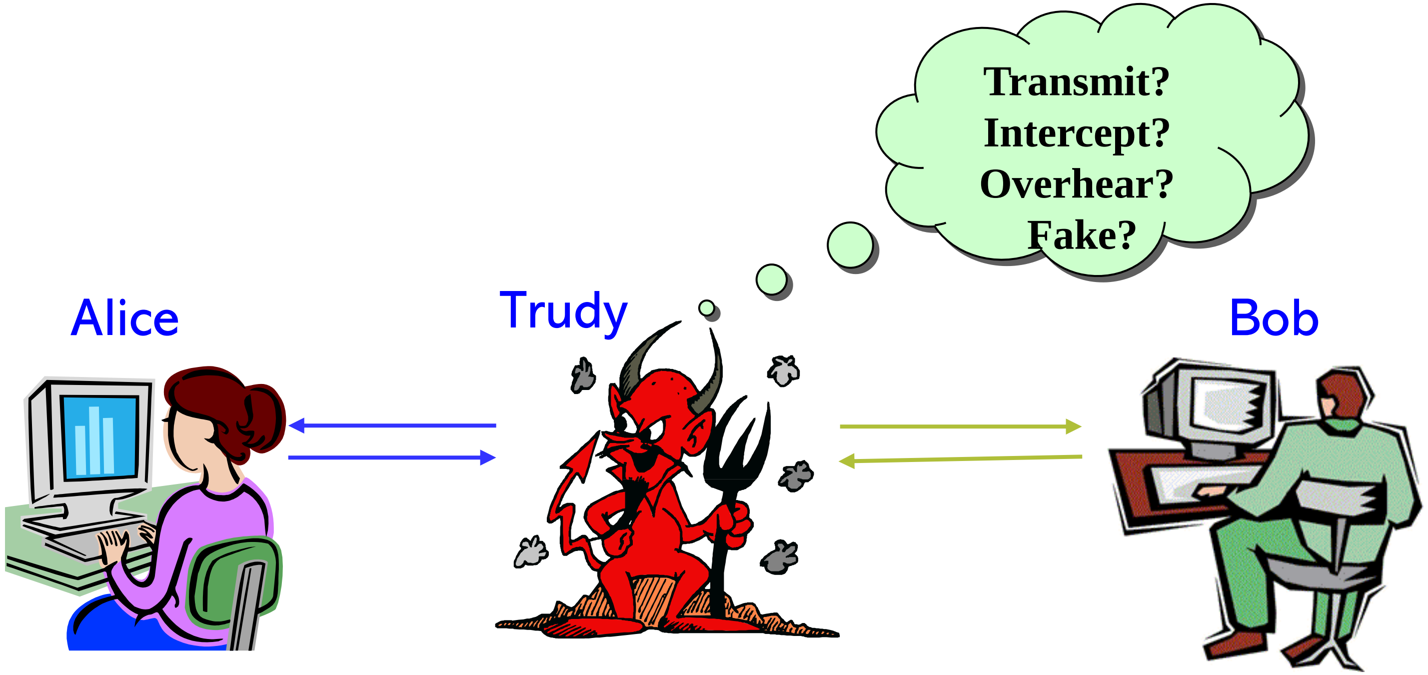
Authentication Protocols

(CS-352)

Authentication Protocols

- Used to convince parties of each other's identity and to exchange session keys



Authentication Protocols

- Published protocols are often found to have flaws and need to be modified
- Key issues are
  - Confidentiality
    - To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form.
  - Timeliness – to prevent replay attacks

Replay Attacks

- A valid signed message is copied and later resent
  - Simple replay: the opponent simply copies a message and replays it later

Replay Attacks

- A valid signed message is copied and later resent
  - Repetition that can be logged: an opponent can replay a timestamped message within the valid time window
  - Repetition that cannot be detected: may arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.

Replay Attacks

- A valid signed message is copied and later resent
  - Backward replay without modification: a replay back to the sender
    - When using symmetric encryption, the sender cannot easily recognize the difference between messages sent and messages received.

Replay Attacks

- Countermeasures
  - Attach a sequence number to each message used in an authentication exchange
    - Generally impractical – requires each party to keep track of the last sequence number for each claimant it has dealt with
  - Timestamps: party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time.
    - Needs synchronized clocks

Replay Attacks (Cont.)

