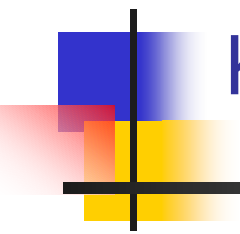


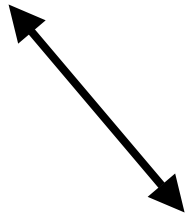
Authentication in real world: Kerberos, SSH and SSL

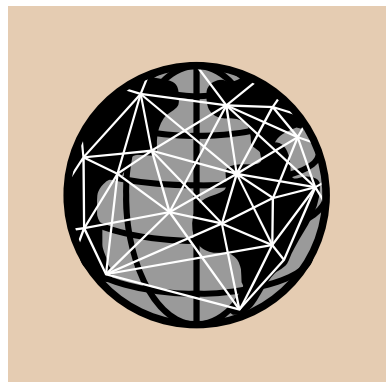
Zheng Ma

Apr 19, 2005

Where are we?

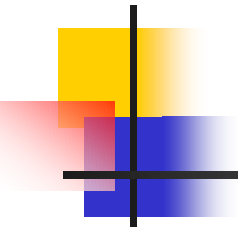
- After learning all the foundation of modern cryptography, we are ready to see some real world applications based on them.





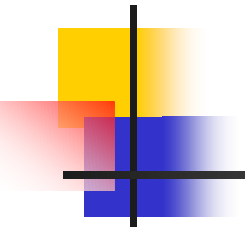
- What happened when you use your Yale netid and password? How does our system authenticate yourself

- Internet is a tough environment, security protocols need to deal with many different scenarios of attacks.



Think about Authentication

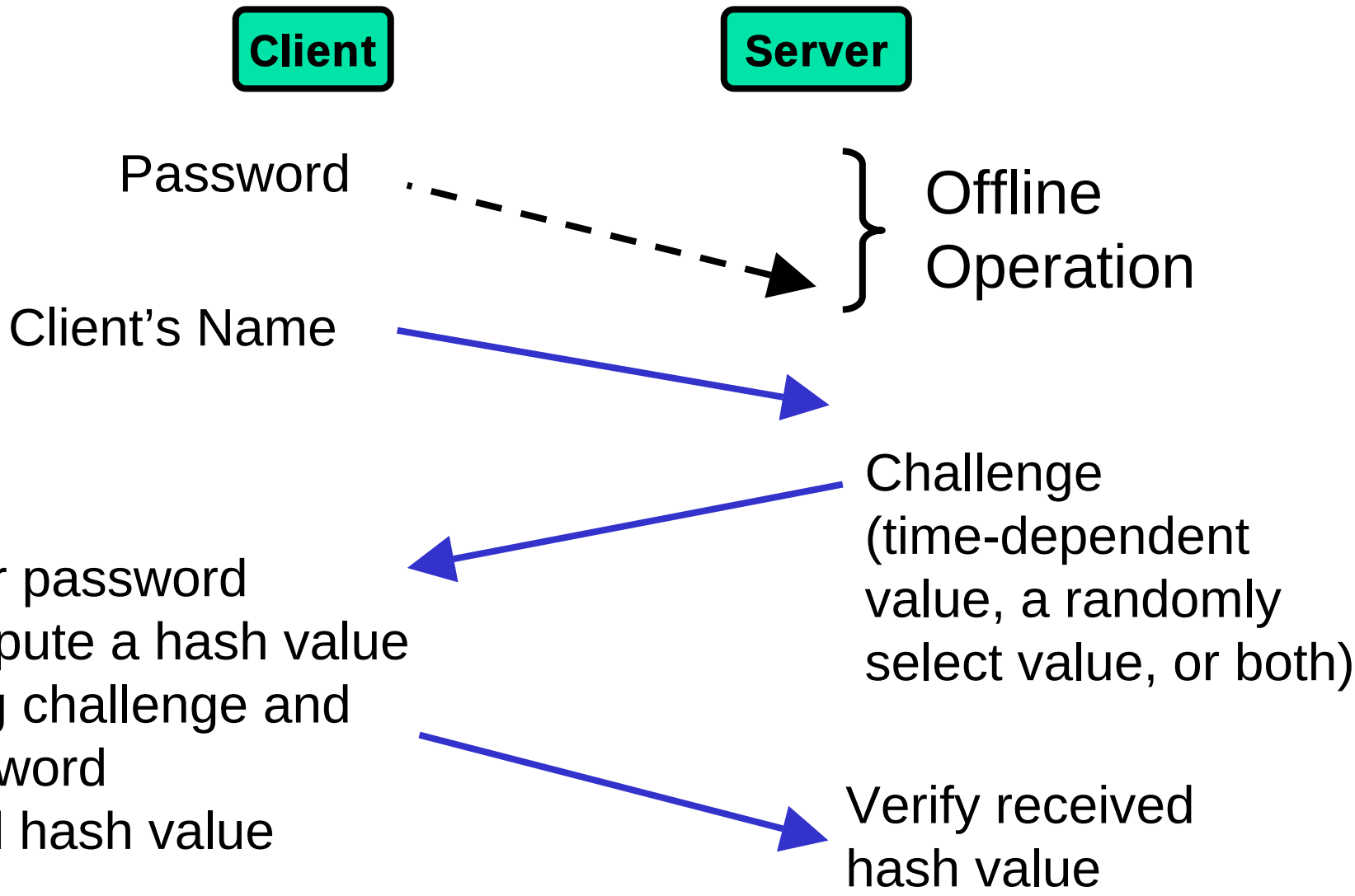
- Authentication provides a means to identify a client that requires access to some system.
  - Network services, such as telnet, pop3, and nfs, need to authenticate individual users, by using their passwords, for example. We use our netid/password to access sis, email, pantheon and etc everyday.
- Note that firewalls can not replace authentication
- In general, good users may be on bad hosts, and bad users may be on good hosts.
  - Thus, blocking traffic based on IP addresses and port numbers is not sufficient
- The mechanism for authentication is typically undertaken through the exchange of keys or certificates between the client and the server. What should we do?

