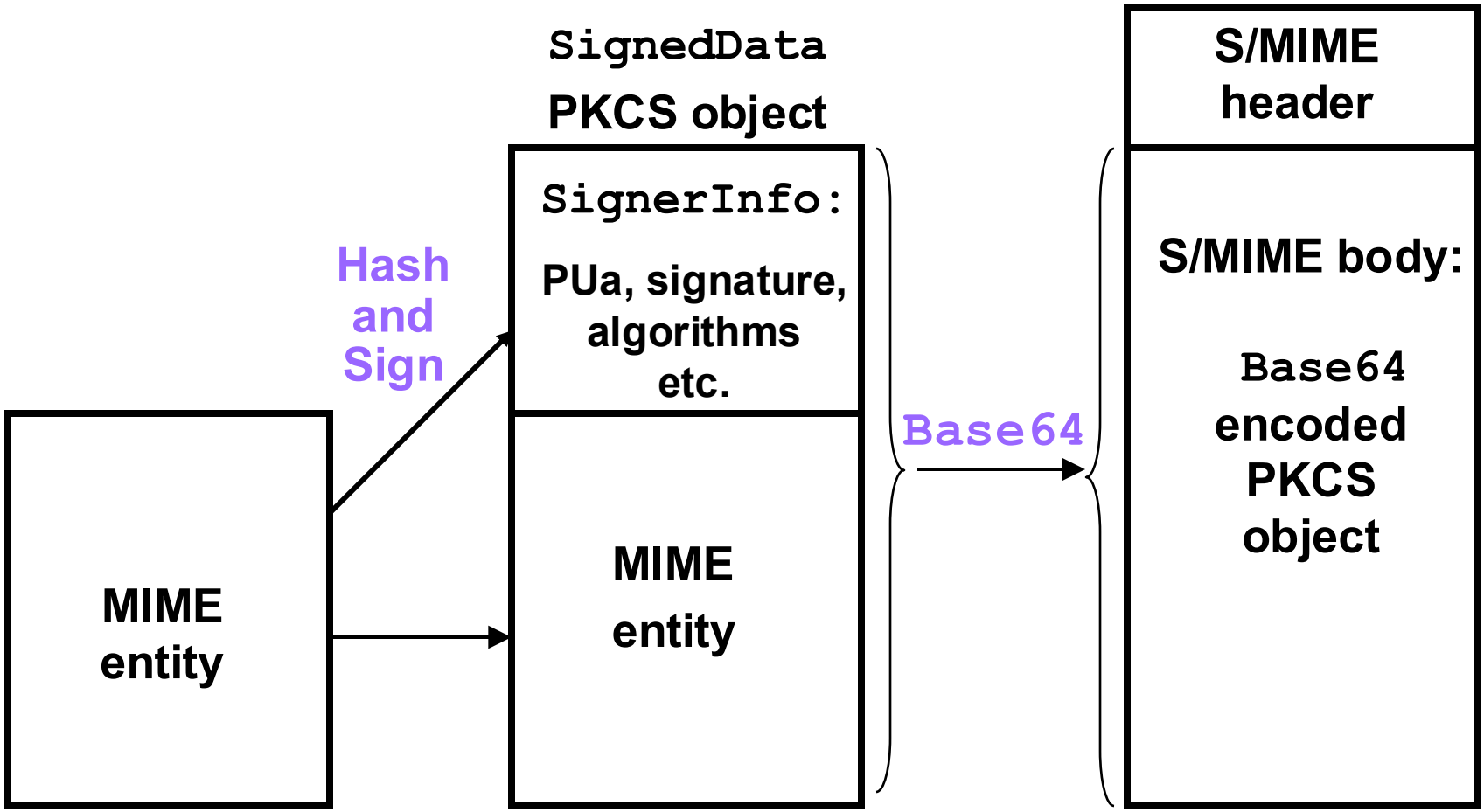


- PKCS: Public Key Cryptography Standard.
- A PKCS object includes the original content plus all information needed for the recipient to perform security processing.

7. S/MIME: EnvelopedData



7. S/MIME: SignedData



7. S/MIME: Certificate Management

- S/MIME uses X.509 v3 certificates
- Increasing levels of checks & hence trust

Class	Identity Checks	Usage
1	name/email check	web browsing/email
2	+ enroll/addr check	email, subs, s/w validate
3	+ ID documents	e-banking/service access

Centralized Authentication

Outline

- Introduction to Centralised authentication
- NTLM
- Kerberos v4
- Kerberos v5

Distributed Client-Server System

- There is a distributed client-server architecture.
  - Users are using machines in an open, distributed environment.
  - They need to be able to access services on servers in different locations.
- Servers should only serve authenticated & authorised users.

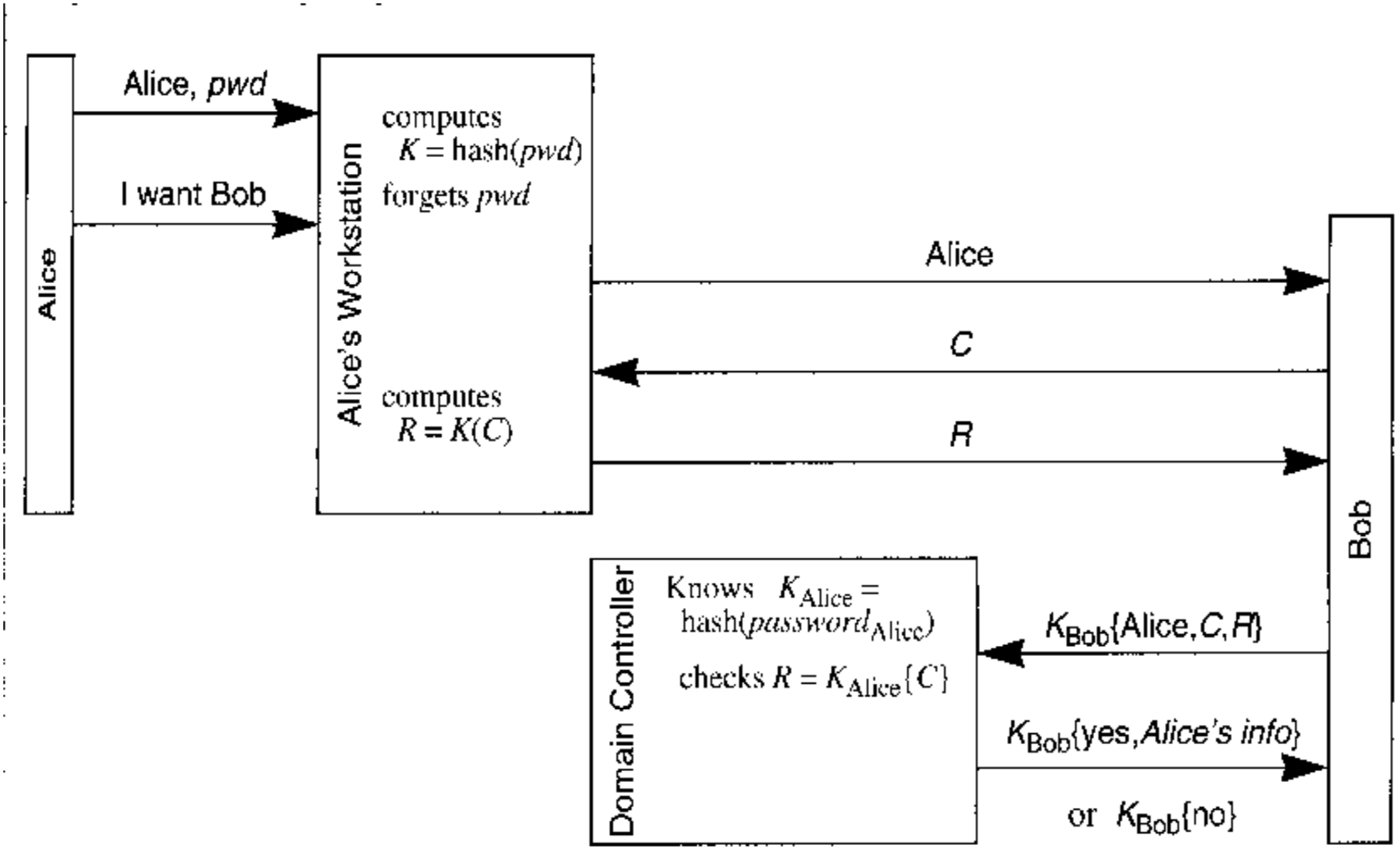
Centralised Approach

- There is a centralised authentication server (AS) who manages all the long-term user credentials
- The centralised AS assists other servers to authenticate the clients and establish session keys