- Countermeasures
  - Nonce: a random number that illustrates the freshness of a session.
    - Party A sends B a nonce and requires that the subsequent response received from B contains the correct nonce value.

Using Symmetric Encryption

- As discussed previously can use a two-level hierarchy of keys
- Usually with a trusted Key Distribution Center (KDC)
    - Each party shares own master key with KDC
    - KDC generates session keys used for connections between parties
    - Master keys used to distribute these to them

Needham-Schroeder Symmetric Key Protocol (Revisited)

- Original third-party key distribution protocol
- For session between A B mediated by KDC
- Protocol:

  1. A->KDC: $ID_A \| ID_B \| N_1$

  2. KDC -> A: $E_{Ka}[Ks \| ID_B \| N_1 \| E_{Kb}[Ks\|ID_A]]$

  3. A -> B: $E_{Kb}[Ks\|ID_A]$

  4. B -> A: $E_{Ks}[N_2]$

  5. A -> B: $E_{Ks}[f(N_2)]$

Needham-Schroeder Symmetric Key Protocol (Revisited)

3. A -> B: $E_{Kb}[Ks||ID_A]$
4. B -> A: $E_{Ks}[N_2]$
5. A -> B: $E_{Ks}[f(N_2)]$

- Suppose that an attacker X has been able to compromise an old session key.

Attack: Needham-Schroeder Protocol (Revisited)

3. A -> B: $E_{Kb}[K_s \| ID_A]$
4. B -> A: $E_{Ks}[N_2]$
5. A -> B: $E_{Ks}[\mathrm{f}(N_2)]$

- Suppose that an attacker X has been able to compromise an old session key.
- X can impersonate A and trick B into using the old key by simply replaying step 3.
- Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay.
- X then intercepts the step 4 and sends bogus messages to B that appear to B to come from A using an authenticated session key.

Solution: Needham-Schroeder Protocol (Revisited)

- Use a timestamp T that assures A and B that the session key has only just been generated.
- Revised protocol:

1. A->KDC: $ID_A \| ID_B \| N_1$

2. KDC -> A: $E_{Ka}[Ks \| ID_B \| N_1 \| E_{Kb}[Ks\|ID_A\|T]]$

3. A -> B: $E_{Kb}[Ks\|ID_A\|T]$

4. B -> A: $E_{Ks}[N_2]$

5. A -> B: $E_{Ks}[f(N_2)]$

Timestamp

- Principals can verify the timeliness by checking:

$$|Clock - T| < \Delta t_1 + \Delta t_2$$

  - Δt_1: The estimated normal discrepancy between the KDC's clock and the local clock (principals' clock)
  - Δt_2: The expected network delay time
- Need to synchronize clock

Suppress-Replay Attacks

- Suppress-replay attacks: when the sender's clock is ahead of the receiver's clock, the opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the receiver's site.