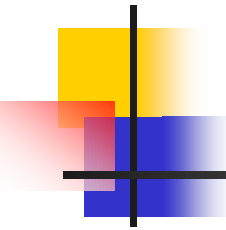


Use of Password over a Network

- Of course, passwords should not be sent in clear text
- What about sending encrypted passwords? No, they should not be sent over the network either. This is to avoid replay attacks
- Next slide shows a typical method of defending against password replay attacks. The method uses no encrypted password

Use of Challenges to Defend Against Password Replay



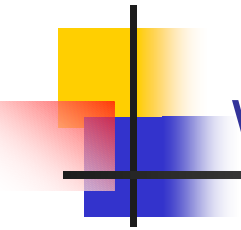


The "O(N^2) Password Management Problem"

- Each of the N servers authenticates each of the N users
- Every server keeps track of the password of every user
- Thus a total of O(N^2) pieces of information items to manage

Kerberos' Objective: Provide an O(N) Solution

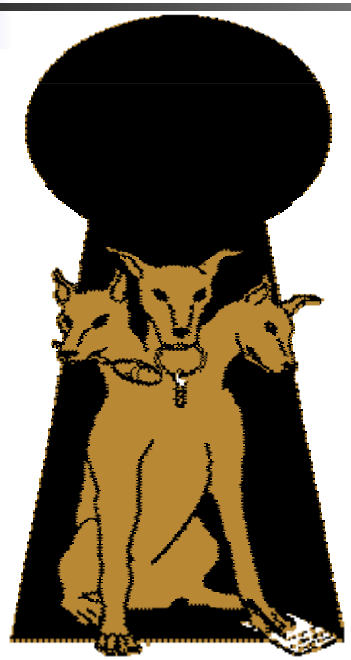
- Use a single authentication server that has trusted relationship with N clients and N servers. Thus, only O(N) keys to worry about
- The authentication server will generate session keys (aka "tickets") for each client-server session



What is Kerberos?

- Part of project Athena (MIT).
- Trusted 3rd party authentication scheme. Key Distribution Center (KDC)
- Assumes that hosts are not trustworthy.
- Requires that each client (each request for service) prove it's identity.
- Does not require user to enter password every time a service is requested!

Kerberos: etymology



The 3-headed dog that guards the entrance to Hades

Fluffy, the 3 headed dog, from "Harry Potter and the Sorcerers Stone"

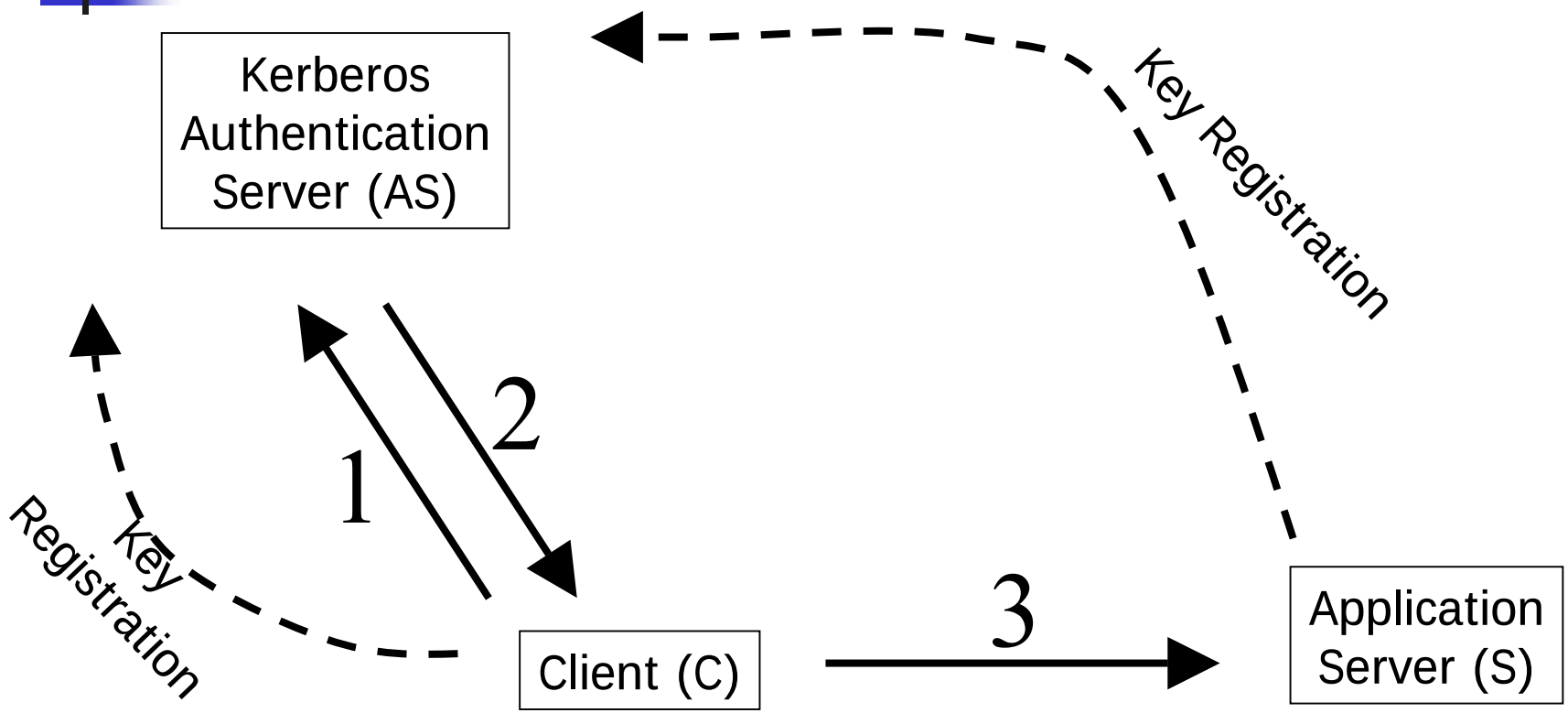
Originally, the 3 heads represented the 3 A's (Authentication, Authorization, and Accounting)

We'll focus on authentication

How Kerberos Tickets Work (Daily Experience)

- A user first gets a ticket from the Kerberos authentication server. A ticket is like a driver's license issued by the DMV
- When attempting to make use of a network service, the user presents the ticket to the service, along with the user's "authenticator". The service then examines the ticket and the authenticator to verify the identity of the user. If all checks out, then the user is accepted
- This is like a customer presenting his driver's license to a supermarket manager when trying to cash a personal check. In this case, the customer's "authenticator" is the customer's face with which the supermarket manager can match the photo on the driver's license
- Note that a ticket can be used many times until it expires