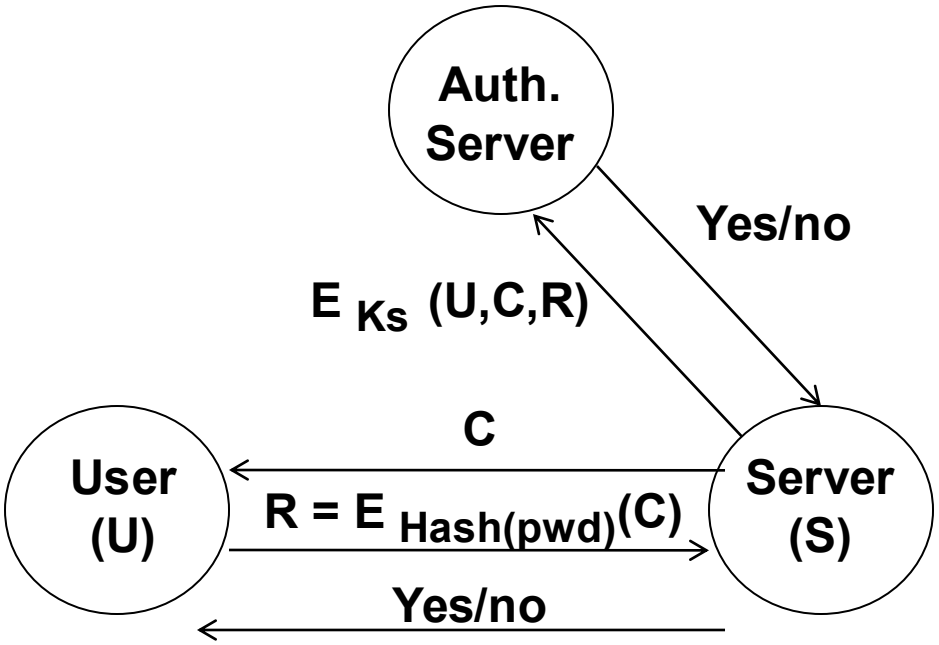
Centralised Approach

- Windows security mechanisms:
  - NTLM: used in Windows NT
  - Kerberos: used since Windows 2000

NTLM

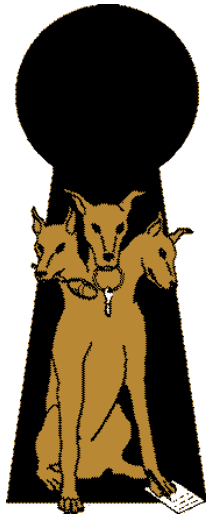

Alice
Alice, pwd
I want Bob
Alice's Workstation
computes
K = hash(pwd)
forgets pwd
computes
R = K(C)
Alice
C
R
Bob
Domain Controller
Knows K_Alice = hash(password_Alice)
checks R = K_Alice{C}
K_Bob{Alice, C, R}
K_Bob{yes, Alice's info}
or K_Bob{no}

NTLM



- Auth. Server has: hashed pwds
- Server sends a challenge – nonce C
- User sends a response R – encrypted C with hashed pwd
- Encrypt R is forwarded to Auth. Server (with S's key shared with AS)

Kerberos



- Named after the three headed watchdog that guarded the gates of Hades in Greek mythology.
- It is an authentication service developed at MIT as part of project Athena.
- Scenario:
  - Users are using workstations in an open, distributed environment. They need to be able to access services on servers in different locations. There is a distributed client/server architecture.
  - Servers should only serve authorised users and should be able to authenticate requests.
- Kerberos is an example of an *Authentication and authorisation infrastructure (AAI)*.

Kerberos

Kerberos has three kinds of servers:

- **Kerberos authentication server (AS):**

  A centralized trusted authentication server that issues long lifetime tickets for the whole system.

- **Ticket-granting servers (TGS) :**

  Issue short lifetime tickets.

- **Service servers (S) :**

  Provide different services.

Kerberos Architecture

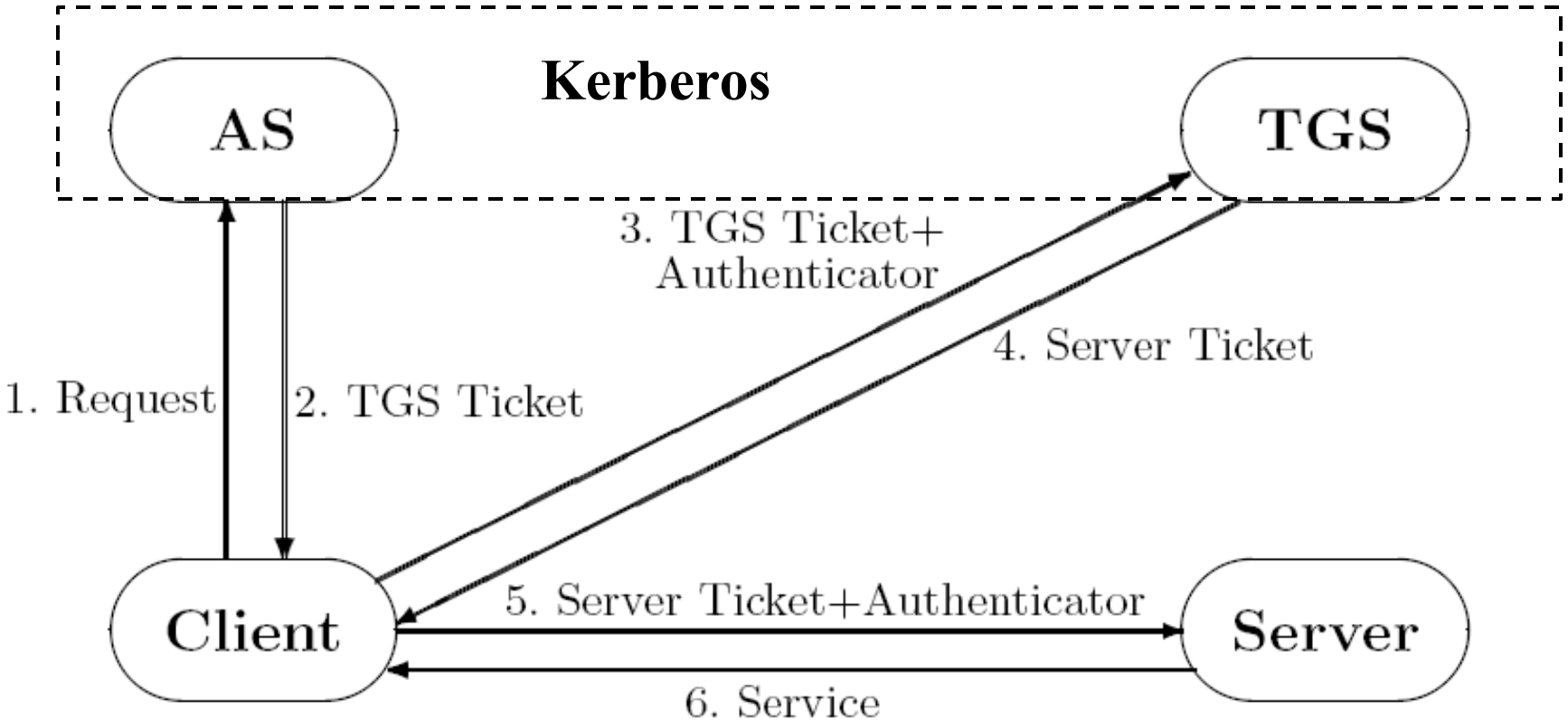


Figure 1. The Framework of Kerberos

Kerberos Operation Overview

- Once per user logon session:
  - (1) C → AS: ID_C, ID_{tgs}
  - (2) AS → C: $E(K_C, K_{c,tgs})$, $Ticket_{tgs}$
- Once per type of service:
  - (3) C →TGS: ID_C, ID_v ,$Ticket_{tgs}$, $Auth_{c,tgs}$
  - (4) TGS → C: $E(K_{c,tgs}, K_{c,v})$, $Ticket_v$
- Once per service session:
  - (5) C → V: ID_C, $Ticket_v$, $Auth_{c,v}$

Kerberos Protocol V4

1: $C \rightarrow AS$: ID_C, ID_{tgs}, TS_1

2: $AS \rightarrow C$: $E_{K_c}[K_{c,tgs}, ID_{tgs}, TS_2, Lifetime_2, Ticket_{tgs}]$

$Ticket_{tgs} = E_{K_{tgs}}[K_{c,tgs}, ID_C, AD_C, ID_{tgs}, TS_2, Lifetime_2]$

3: $C \rightarrow TGS$: $ID_V, Ticket_{tgs}, Authenticator_C$

$Authenticator_C = E_{K_{c,tgs}}[ID_C, AD_C, TS_3]$

4: $TGS \rightarrow C$: $E_{K_{c,tgs}}[K_{C,V}, ID_V, TS_4, Lifetime_4, Ticket_V]$

$Ticket_V = E_{K_v}[K_{c,v}, ID_C, AD_C, ID_V, TS_4, Lifetime_4]$

5: $C \rightarrow V$: $Ticket_V, Authenticator_C$

$Authenticator_C = E_{K_{c,v}}[ID_C, AD_C, TS_5]$

6: $V \rightarrow C$: $E_{K_{c,v}}[TS_5 + 1]$

Step 1: Client requests…

$C \rightarrow AS: \ ID_C, ID_{tgs}, TS_1$

- Once the user is authenticated to the Client (C), the Client sends the authentication server a request on the behalf of the user:
  - This request includes a time-stamp (TS_1) and two identities:
    - ID_C - to inform AS of the user
    - ID_{tgs} - to inform AS of the Ticket Granting Service required.
  - There may be multiple TGS's.