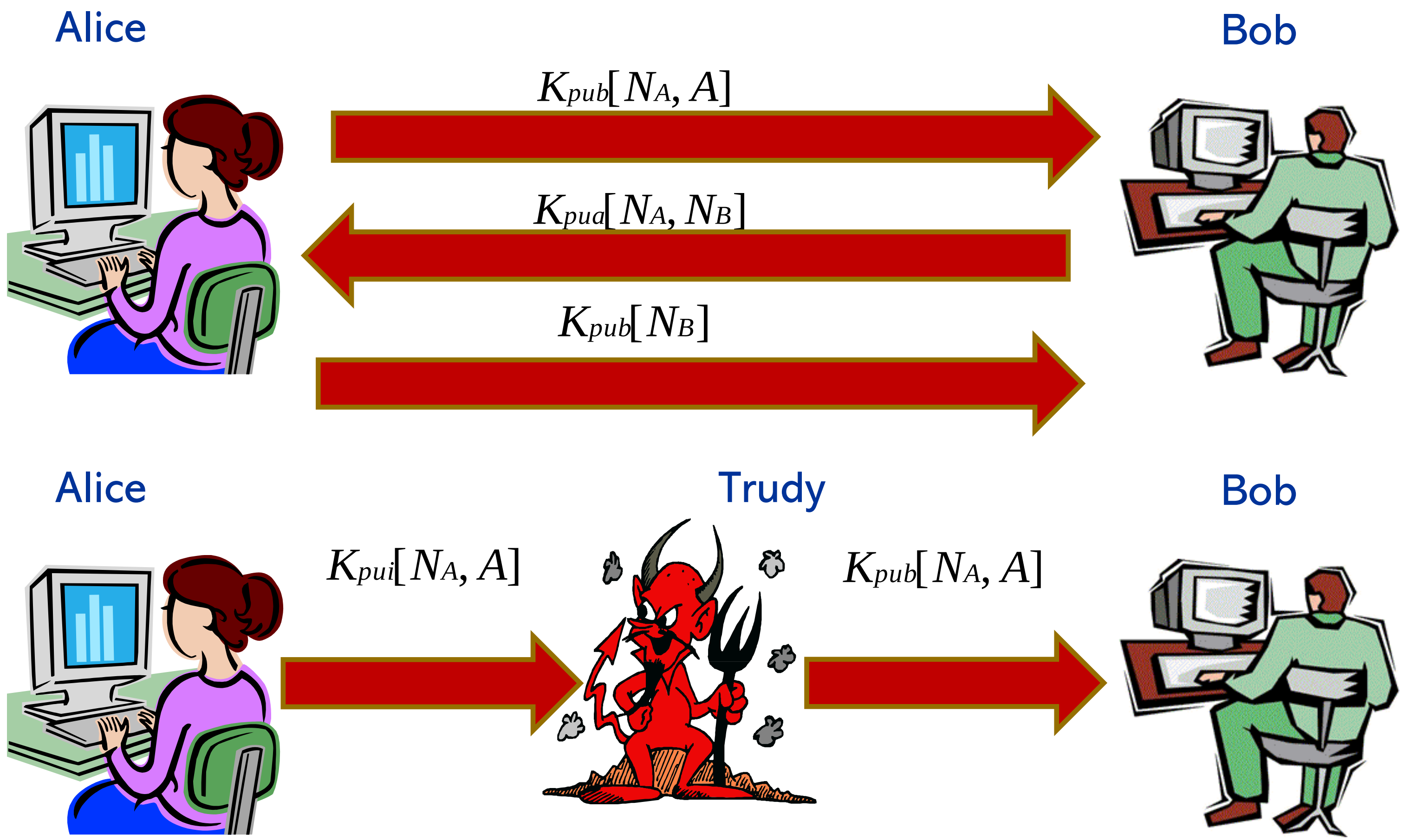
Suppress-Replay Attacks

- Suppress-replay attacks: when the sender's clock is ahead of the receiver's clock, the opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the receiver's site.
- Countermeasure:
  - Enforce the requirement that parties regularly check their clocks against the KDC's clock.
  - Rely on handshaking protocols using nonces.