Kerberos Authentication

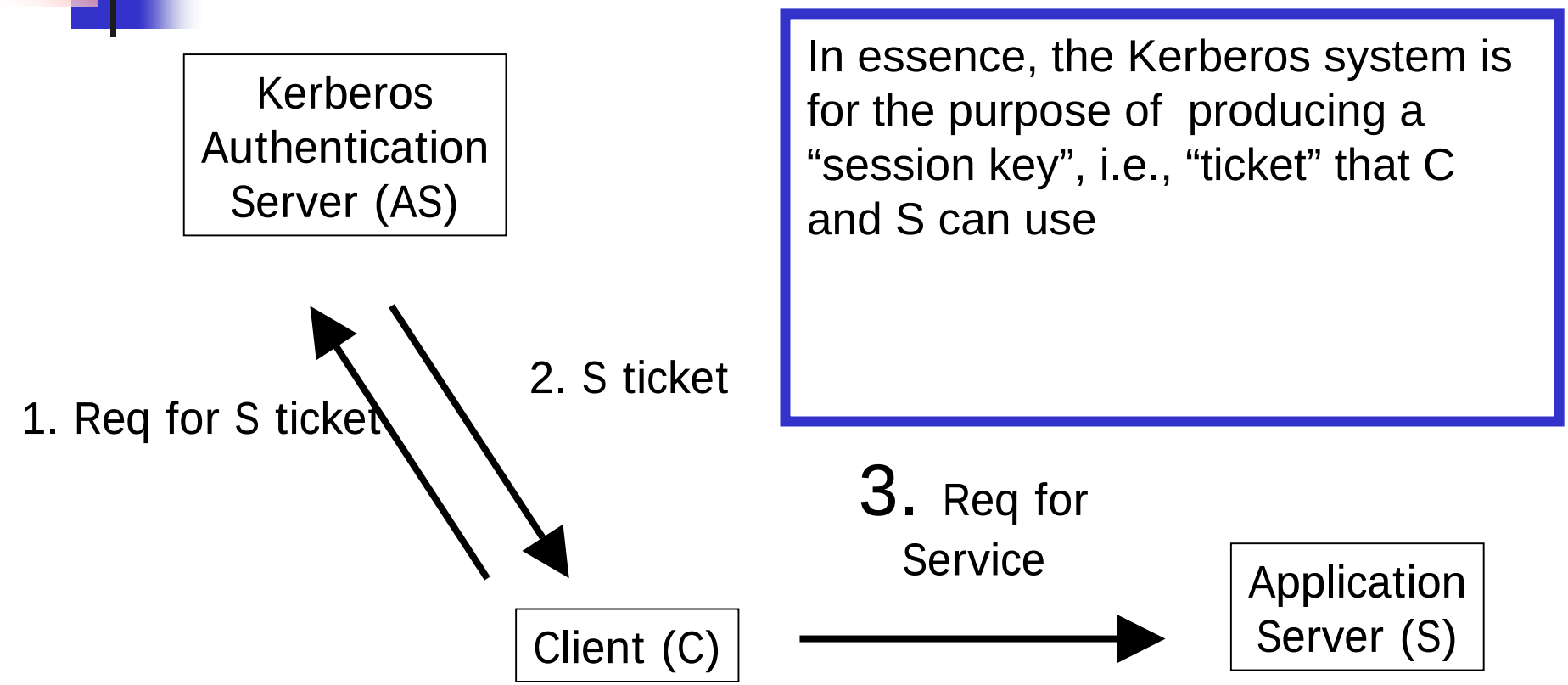


1. Req for application server ticket
2. Ticket for application server
3. Req for service

Kerberos Terminology and Abbreviations

- c client id
- s server id
- addr client's IP address
- life lifetime of ticket
- TGS ticket granting server
- K_x x's secret key (x being a client or server)
- $K_{x,y}$ session key for x and y
- $\{abc\}K_x$ abc encrypted in x's key
- $T_{x,y}$ x's ticket to use y (used many times)
- A_x authenticator for x, containing x's name (e.g., zheng.ma@yale.edu, current time (to defeat replay) and checksum

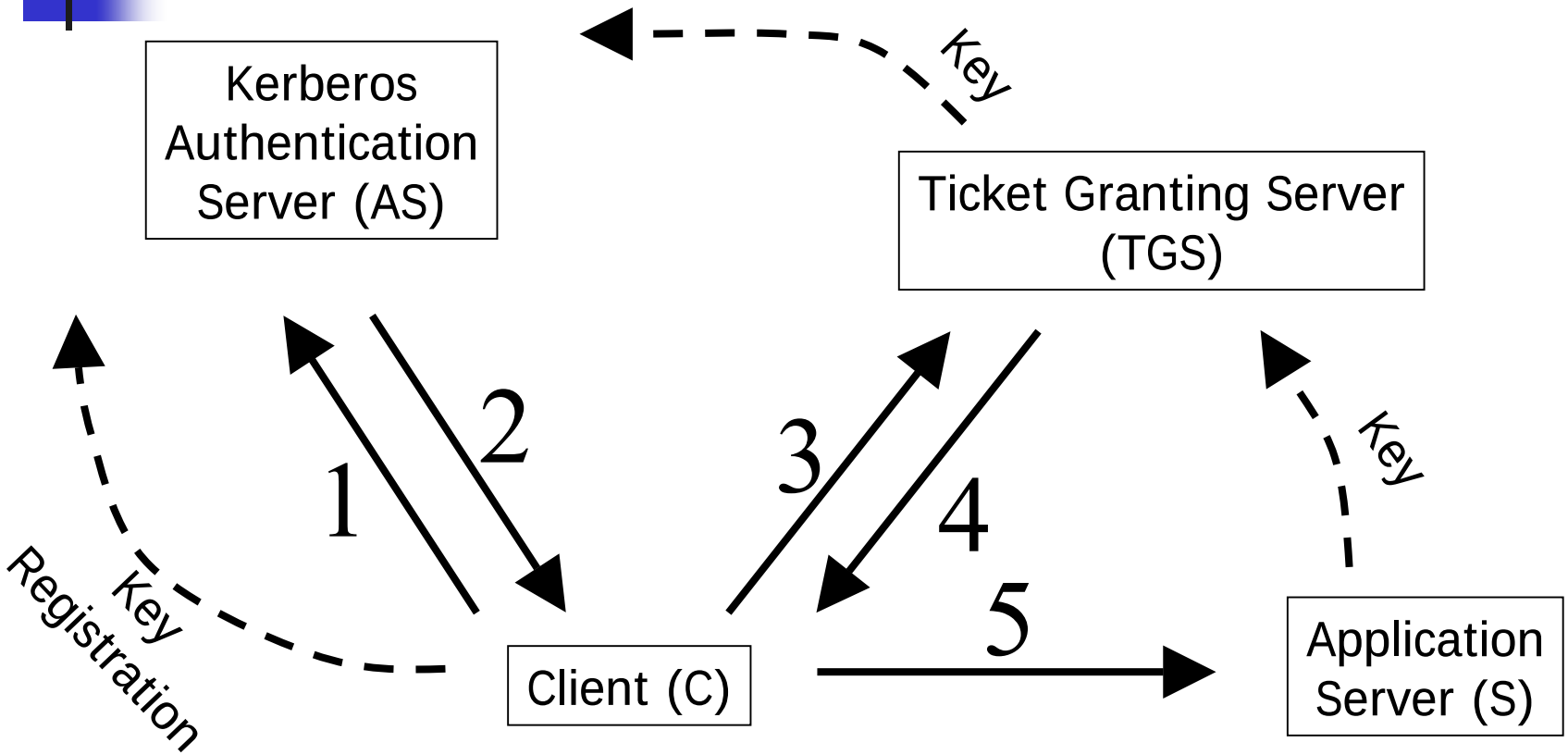
Kerberos Authentication (Detail)