Step 2: AS responds…

$AS \rightarrow C: \quad E_{K_c}[K_{c,tgs}, ID_{tgs}, TS_2, Lifetime_2, Ticket_{tgs}]$

$Ticket_{tgs} = E_{K_{tgs}}[K_{c,tgs}, ID_C, AD_C, ID_{tgs}, TS_2, Lifetime_2]$

- A session key, $K_{c,tgs}$, is generated for secure communication with the ticket granting server indicated by $ID_{tgs.}$

  A time-stamp (TS_2) is specified, as is a lifetime ($Lifetime_2$) for the ticket.

  $Ticket_{tgs}$ – This is for access to TGS: It includes:

  - The same session key, identity, time-stamp and lifetime.
  - ID_C indicating the user.
  - AD_C indicated the address of the client/user.

Step 3: Ticket Granting request

$C \rightarrow TGS: ID_V, Ticket_{tgs}, Authenticator_C$

$Authenticator_C = E_{K_{c,tgs}}[ID_C, AD_C, TS_3]$

- The client now has a ticket to communicate with a ticket granting service, and in this step it communicates with the TGS to request a Server ticket.
  - ID_V indicates the relevant server.
  - $Ticket_{tgs}$ is the client's permission to access the TGS.
  - $Authenticator_C$
    - Only C and TGS can open it.
    - It is used by TGS to authenticate C.
    - Contains ID_C, AD_C, TS_3.

Step 4: Ticket granting response

$$TGS \rightarrow C: \; E_{K_{c,tgs}}[K_{C,V}, ID_V, TS_4, Lifetime_4, Ticket_V]$$

$$Ticket_V = E_{K_v}[K_{c,v}, ID_C, AD_C, ID_V, TS_4, Lifetime_4]$$

The TGS returns a ticket to C, granting access to server/service V.

- The message is encrypted:
  - Provides confidentiality and authentication.
- A key, $K_{C,V}$, for C to talk to V.
- ID_V is the identity of the server
- There is a new time-stamp (TS_4) and a lifetime for the new ticket.

Step 5: Client request (of server)

$C \rightarrow V$: $Ticket_V$, $Authenticator_C$

$Authenticator_C = E_{K_{c,v}}[ID_C, AD_C, TS_5]$

- The client now communicates with V for access.
  - $Ticket_V$
  - $Authenticator_C$ - Only C and V can open it
    - Used by V to authenticate C.
    - Contains ID_C, AD_C, TS_5.

Step 6: Server response (to client)

$$V \rightarrow C: \; E_{K_{c,v}}[TS_5 + 1]$$

- In this step the server acknowledges the message from the client.

Inter-realm

- A **realm** is a Kerberos server, set of clients and a set of application servers, such that:
  - The Kerberos server has the user ID's and hashed passwords of all participating users. All users are registered with the Kerberos server.
  - The Kerberos server shares a secret key with each server, each of which is registered with the Kerberos server.

Inter-realm

- To authenticate across realms we need another property.
  - Each Kerberos server shares a secret key with the Kerberos servers in other realms.
- Some changes are needed in the protocol, and some extra steps are needed too.
  - The ticket requests now reference a service in a remote realm.

Inter-realm

3: $C \rightarrow TGS: ID_{TGS_r}, Ticket_{TGS}, Authenticator_C$

$Authenticator_C = E_{K_{c,tgs}}[ID_C, AD_C, TS_3]$

4: $TGS \rightarrow C: E_{K_{c,tgs}}[K_{C,TGS_r}, ID_{TGSr}, TS_4, Lifetime_4, Ticket_{TGS_r}]$

$Ticket_{TGS_r} = E_{K_{TGS,TGSr}}[K_{c,TGS_r}, ID_C, AD_C, ID_{TGS_r}, TS_4, Lifetime_4]$

5: $C \rightarrow TGS_r: ID_{Vr}, Ticket_{TGSr}, Auth_C$

$Auth_C = E_{K_{c,tgsr}}[ID_C, AD_C, TS_5]$

6: $TGS_r \rightarrow C: E_{K_{c,tgsr}}[K_{C,Vr}, ID_{Vr}, TS_6, Ticket_{Vr}]$

$Ticket_{Vr} = E_{K_{vr}}[K_{c,v}, ID_C, AD_C, ID_{Vr}, TS_6, Lifetime_6]$

7: $C \rightarrow V_r: Ticket_{Vr}, Auth_C$

$Auth_C = E_{K_{c,vr}}[ID_C, AD_C, TS_7]$

Kerberos V4 Limitations

- **Encryption:** V4 uses DES only. V5 allows any encryption method.
- **Restricted ticket lifetime:** V4 uses an 8 bit lifetime, for a maximum of about 21 hours. V5 allows the specification of start and end times.
- **Authentication forwarding:** V4 does not allow credentials issued to one client to be forwarded to another host. Consider the following example of when this might be desirable: A client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials.
- **Double encryption** of the tickets in steps two and four. This is unnecessary and inefficient.
- **Offline dictionary attack:** The message from the authentication server to the client (step 2) can be captured. A **password attack** against it can be launched where success occurs if the decrypted result is of an appropriate form.

Kerberos V5

1: $C \rightarrow AS$: Options, ID_C, R_C, ID_{tgs}, Times, N_1

2: $AS \rightarrow C$: R_C, ID_C, $Ticket_{tgs}$,$E_{Kc}[K_{c,tgs}$,Times, N_1, R_{tgs}, $ID_{tgs}]$
$Ticket_{tgs} = E_{K_{tgs}}[Flags, K_{c,tgs}, R_C, ID_C, AD_C, Times]$

3: $C \rightarrow TGS$: Options, ID_V,Times,N_2,$Ticket_{tgs}$, $Auth_C$
$Auth_C = E_{K_{c,tgs}}[ID_C, R_C, TS_1]$

4: $TGS \rightarrow C$: R_C, ID_C, $Ticket_V$, $E_{K_{c,tgs}}[K_{C,V}$, Times, N_2, R_V, $ID_V]$
$Ticket_V = E_{K_v}[Flags, K_{c,v}, R_C, ID_C, AD_C, Times]$

5: $C \rightarrow V$: Options, $Ticket_V$, $Auth_C$
$Auth_C = E_{K_{c,v}}[ID_C, R_C, TS_2, Subkey, Seq\#]$

6: $V \rightarrow C$: $E_{K_{c,v}}[TS_2$, Subkey, Seq#]

Kerberos V5

- **N** is for Nonce
- **R** is for Realm
- **Options** provides a request for certain **flags** (indicating properties) to be set in the returned ticket, e.g.,
  - PRE-AUTHENT: AS authenticates the client before issuing a ticket
  - HW-AUTHENT: Hardware based initial authentication is employed
  - RENEWABLE: A ticket with this flag set includes two expiration times
    - One for this specific ticket
    - One for the latest permissible expiration time

    A client can have a ticket renewed, if the ticket is not reported stolen
  - FORWARDABLE: A new ticket-granting ticket with a different network address may be issued based on this ticket.
  - ……

Kerberos V5

- **Times** are requested by the client for ticket configuration.
  - *from*: a start-time.
  - *till*: the requested expiration time.
  - *rtime*: requested renew-till time, i.e. allow continued use until.
- **Subkey** is an optional sub-encryption key used to protect a specific session of an application. The default is the session key.
- The **Sequence number (Seq#)** is an optional sequence start number to be used by the server. It is used to protect the system from replay attacks.

Secure SHell (SSH)

I wrote the initial version of SSH (Secure Shell) in Spring 1995. It was a time when telnet and FTP were widely used.

Anyway, I designed SSH to replace both `telnet` (port 23) and `ftp` (port 21). Port 22 was free. It was conveniently between the ports for `telnet` and `ftp`. I figured having that port number might be one of those small things that would give some aura of credibility. But how could I get that port number? I had never allocated one, but I knew somebody who had allocated a port.

The basic process for port allocation was fairly simple at that time. Internet was smaller and we were in the very early stages of the Internet boom. Port numbers were allocated by IANA (Internet Assigned Numbers Authority). At the time, that meant an esteemed Internet pioneer called Jon Postel and Joyce K. Reynolds. Among other things, Jon had been the editor of such minor protocol standards as IP (RFC 791), ICMP (RFC 792), and TCP (RFC 793). Some of you may have heard of them.

To me Jon felt outright scary, having authored all the main Internet RFCs!