Using Public-Key Encryption

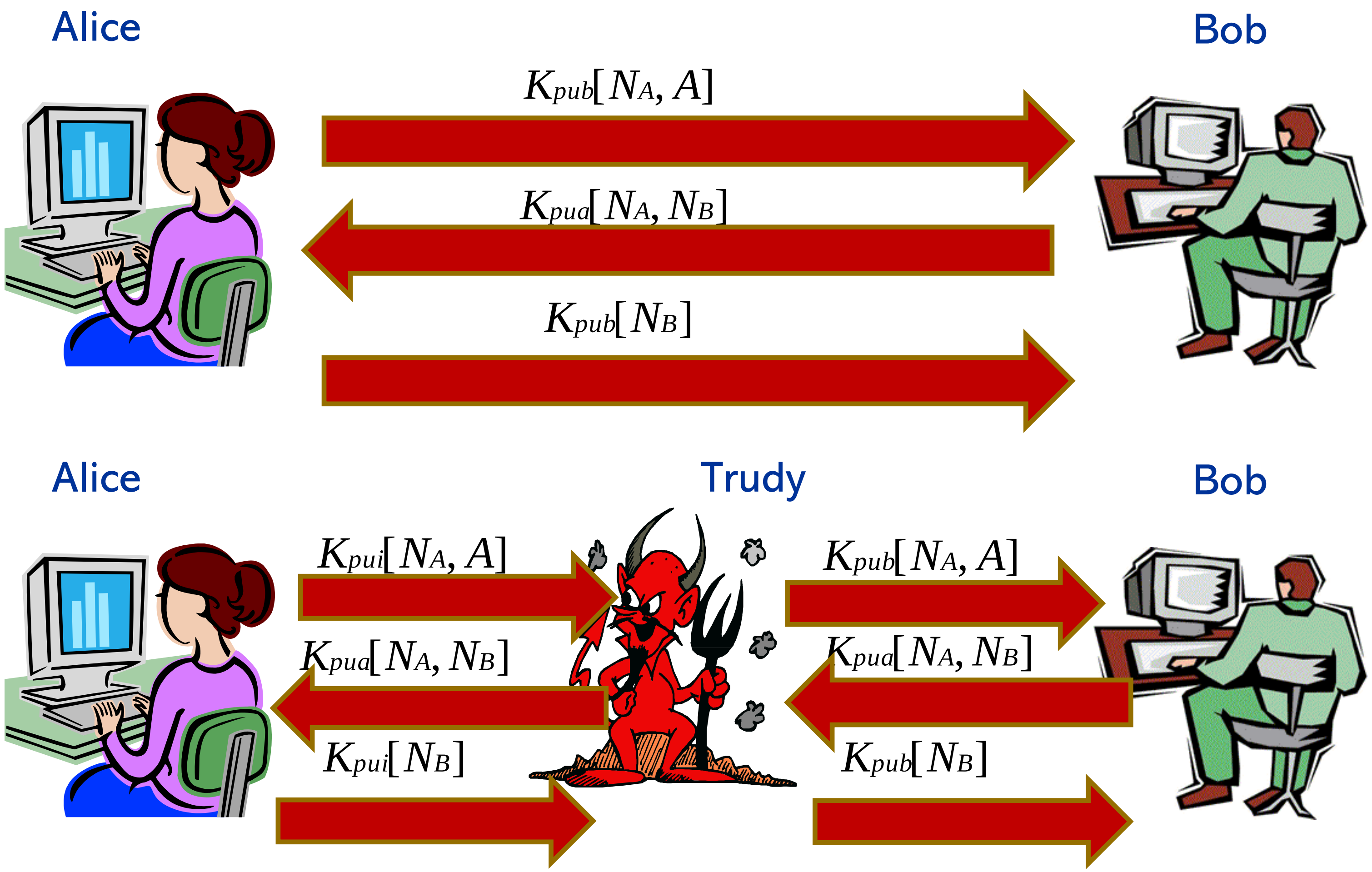
- Have a range of approaches based on the use of public-key encryption
- Need to ensure we have correct public keys for other parties
- Various protocols exist using timestamps or nonces

Needham-Schroeder Public Key Protocol



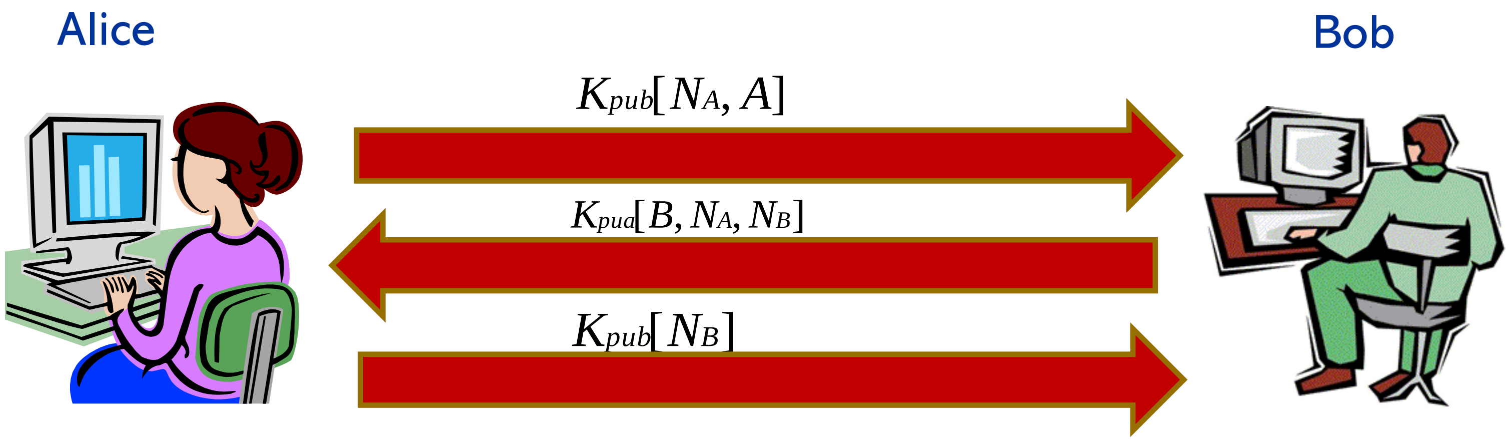
[Lowe, 1995]