1. c, s
2. $\{K_{c,s}, \{T_{c,s}\}K_s\}K_c$
3. $\{A_c\}K_{c,s}, \{T_{c\,s}\}K_s$

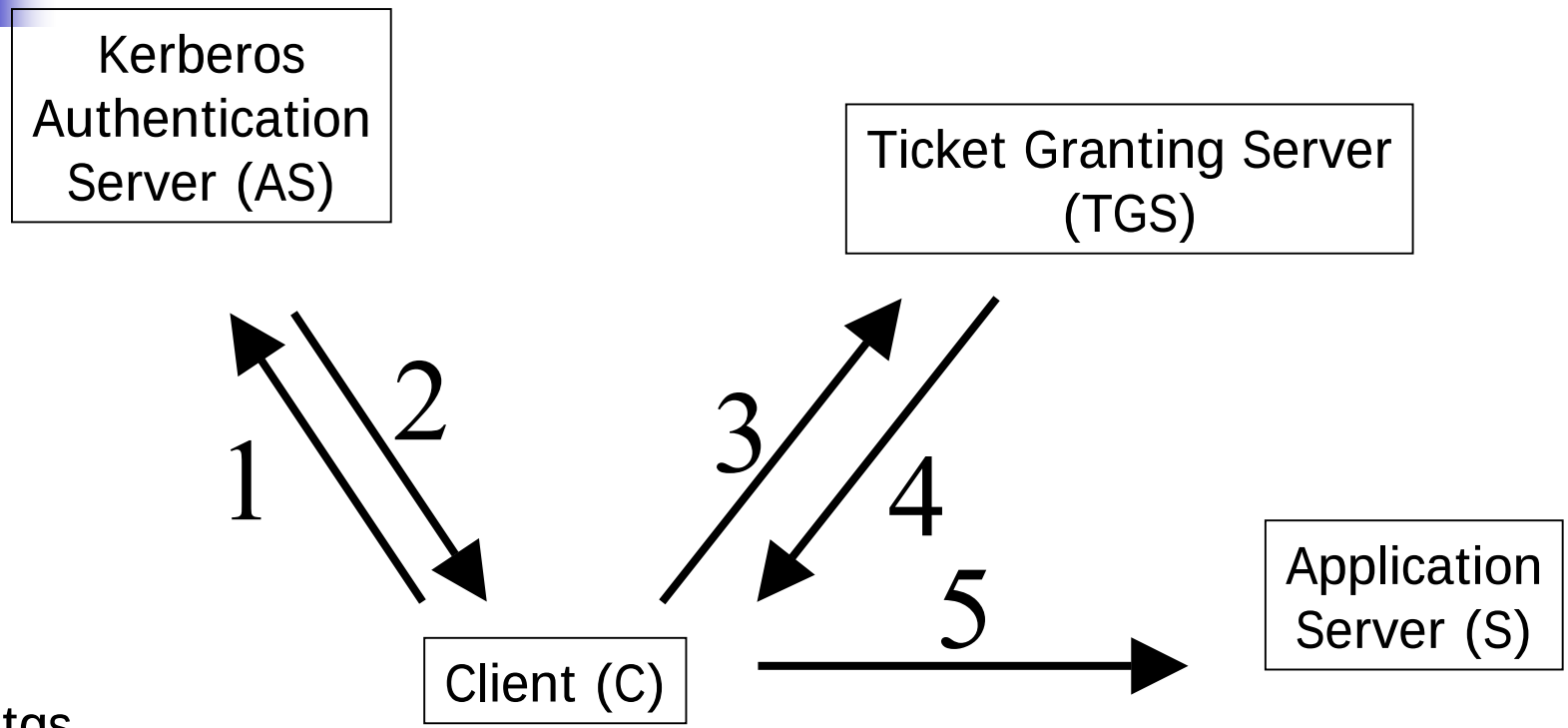
- $T_{c,s}$ contains session key $K_{c,s}$
- In step 2, user enters password to decrypt the received message
- If S can decode $\{A_c\}K_{c,s}$, then user must have entered the correct password!

Kerberos Authentication w/ TGS



1. Req for TGS ticket
2. Ticket for TGS
3. Req for application server ticket
4. Ticket for application server
5. Req for service

Kerberos Authentication w/ TGS (Detail)

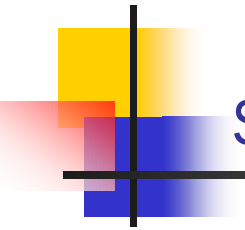


1. c, tgs
2. $\{K_{c,\,tgs}, \{T_{c,\,tgs}\}K_{tgs}\}K_c$
3. s, $\{A_c\}K_{c,\,tgs}$, $\{T_{c,\,tgs}\}K_{tgs}$
4. $\{K_{c,s}, \{T_{c,s}\}K_s\}K_{c,tgs}$
5. $\{A_c\}K_{c,s}$, $\{T_{c\,s}\}K_s$

In step 4 client uses stored $K_{c,tgs}$ rather than user entering password. This is convenient. But system now needs to believe that client can be trusted for the period when $K_{c,tgs}$ is valid

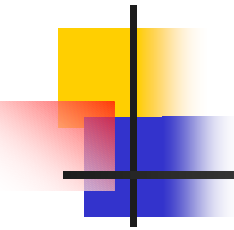
Kerberos' Stateless Model

- TGS does not send $\{K_{c,s}\}K_s$ to S directly. Instead, TGS sends $\{T_{c,s}\}K_s$, with $T_{c,s}$ containing $K_{c,s}$, to C and let C forward it to S
  - Otherwise, S would need to keep state, i.e., keep received $K_{c,s}$ around, and this would complicate implementation
- In general, servers do not talk to each other directly. Clients initialize transactions and complete them
- This stateless model is simple and elegant



Scaling Kerberos

- To scale, divide the network into *realms* each having its own AS and its own TGS
- To allow for cross-realm authentication, i.e., to allow users in one realm to access services in another, the user's realm may register a *remote* TGS (RTGS) in the service's realm
- To reduce cross-realm registration, use a hierarchy of realms