Anyway, just before announcing `ssh-1.0` in July 1995, I sent this e-mail to IANA:

```
From ylo Mon Jul 10 11:45:48 +0300 1995 From: Tatu Ylonen <ylo@cs.hut.fi>
To: Internet Assigned Numbers Authority <iana@isi.edu>
Subject: request for port number
Organization: Helsinki University of Technology, Finland
Dear Sir, I have written a program to securely log from one machine into another over an insecure network.
It provides major improvements in security and functionality over existing telnet and rlogin protocols
and implementations. In particular, it prevents IP, DNS and routing spoofing.  My plan is to distribute
the software freely on the Internet and to get it into as wide use as possible. I would like to get a registered
privileged port number for the software.

The number should preferably be in the range 1-255 so that it can be used in the WKS field in name servers.
I'll enclose the draft RFC for the protocol below. The software has been in local use for several months,
and is ready for publication except for the port number. If the port number assignment can be arranged in time,
I'd like to publish the software already this week. I am currently using port number 22 in the beta test.
It would be great if this number could be used (it is currently shown as Unassigned in the lists).
The service name for the software is "ssh" (for Secure Shell).
Yours sincerely,  Tatu Ylonen <ylo@cs.hut.fi>  ... followed by protocol specification for ssh-1.0
```

The next day, I had an e-mail from Joyce waiting in my mailbox:

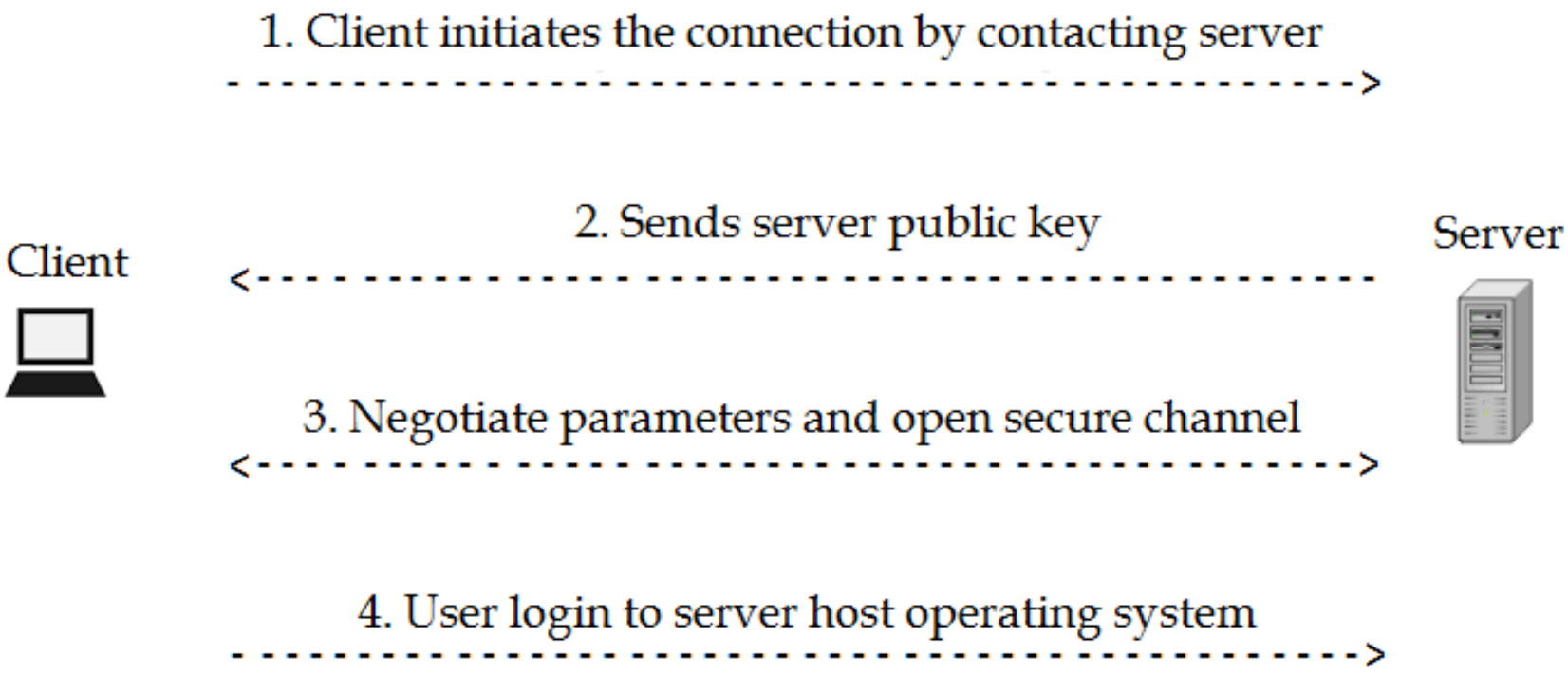
```
Date: Mon, 10 Jul 1995 15:35:33 -0700 From: jkrey@ISI.EDU To: ylo@cs.hut.fi Subject: Re: request for port number
Cc: iana@ISI.EDU  Tatu,  We have assigned port number 22 to ssh, with you as the point of contact.  Joyce
```

There we were! SSH port was 22!!!

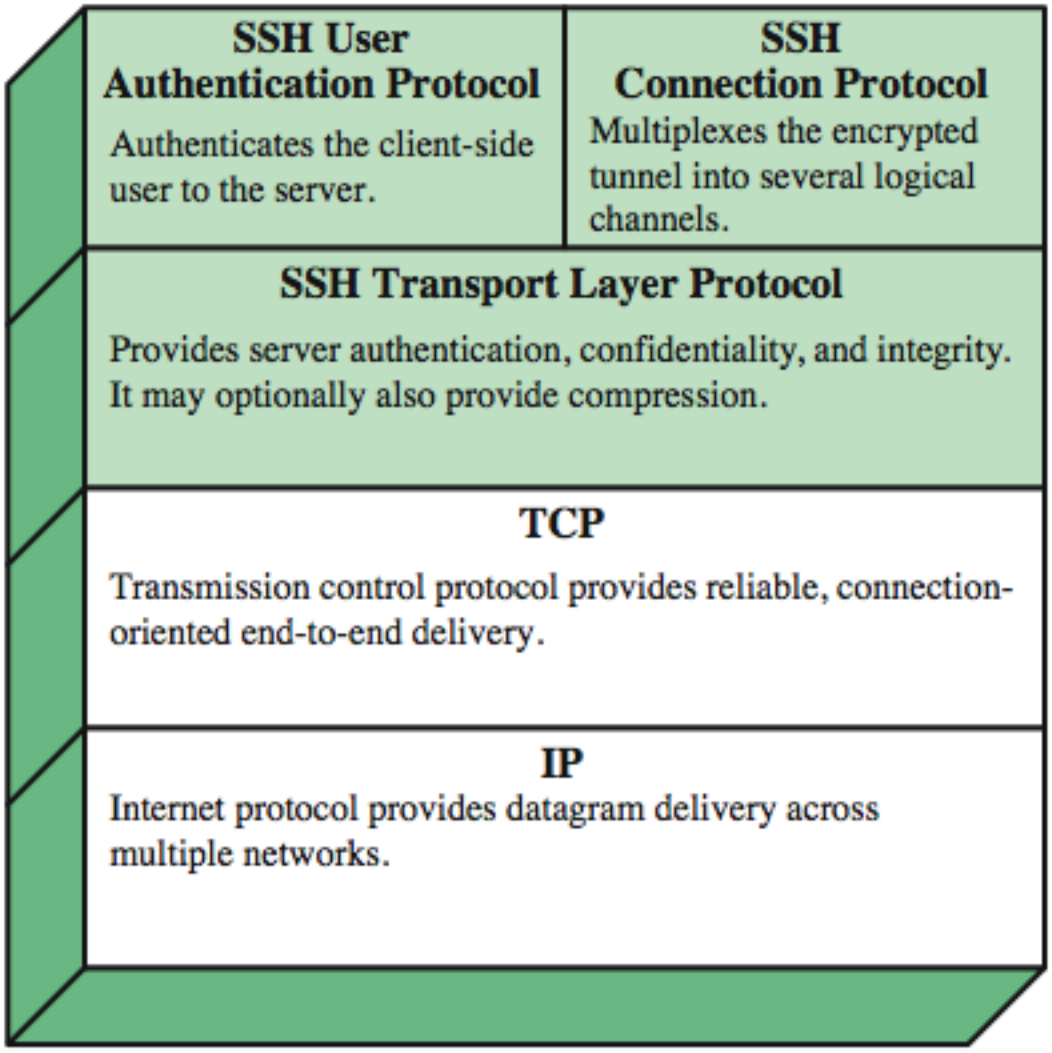
SSH Overview

- SSH = Secure Shell
  - Initially designed to replace insecure rsh, telnet utilities.
  - Secure remote administration (mostly of Unix systems).
  - Latter, provide a general secure channel for network applications.
  - Only covers traffic explicitly protected.
  - Applications need modification, but port-forwarding eases some of this

SSH Overview



SSH Protocol Stack



SSH-2 Architecture

SSH-2 adopts a three layer architecture:

- SSH Transport Layer Protocol.
  - Initial connection.
  - Server authentication
  - Sets up secure channel between client and server via key exchange etc.
- SSH Authentication Protocol
  - Client authentication over secure transport layer channel.
- SSH Connection Protocol
  - Supports multiple connections over a single transport layer protocol secure channel.
  - Efficiency (session re-use).