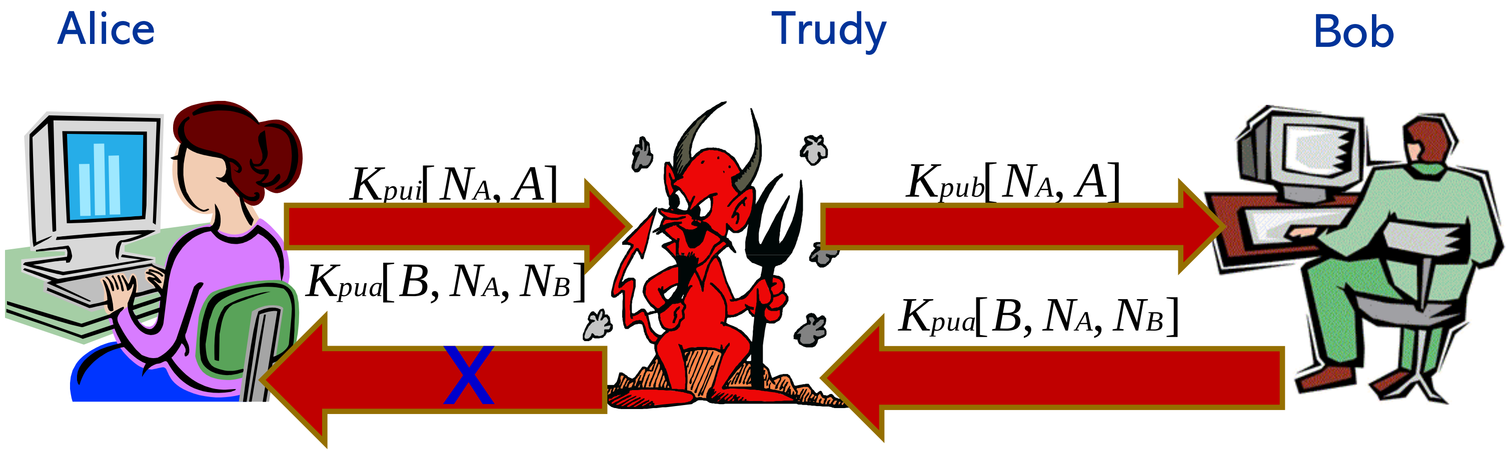
Needham-Schroeder Public Key Protocol



[Lowe, 1995]

Needham-Schroeder-Lowe Public Key Protocol





One-Way Authentication

- Required when sender and receiver are not in communications at same time (eg. email)
- Have header in clear so can be delivered by email system
- May want contents of body protected & sender authenticated

Using Symmetric Encryption

- Can refine use of KDC but cannot have exchange of nonces:

1. A->KDC: $ID_A \,||\, ID_B \,||\, N_1$
2. KDC -> A: $\mathrm{E}_{\mathrm{Ka}}[\mathrm{Ks} \,||\, ID_B \,||\, N_1 \,||\, \mathrm{E}_{Kb}[Ks||ID_A]\,]$
3. A -> B: $E_{Kb}[Ks||ID_A] \,||\, \mathrm{E}_{\mathrm{Ks}}[\mathrm{M}]$