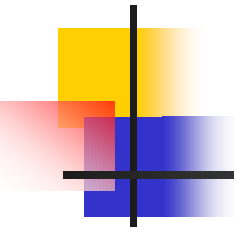


Kerberos Authorization and Accounting

- In Kerberos, authorization and accounting are supported by having AS inserting some predefined information, e.g., access control list, in the ticket
  - It is encrypted in the ticket, so it is tamper-proof
  - The information are left for the server to interpret

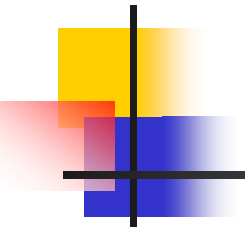
Advantages of Kerberos

- Passwords aren't exposed to eavesdropping
- Password is only typed to the local workstation
  - It never travels over the network
  - It is never transmitted to a remote server
- Password guessing more difficult
- Single Sign-on
  - More convenient: only one password, entered once
  - Users may be less likely to store passwords
- Stolen tickets hard to reuse
  - Need authenticator as well, which can't be reused
- Much easier to effectively secure a small set of limited access machines (the AS's)
- Easier to recover from host compromises
- Centralized user account administration



Kerberos caveats

- Kerberos server can impersonate anyone
- AS is a single point of failure
  - Can have replicated AS's
- AS could be a performance bottleneck
  - Everyone needs to communicate with it frequently
  - Not a practical concern these days
  - Having multiple AS's alleviates the problem
- If local workstation is compromised, user's password could be stolen by a trojan horse
  - Only use a desktop machine or laptop that you trust
  - Use hardware token pre-authentication
- Kerberos vulnerable to password guessing attacks
  - Choose good passwords!
  - Use hardware pre-authentication
    - Hardware tokens, Smart cards etc



Summary of Kerberos

- Kerberos provides an authentication server (AS) that issues "tickets" or "session keys" to clients for various services
  - The $O(N^2)$ password management problem is alleviated
  - In addition, by using the TGS, users no longer need to type in passwords all the time
- AS and TGS need to be trusted
  - For large systems, should PKI (Public Key Infrastructure) be used instead?
  - For small systems, do we need Kerberos? SSH may be just fine.

Ssshhhhh....

- Be very quiet so Eve can’t hear anything
- Encrypt the communication between the terminal and the server
- How?