SSH-2 Security Goals

- Server authenticated in transport layer protocol.
- Client authenticated in authentication protocol.
  - By public key (DSS, RSA).
  - Or simple password
- Establishment of a fresh, shared secret.
  - Shared secret used to derive further keys (Enc Keys, MAC keys, IVs), similar to SSL/TLS.
  - For confidentiality and authenticity in SSH transport layer protocol.
- Secure ciphersuite negotiation.
  - Encryption, MAC, and compression algorithms.

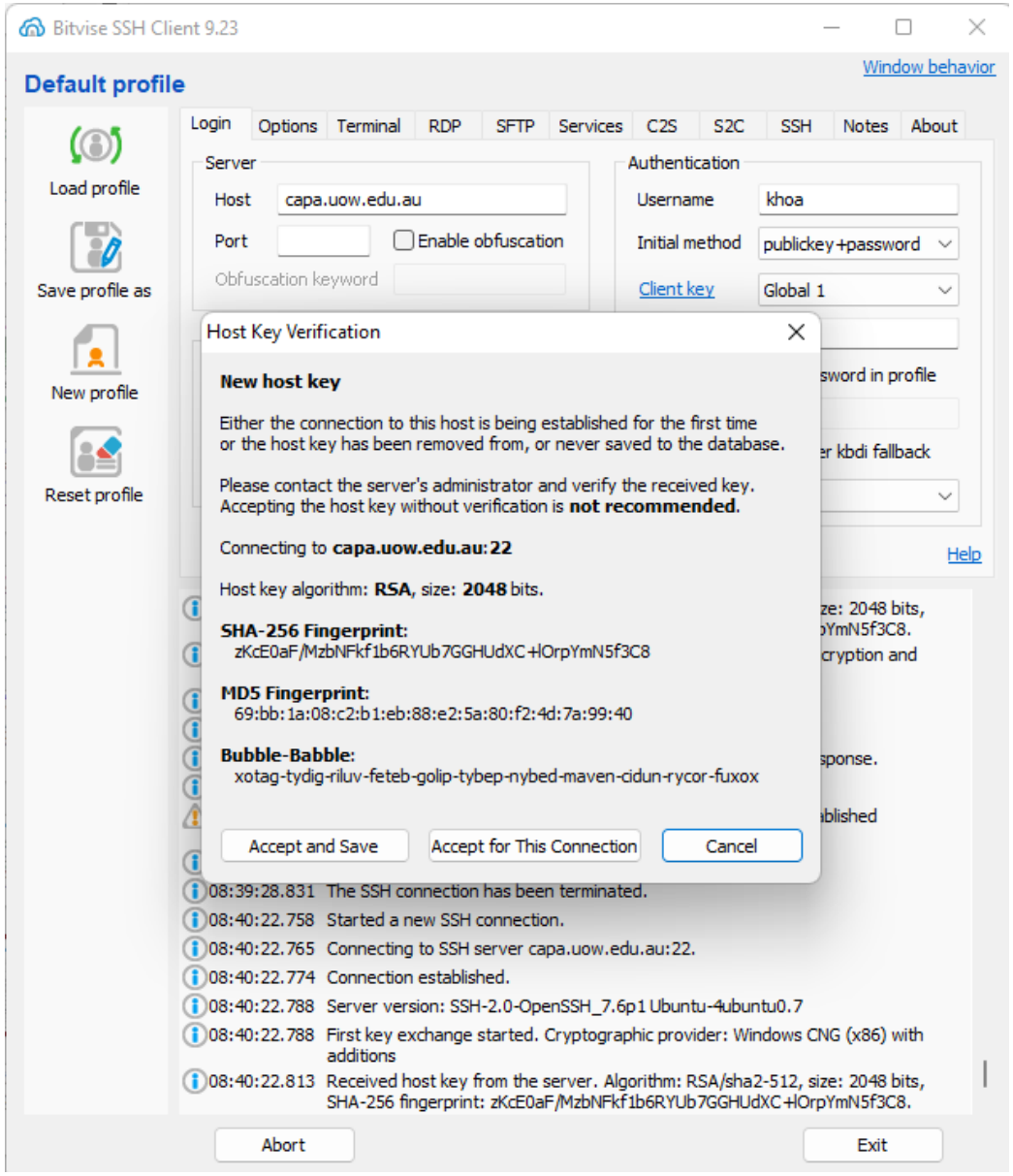
SSH Transport Layer Protocol

- Server authentication, based on server's host key pair(s)
- Packet exchange
  - establish TCP connection
  - can then exchange data (packet exchange)
    - identification string exchange, algorithm negotiation, key exchange, end of key exchange, service request
  - service request: either the User Authentication or the Connection Protocol

SSH Key Fingerprints

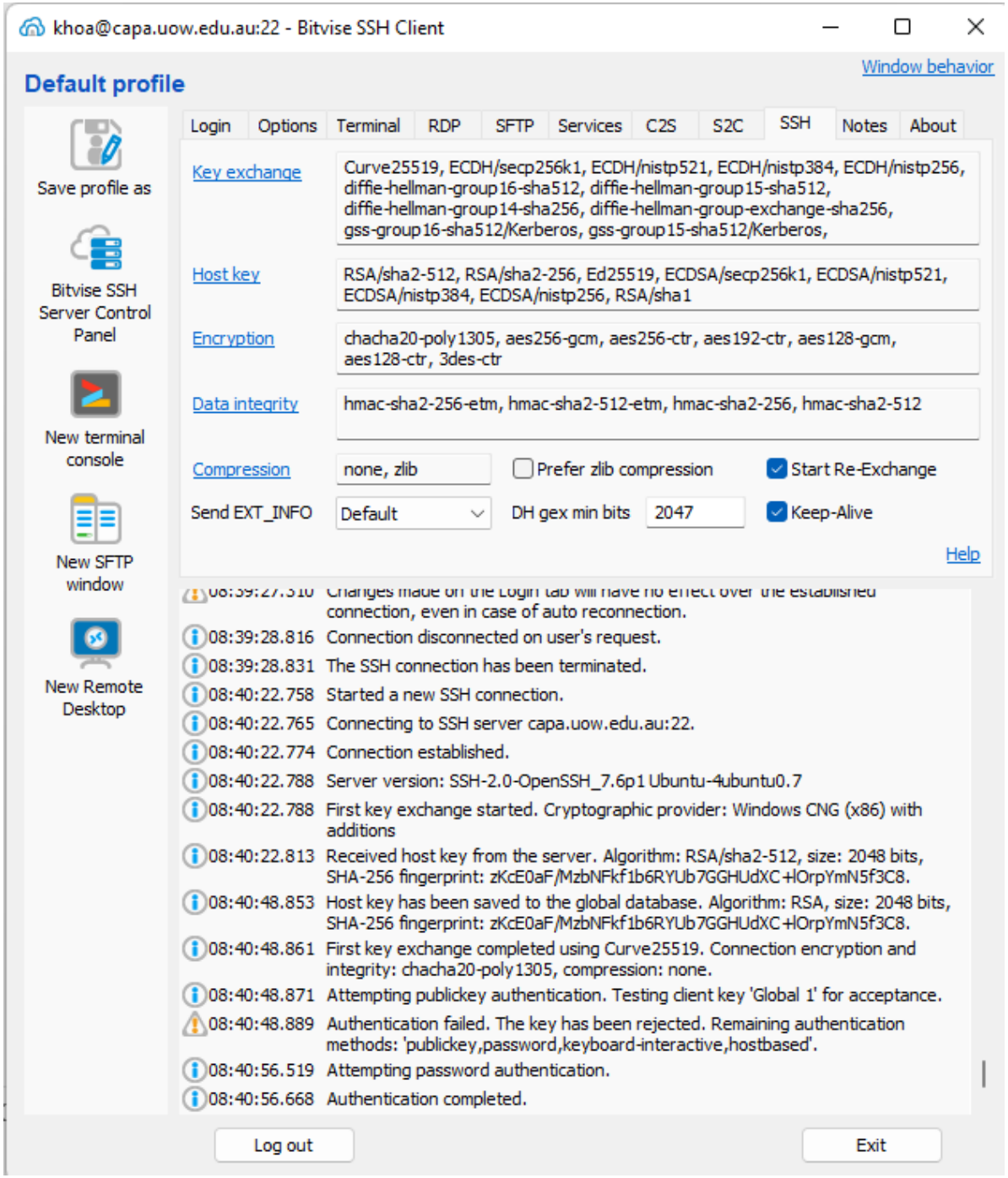
- **The security of the connection relies on *the server authenticating itself to the client*.**
- **When you connect to a remote host computer for the first time, the host sends your local computer its public key in order to identify itself. To help you to verify the host's identity, a fingerprint of the host's public key is presented to you for verification.**
- **Many users just blindly accept the presented key.**