Using Symmetric Encryption

- Can refine use of KDC but cannot have exchange of nonces:

  1. A->KDC: $ID_A \| ID_B \| N_1$

  2. KDC -> A: $E_{Ka}[Ks \| ID_B \| N_1 \| E_{Kb}[Ks\|ID_A]]$

  3. A -> B: $E_{Kb}[Ks\|ID_A] \| E_{Ks}[M]$

- Guarantees that only the intended recipient of a message will be able to read it.
- Does not protect against replays
  - Could rely on timestamp in message, though email delays make this problematic

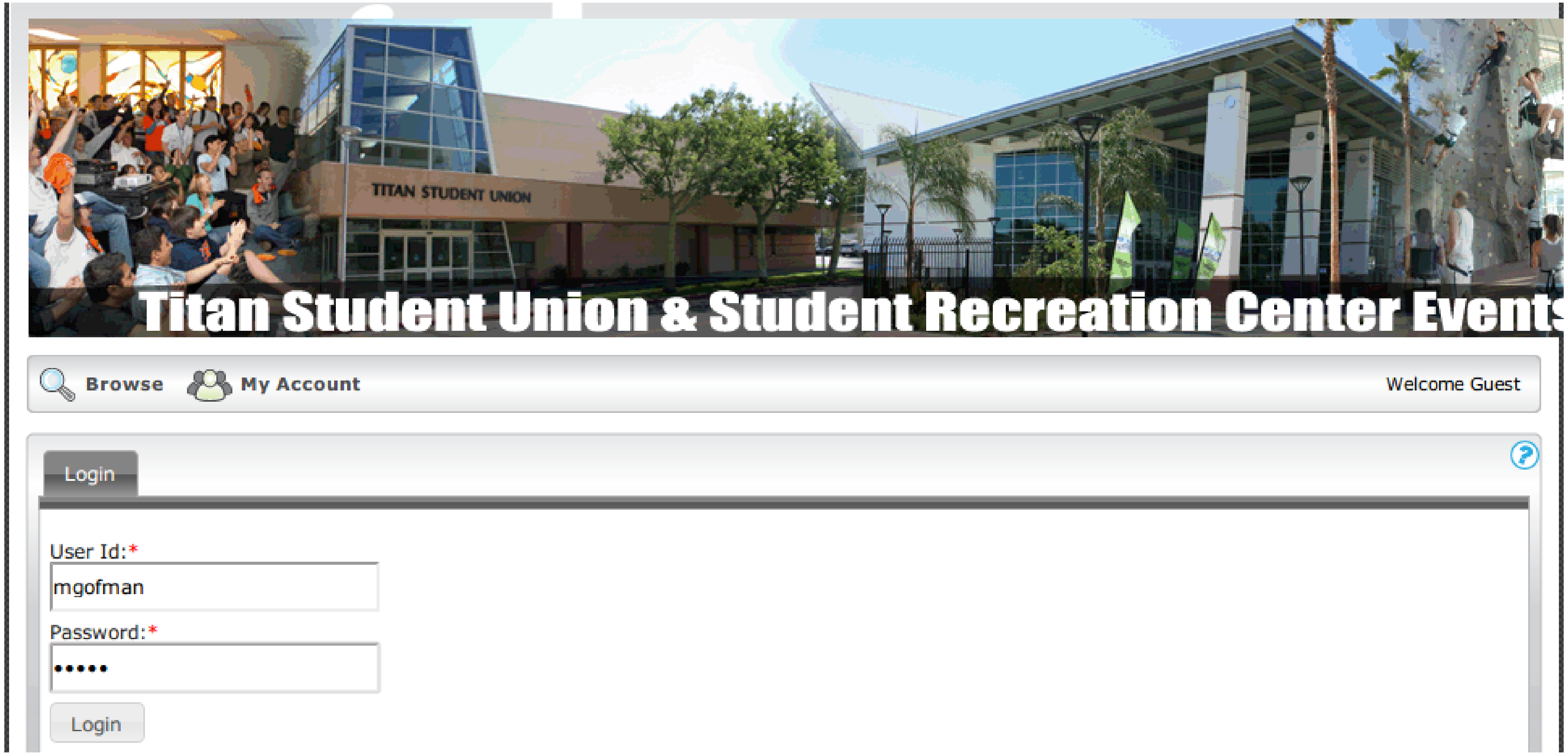
Secure Sockets Layer/Transport Layer Security (SSL/TLS) and Web Security