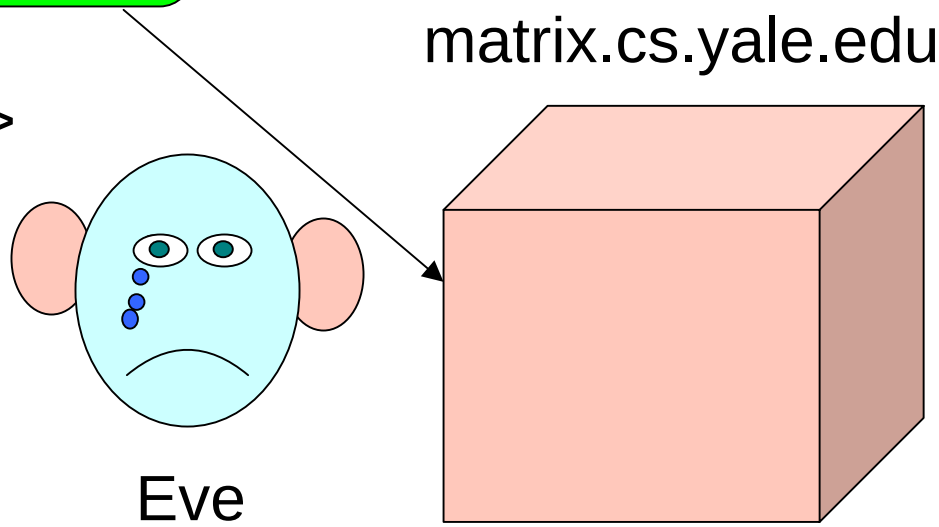
Simplified SSH Protocol

Terminal

```
Login: zm25
Password: ***********
```

login sends

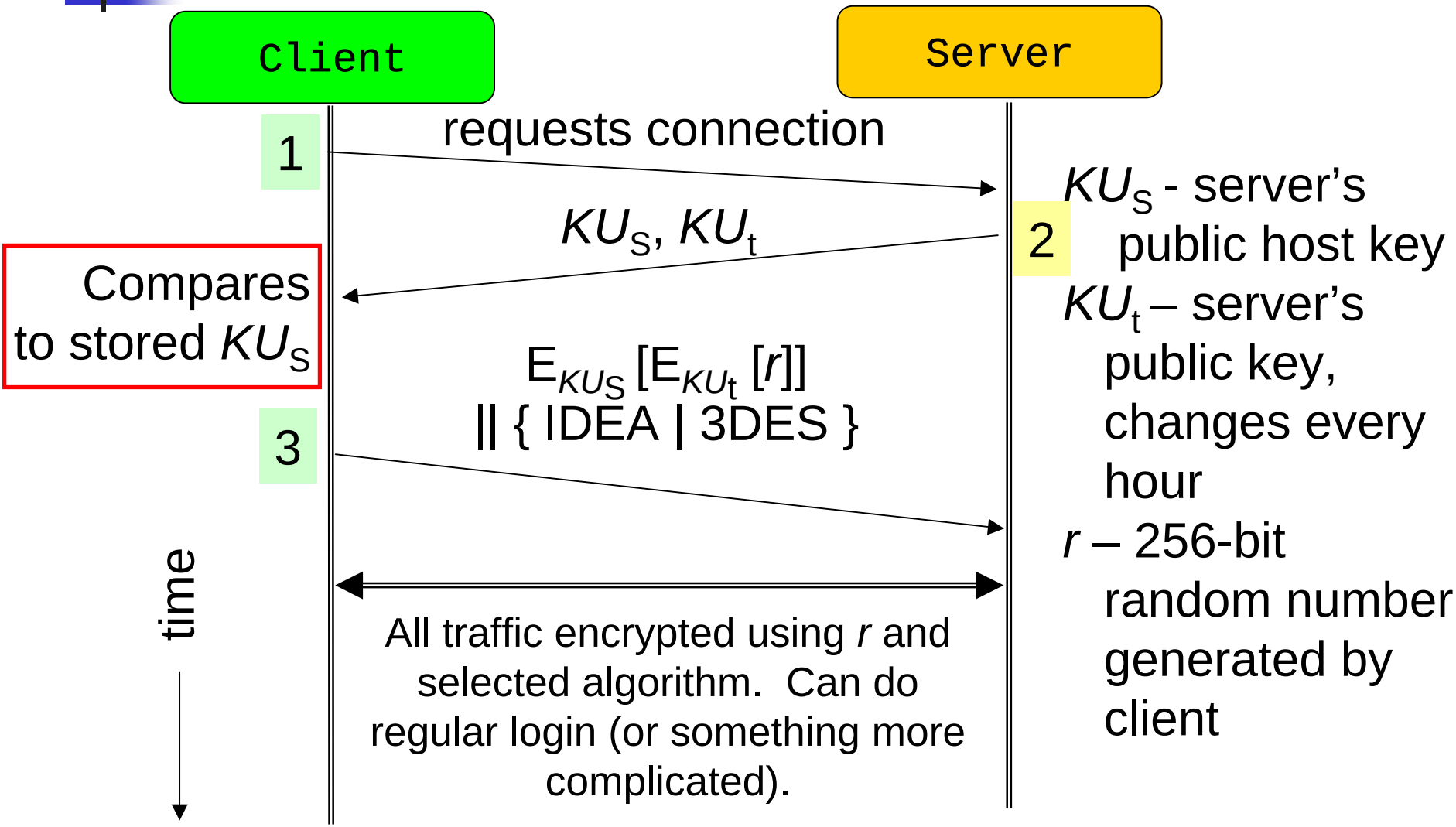
$E_{KU_{matrix}}$<“zm25”, *password*>



Eve

Can’t decrypt without KR_{matrix}

Actual SSH Protocol

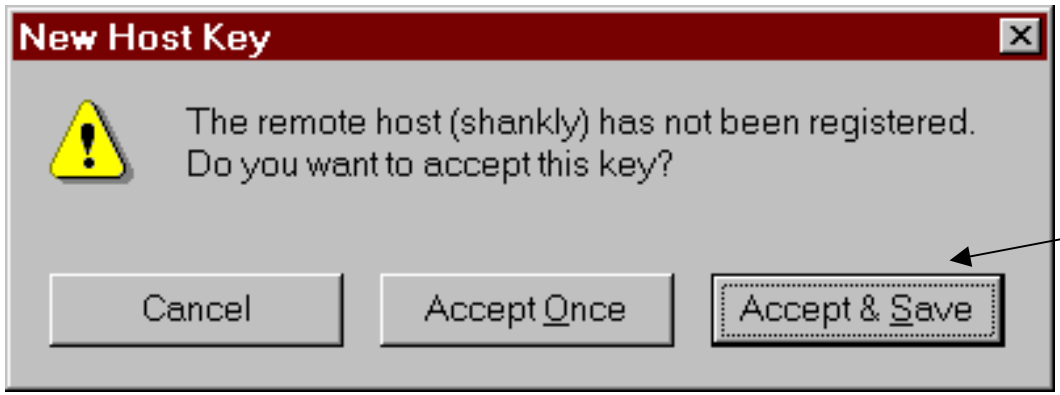


Comparing to stored KU_S

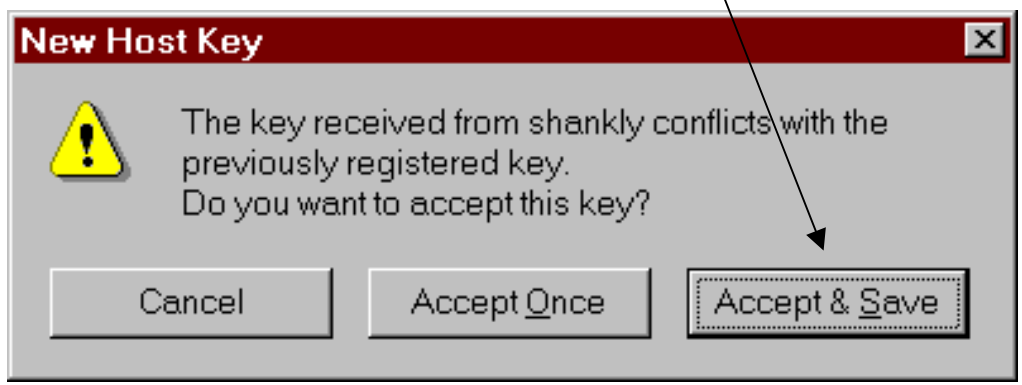
- It better be stored securely
  - PuTTY stores it in windows registry (HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\SshHostKeys)

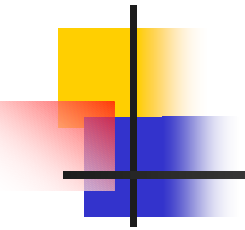
Accept and Save

SecureCRT



Default choice!