SSH Key Fingerprints



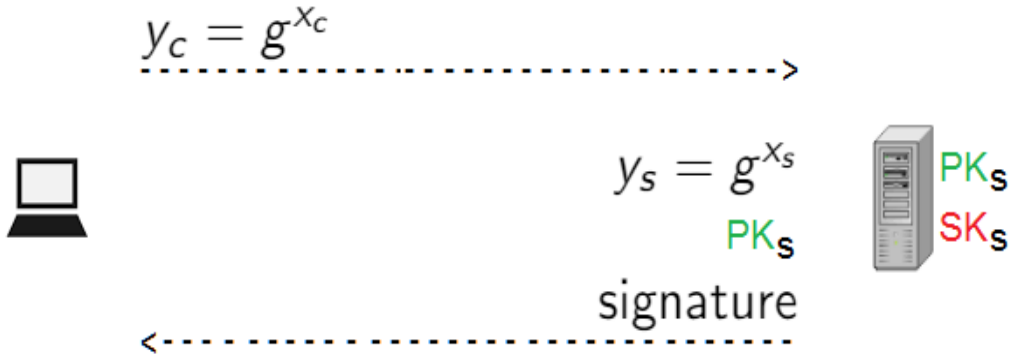
SSH-2 Algorithms

- Key establishment through Diffie-Hellman key exchange.
  - Ephemeral Diffie-Hellman
- Server authentication via RSA or DSS signatures
- HMAC-SHA1 or HMAC-SHA256 for MAC algorithm.
- 3DES, AES, RC4, etc. for Encryption algorithm


khoa@capa.uow.edu.au:22 - Bitvise SSH Client
Default profile
Window behavior
Save profile as
Bitvise SSH Server Control Panel
New terminal console
New SFTP window
New Remote Desktop
Login
Options
Terminal
RDP
SFTP
Services
C2S
S2C
SSH
Notes
About
Key exchange
Curve25519, ECDH/secp256k1, ECDH/nistp521, ECDH/nistp384, ECDH/nistp256, diffie-hellman-group16-sha512, diffie-hellman-group15-sha512, diffie-hellman-group14-sha256, diffie-hellman-group-exchange-sha256, gss-group16-sha512/Kerberos, gss-group15-sha512/Kerberos,
Host key
RSA/sha2-512, RSA/sha2-256, Ed25519, ECDSA/secp256k1, ECDSA/nistp521, ECDSA/nistp384, ECDSA/nistp256, RSA/sha1
Encryption
chacha20-poly1305, aes256-gcm, aes256-ctr, aes192-ctr, aes128-gcm, aes128-ctr, 3des-ctr
Data integrity
hmac-sha2-256-etm, hmac-sha2-512-etm, hmac-sha2-256, hmac-sha2-512
Compression
none, zlib
Prefer zlib compression
Start Re-Exchange
Send EXT_INFO
Default
DH gex min bits
2047
Keep-Alive
Help
connection, even in case of auto reconnection.
08:39:28.816 Connection disconnected on user's request.
08:39:28.831 The SSH connection has been terminated.
08:40:22.758 Started a new SSH connection.
08:40:22.765 Connecting to SSH server capa.uow.edu.au:22.
08:40:22.774 Connection established.
08:40:22.788 Server version: SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.7
08:40:22.788 First key exchange started. Cryptographic provider: Windows CNG (x86) with additions
08:40:22.813 Received host key from the server. Algorithm: RSA/sha2-512, size: 2048 bits, SHA-256 fingerprint: zKcE0aF/MzbNFkf1b6RYUb7GGHUdXC+lOrpYmN5f3C8.
08:40:48.853 Host key has been saved to the global database. Algorithm: RSA, size: 2048 bits, SHA-256 fingerprint: zKcE0aF/MzbNFkf1b6RYUb7GGHUdXC+lOrpYmN5f3C8.
08:40:48.861 First key exchange completed using Curve25519. Connection encryption and integrity: chacha20-poly1305, compression: none.
08:40:48.871 Attempting publickey authentication. Testing client key 'Global 1' for acceptance.
08:40:48.889 Authentication failed. The key has been rejected. Remaining authentication methods: 'publickey,password,keyboard-interactive,hostbased'.
08:40:56.519 Attempting password authentication.
08:40:56.668 Authentication completed.
Log out
Exit

SSH Transport Layer Protocol

Diffie-Hellman Key Exchange



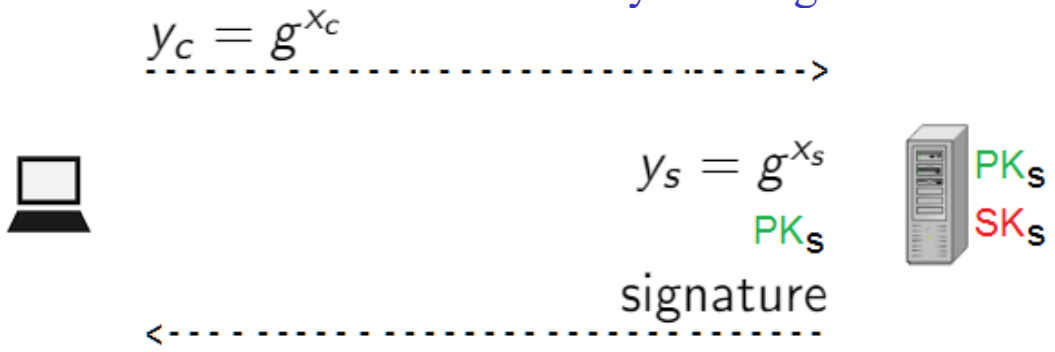
Client generates a random number x_c and computes

$$y_c = g^{x_c} \pmod{p}.$$

Client sends y_c to Server.

SSH Transport Layer Protocol

Diffie-Hellman Key Exchange



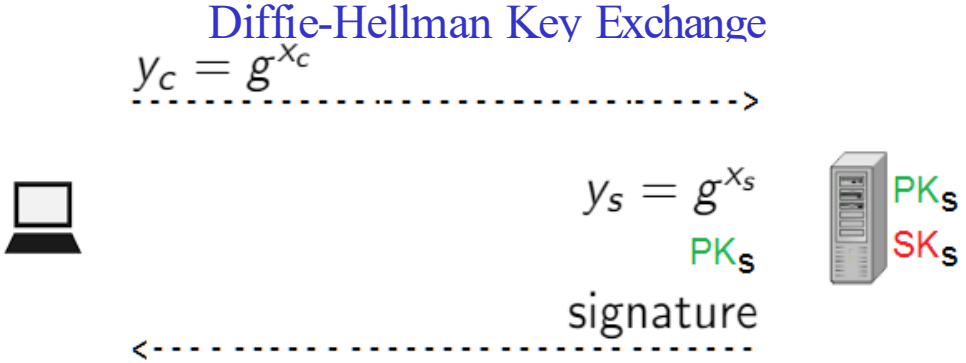
Server generates a random number x_s and computes

$$y_s = g^{x_s} \pmod{p}.$$

Server computes the shared secret

$$K = y_c^{x_s} = g^{x_c x_s} \pmod{p}.$$

SSH Transport Layer Protocol



Server computes the exchange hash value

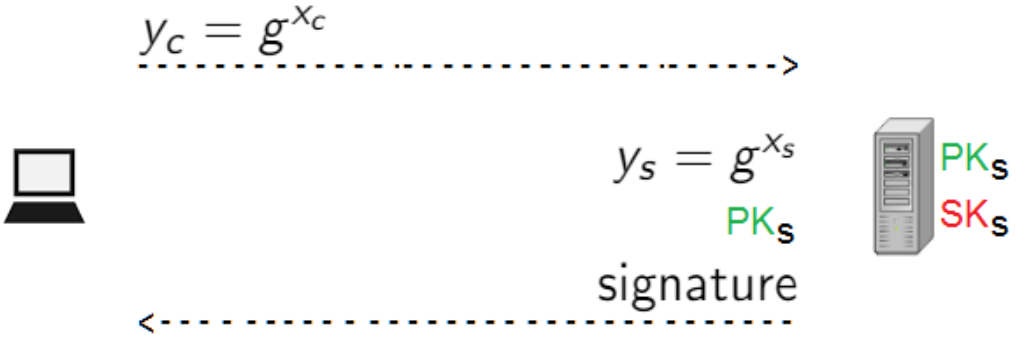
$$H = hash(id_C \,||id_S \,||init_C \,||init_S \,||PK_S \,||y_c \,||y_s \,||K\,)$$

id_S , id_C : Server's and Client's identification strings

$init_S$, $init_C$: Server's and Client's Initial Messages

SSH Transport Layer Protocol

Diffie-Hellman Key Exchange



Server generates the signature on the exchange hash value *signature* = $Sign_{SK_S}(H)$ and sends

(y_s, PK_S, *signature*)

to Client.