Web Security

- World Wide Web is fundamentally a client/server application running over internet and TCP/IP intranet.
- Web now widely used by business, government, individuals
- But Web is vulnerable

Example of a Web Vulnerability (1)

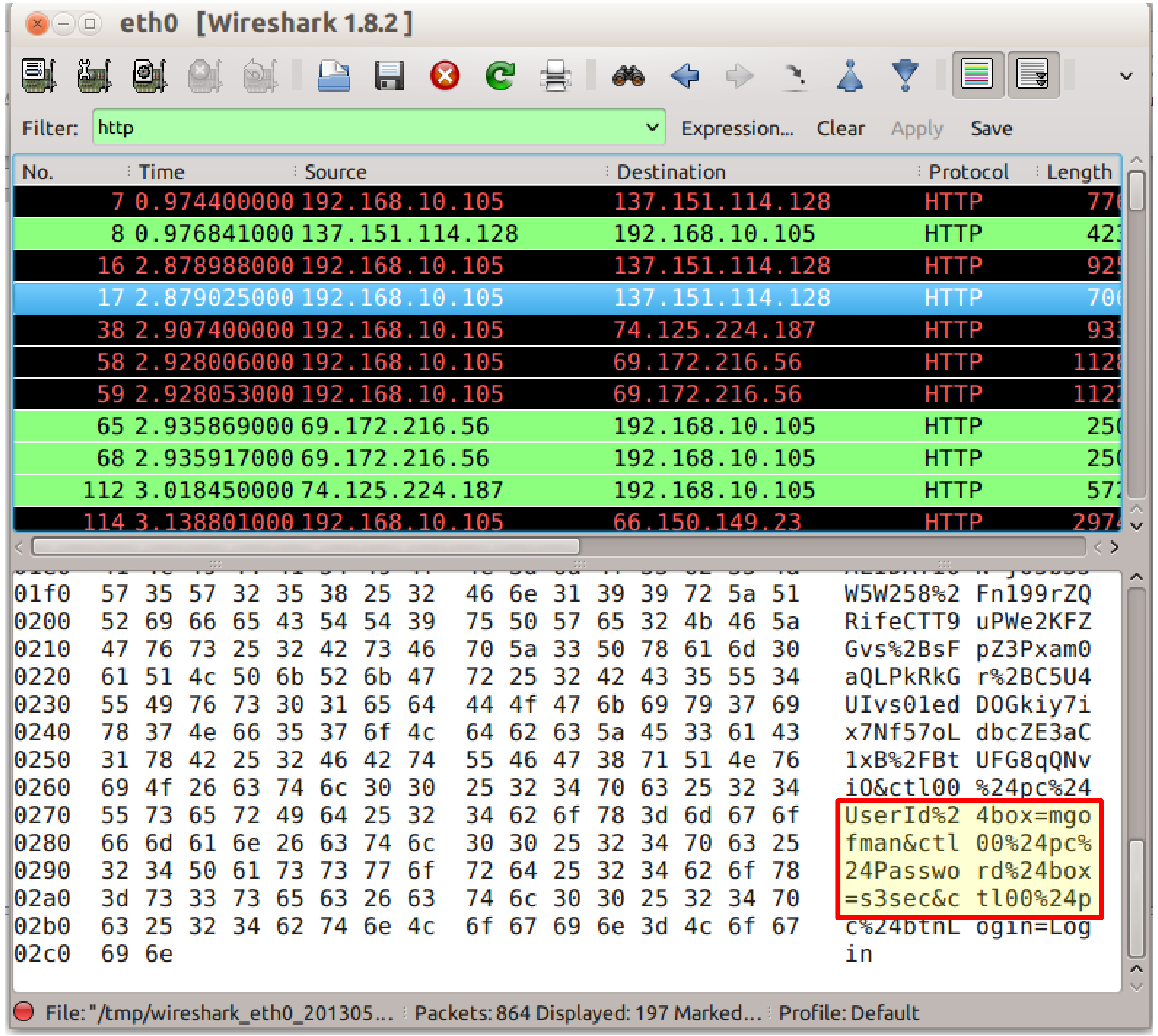
- Website: http://asi.fullerton.edu/VirtualEMS/Login.aspx



- Steps:
  - Enter user name: mgofman
  - Enter password: s3sec
  - Use WireShark packet sniffer to observe the website traffic...