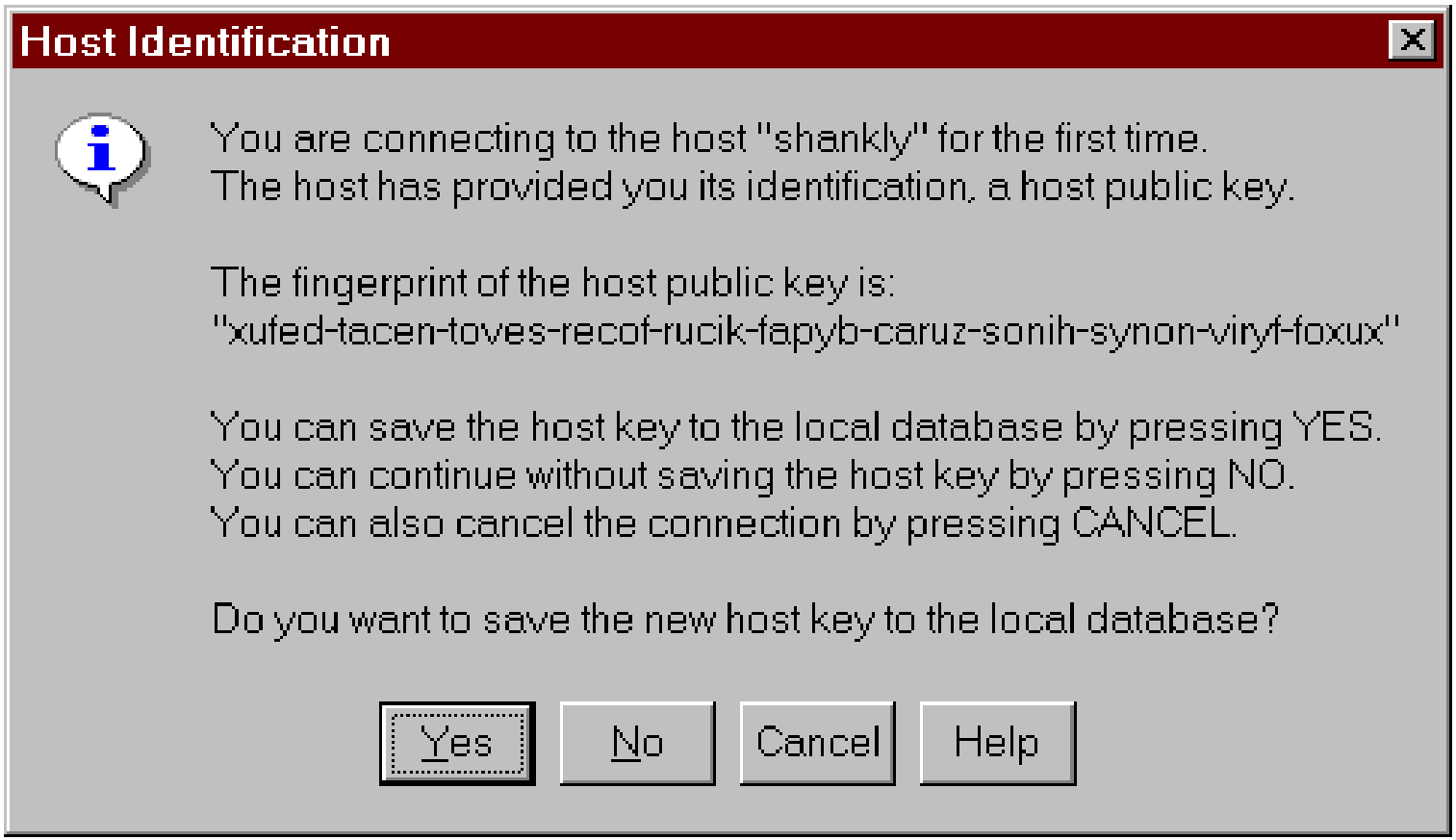




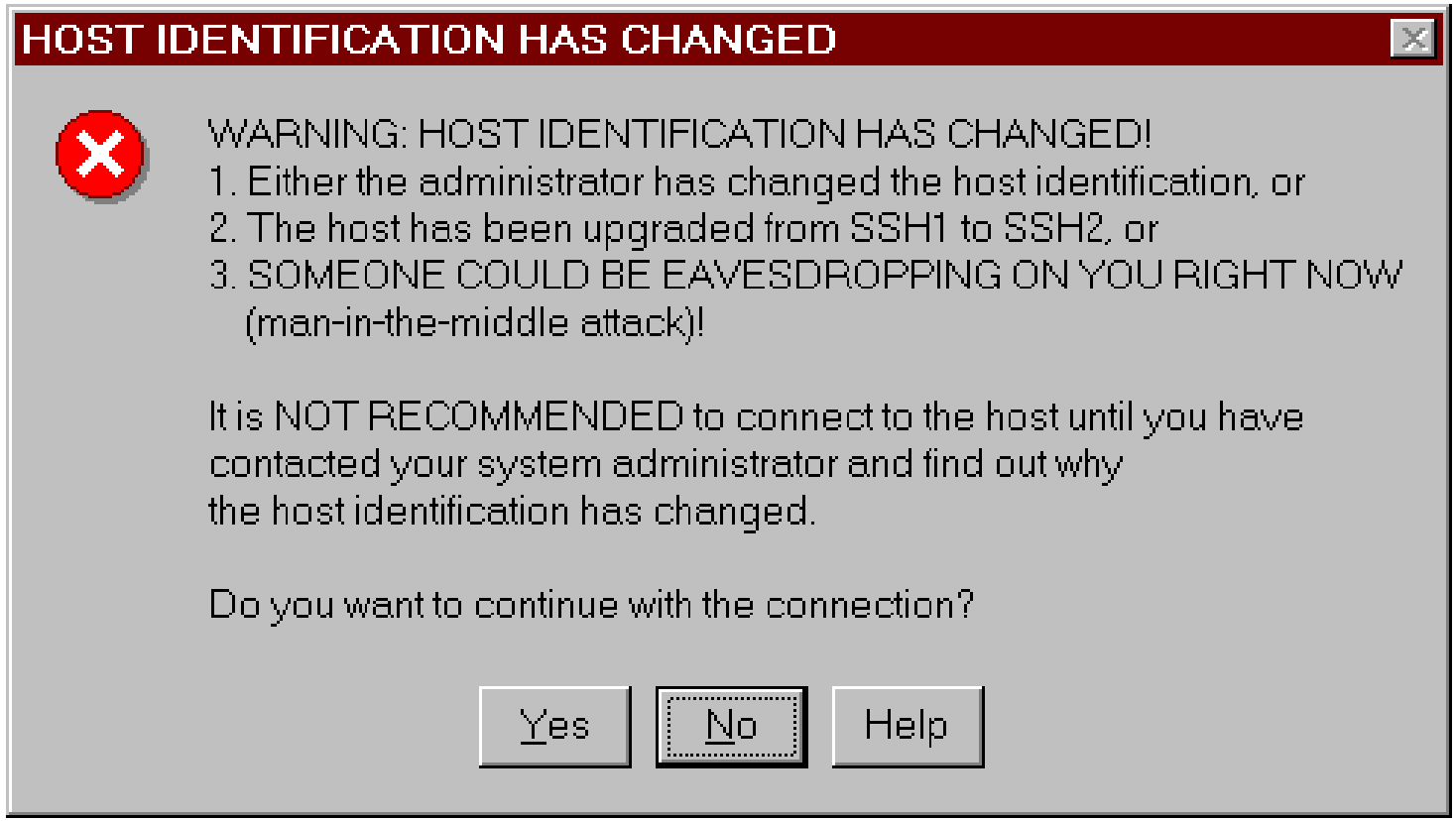
“Usability in normal environments has been a major design concern from the beginning, and SSH attempts to make things as easy for normal users as possible while still maintaining a sufficient level of security.”

Tatu Ylonen, *SSH – Secure Login Connections over the Internet*, June 1996.