SSH Transport Layer Protocol

Key Derivation

- **After the key exchange, both Server and Client obtain two common values:**
  - **a shared secret value *K* , and**
  - **an exchange hash value *H*.**
- **Encryption keys and MAC keys are derived from *K* and *H*.**
- **The exchange hash value *H* from the first key exchange is additionally used as the session identifier.**

SSH Transport Layer Protocol

Key Derivation

Encryption keys must be computed as hash of the shared secret *K*
as follows:

-Initial IV client to server: *hash*(*K*||*H*||"A"||*session id*)

-Initial IV server to client: *hash*(*K*||*H*||"B"||*session id*)

-Encryption key client to server: *hash*(*K*||*H*||"C"||*session id*)

-Encryption key server to client: *hash*(*K*||*H*||"D"||*session id*)

-MAC key client to server: *hash*(*K*||*H*||"E"||*session id*)

-MAC key server to client: *hash*(*K*||*H*||"F"||*session id*)

SSH Transport Layer Protocol

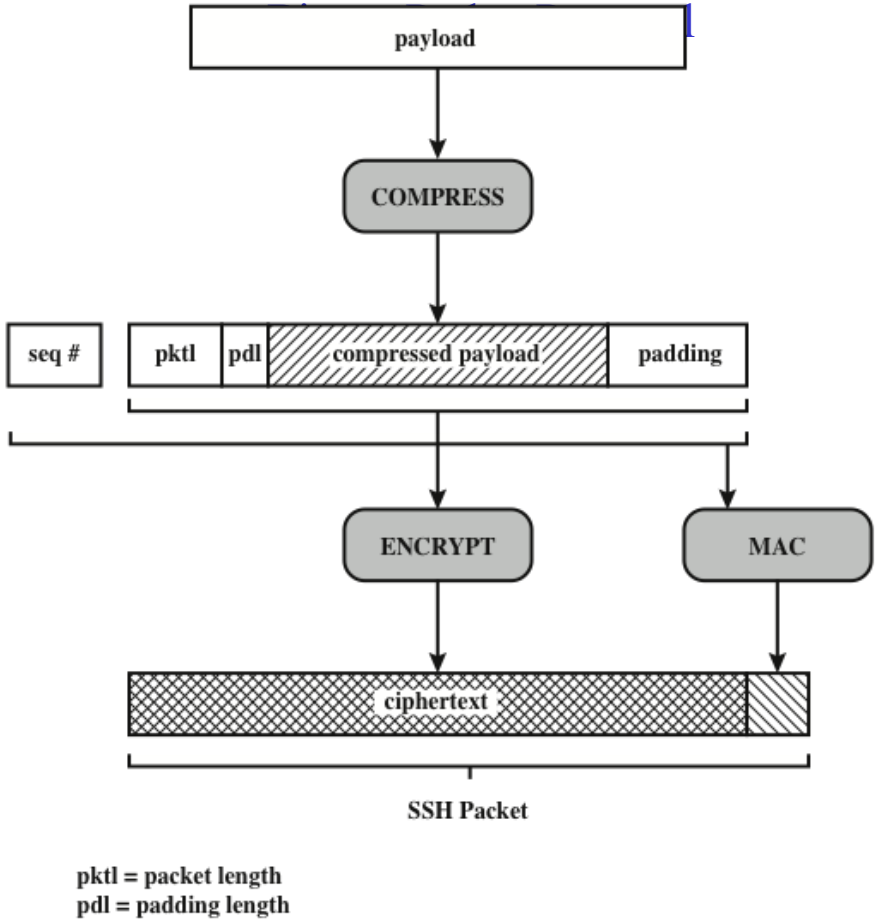


Figure 17.10 SSH Transport Layer Protocol Packet Formation

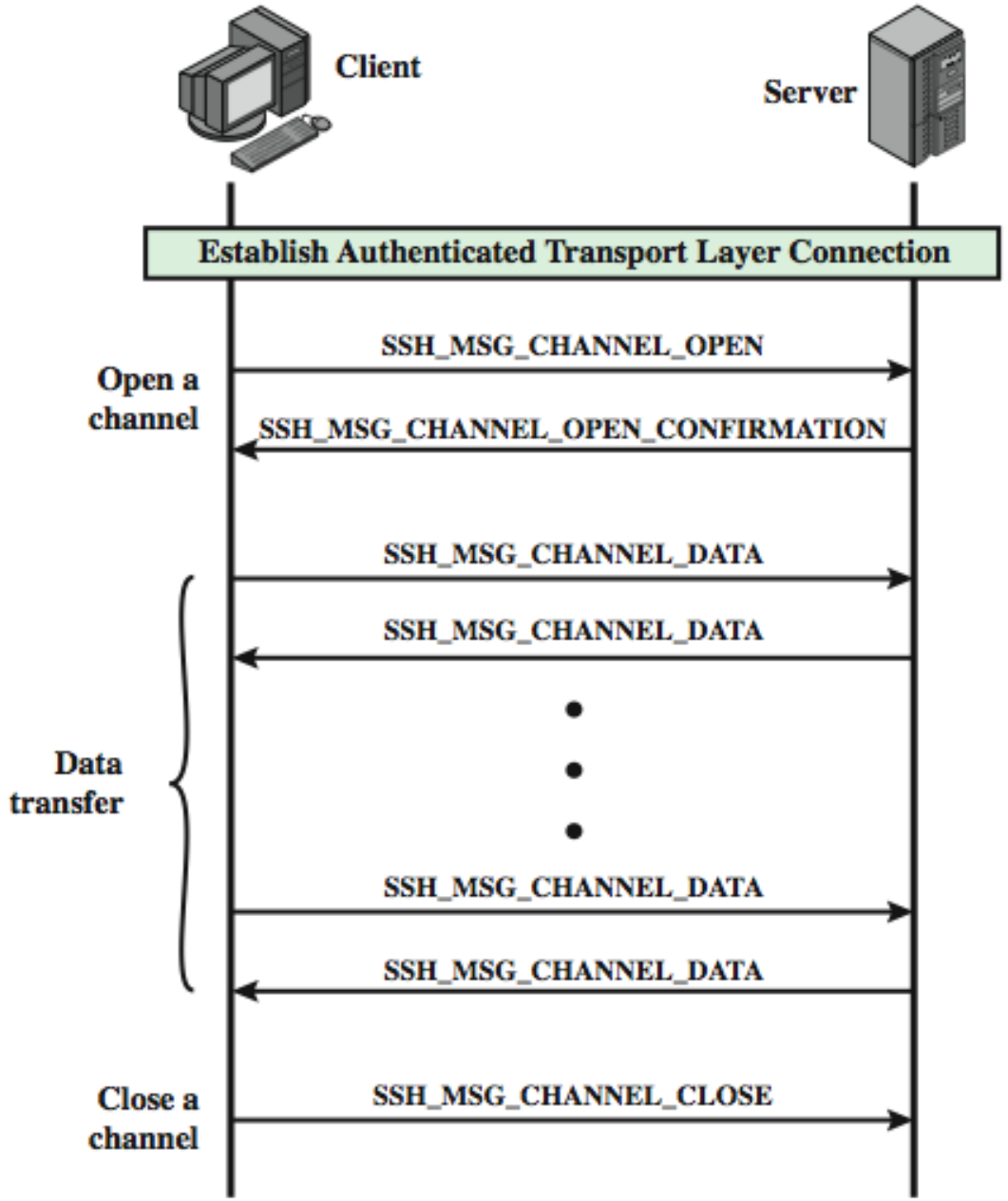
SSH User Authentication Protocol

- Authenticates client to server
- Authentication methods used
  - public key (digital signature)
  - password
  - host-based

SSH Connection Protocol

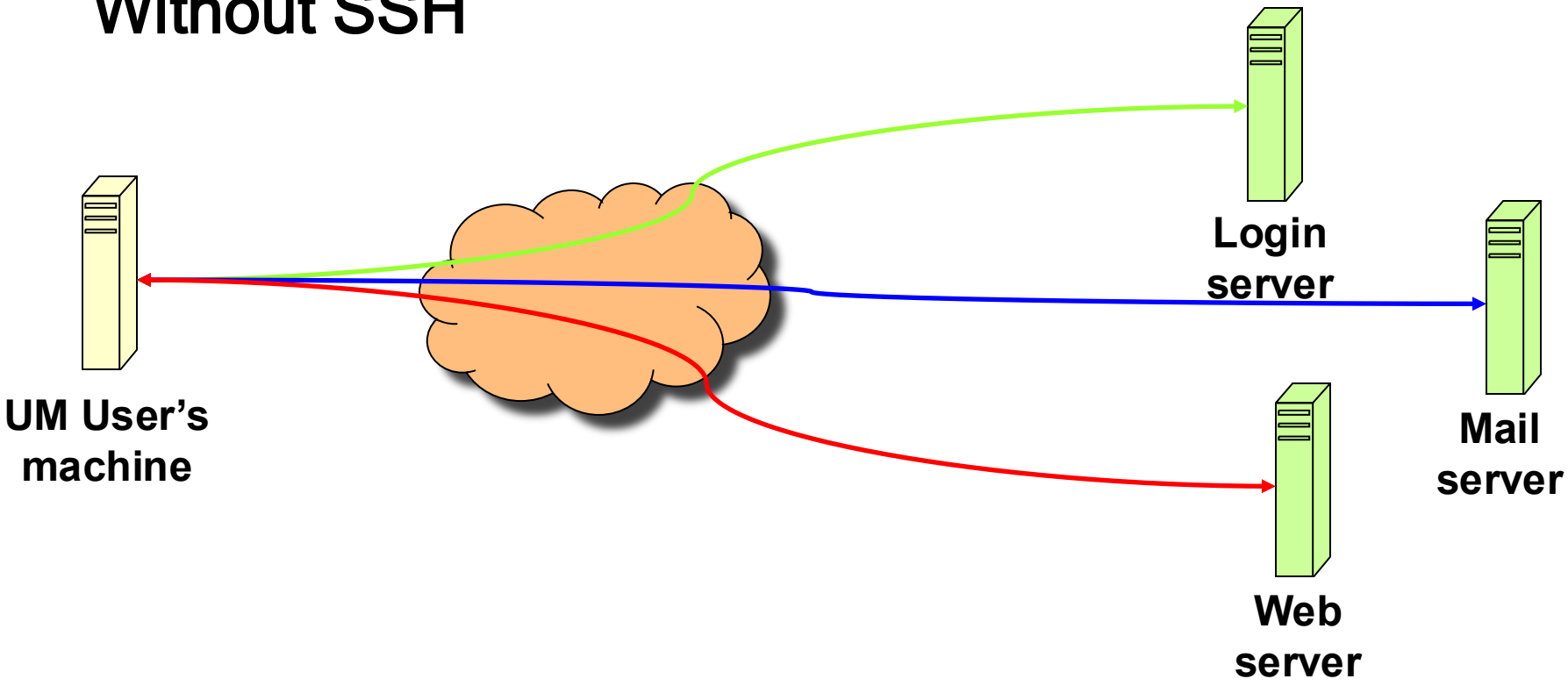
- Run on top of the SSH Transport Layer Protocol
- Assume secure authentication connection
- Used for multiple **logical** channels
  - Different SSH communications use separate channels
  - either side can open a channel with unique id number
  - have three stages:
    - opening a channel, data transfer, closing a channel
  - four types:
    - Session: The remote execution of a program.
    - X11: This refers to the X Window System
    - forwarded-tcpip: This is remote port forwarding
    - direct-tcpip: This is local port forwarding

SSH Connection Protocol Exchange



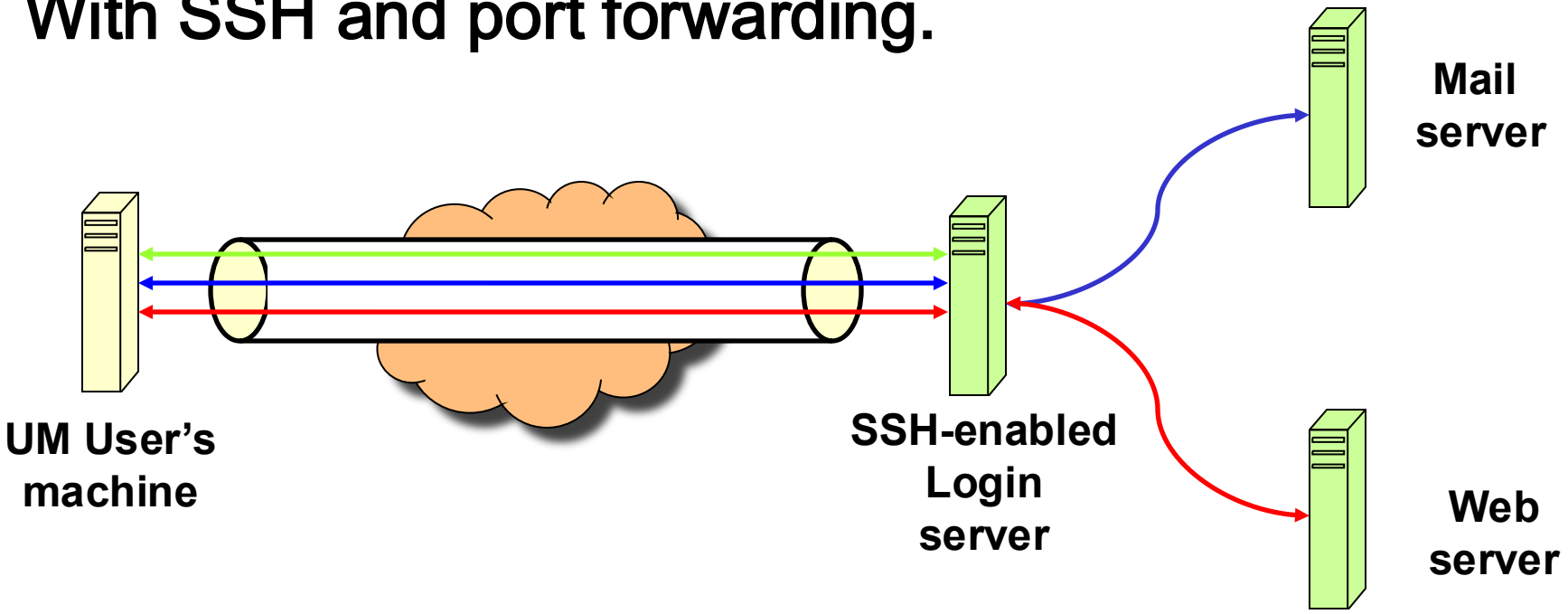
SSH Port Forwarding

Without SSH

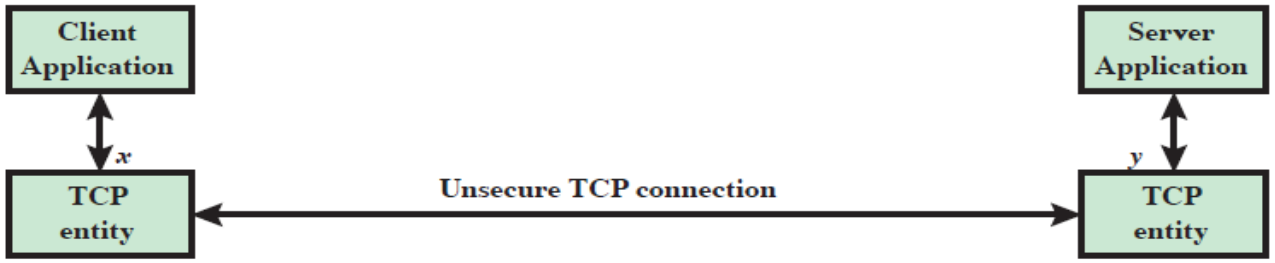


SSH Port Forwarding

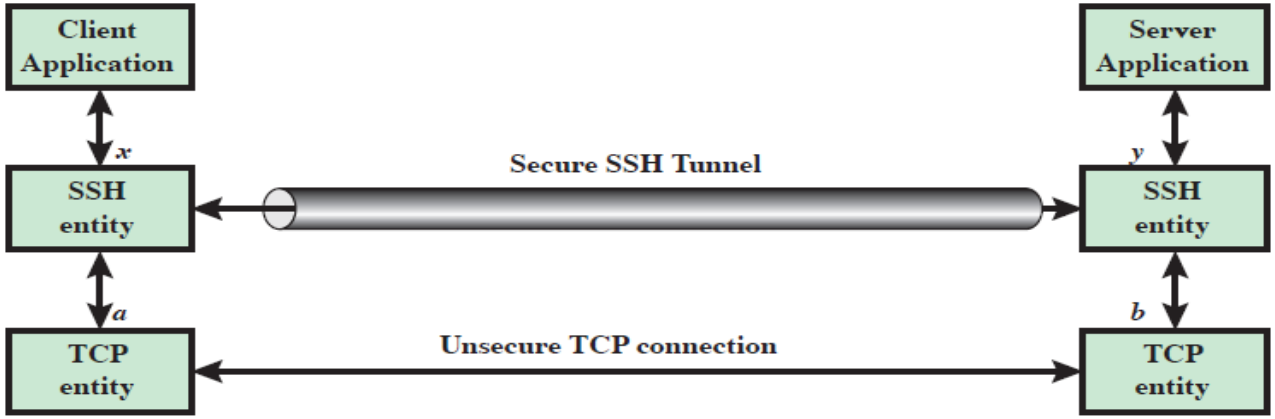
With SSH and port forwarding.



SSH Port Forwarding



(a) Connection via TCP



(b) Connection via SSH Tunnel

Port Forwarding

- **Client sets up an SSH connection to the remote SSH server**
- **Select a local port x and configure SSH to accept traffic from this port destined for port y on a remote application server**
- **Client informs SSH Server to create a connection to the destination (application server port y)**
- **Client takes any bits sent to local port x and sends them to the server via an SSH session. The SSH server decrypts the bits and sends the plaintext to port y of the application server**

SSH Applications

- Anonymous ftp for software updates, patches...
  - No client authentication needed, but clients want to be sure of origin and integrity of software.
- Secure ftp.
  - E.g.upload of webpages to webserver using sftp.
  - Server now needs to authenticate clients.
  - Username and password sufficient, transmitted over secure SSH transport layer protocol.
- Secure remote administration.
  - SysAdmin (client) sets up terminal on remote machine.
  - SysAdmin password protected by SSH transport layer protocol.
- Virtual Private Network.
  - E.g. use SSH + port forwarding to secure the communications of other applications.