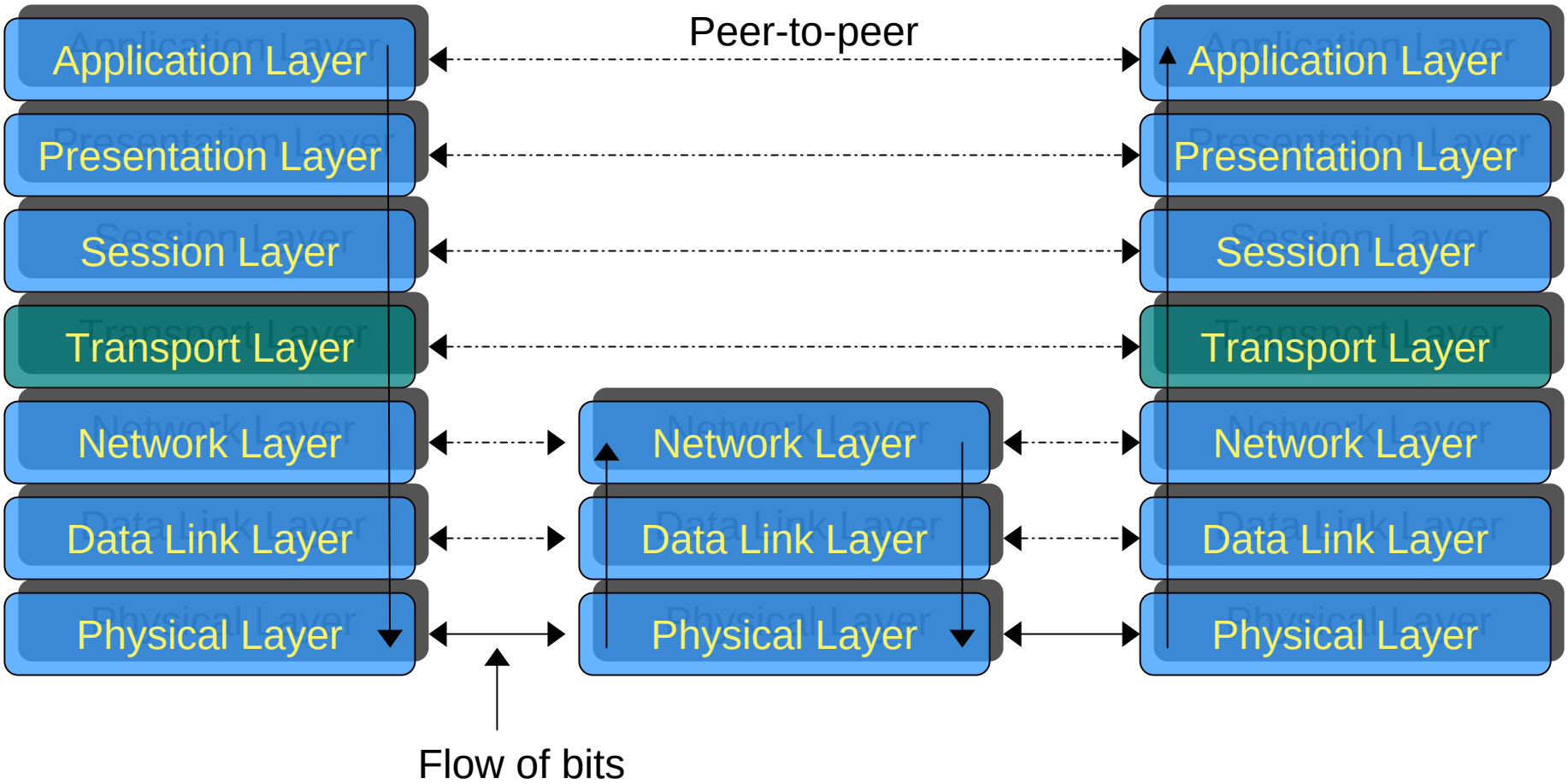
ssh.com's SSH

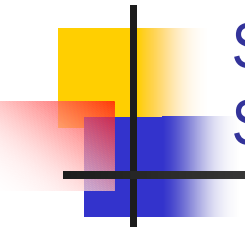

Host Identification
You are connecting to the host "shankly" for the first time.
The host has provided you its identification, a host public key.
The fingerprint of the host public key is:
"xufed-tacen-toves-recof-rucik-fapyb-caruz-sonih-synon-viryf-foxux"
You can save the host key to the local database by pressing YES.
You can continue without saving the host key by pressing NO.
You can also cancel the connection by pressing CANCEL.
Do you want to save the new host key to the local database?
Yes
No
Cancel
Help

ssh Error


HOST IDENTIFICATION HAS CHANGED
WARNING: HOST IDENTIFICATION HAS CHANGED!
1. Either the administrator has changed the host identification, or
2. The host has been upgraded from SSH1 to SSH2, or
3. SOMEONE COULD BE EAVESDROPPING ON YOU RIGHT NOW
(man-in-the-middle attack)!
It is NOT RECOMMENDED to connect to the host until you have
contacted your system administrator and find out why
the host identification has changed.
Do you want to continue with the connection?
Yes
No
Help

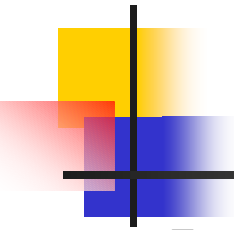
ISO/OSI Model SSL: Security at Transport Layer





Security at the Transport Layer
Secure Socket Layer (SSL)

- Developed by Netscape to provide security in WWW browsers and servers
- SSL is the basis for the Internet standard protocol – Transport Layer Security (TLS) protocol (compatible with SSLv3)
- Key idea: *Connections* and *Sessions*
  - A SSL session is an association between two peers
  - An SSL connection is the set of mechanisms used to transport data in an SSL session



Secure Socket Layer (SSL)

- Each party keeps session information
  - Session identifier (unique)
  - The peer's X.503(v3) certificate
  - Compression method used to reduce volume of data
  - Cipher specification (parameters for cipher and MAC)
  - Master secret of 48 bits
- Connection information
  - Random data for the server & client
  - Server and client keys (used for encryption)
  - Server and client MAC key
  - Initialization vector for the cipher, if needed
  - Server and client sequence numbers
- Provides a set of supported cryptographic mechanisms that are setup during negotiation (handshake protocol)

An example of key exchange using public/private keys

- SSL (Secure Socket Layer) and TLS (Transport Layer Security) use public/private keys to exchange a secret key used during a session
- The SSL handshake consists of several steps, as follows:

Step 1: The client contacts the server and sends SSL version number, a random number X, and some additional information

Step 2: The server sends the client the SSL version number, random number Y, and its public key (packaged into a certificate)

Step 3: The client verifies that the server is who is says it is by examining the certificate (more on this in a bit)

Step 4: The client creates a "premaster secret" using X, Y, and other information. It encrypts the secret using the server's public key.

Step 5: If the server has requested authentication, the client sends its own certificate and the premaster secret to the server

Step 6: The server authenticates the client by examining the client's certificate, uses its private key to decrypt the premaster secret, then uses it to generate the master secret. The client also generates the master secret.

Step 7: Both the client and the server use the master secret to generate the session secret key

Steps 8 (9): The client (server) sends a message to the server (client) telling it that it will use the secret key. It sends a second message encrypted with the secret key

Acknowledgements

- Credits of some slides and images:

- http://www.upenn.edu/computing/pennkey/docs/kerbpres/200207Kerberos.htm
- http://www.eecs.harvard.edu/cs143/
- http://www.cs.virginia.edu/~evans/cs551/