Example of a Web Vulnerability (2)

- Traffic captured using WireShark:



Summary of Web-based Attacks

	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerabilty to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

Hypertext Transfer Protocol Secure (HTTPS)

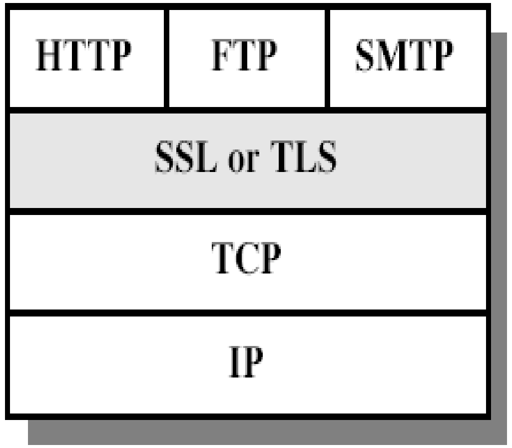
- A combination of the http protocol and a network security protocol
- Also known as Hypertext Transfer Protocol over Secure Socket Layer

Hypertext Transfer Protocol Secure (HTTPS)

- The administrator must create a public X.509 key certificate for the Web server.
- This certificate must be signed by a certificate authority.
    - SSL certificate providers: Verisign, Thawte, InstantSSL, Entrust, Baltimore, Geotrust etc.
- Web browsers are distributed with the public key of major certificate authorities so that they can verify certificates signed by them.

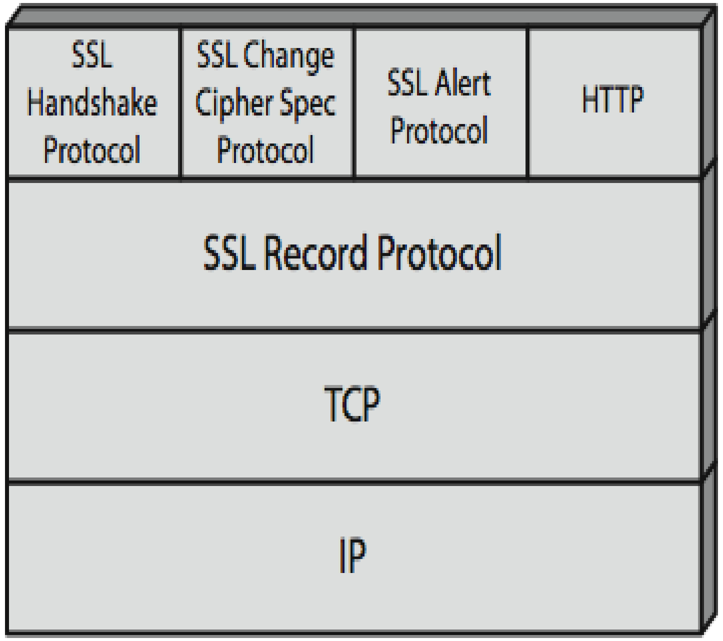
SSL/TLS (Secure Socket Layer/Transport Layer Security)

- A cryptographic protocol that provides security for communications over networks.
- One of the most widely used Web security mechanisms.
- Transport layer security service - designed to make use of TCP to provide a reliable end-to-end security service.
- Originally developed by Netscape
- Subsequently became Internet standard known as TLS (Transport Layer Security)



SSL/TLS Architecture

- Has two layers of protocols
  - Level 1:
    - SSL Record Protocol: provides basic security services to various higher-layer protocols.
  - Level 2:
    - Hypertext Transfer Protocol (HTTP): which provides the transfer service for Web client/server interaction, can operate on top of SSL.
    - Three higher-layer protocols: used in the management of SSL exchanges.



SSL/TLS Handshake Protocol

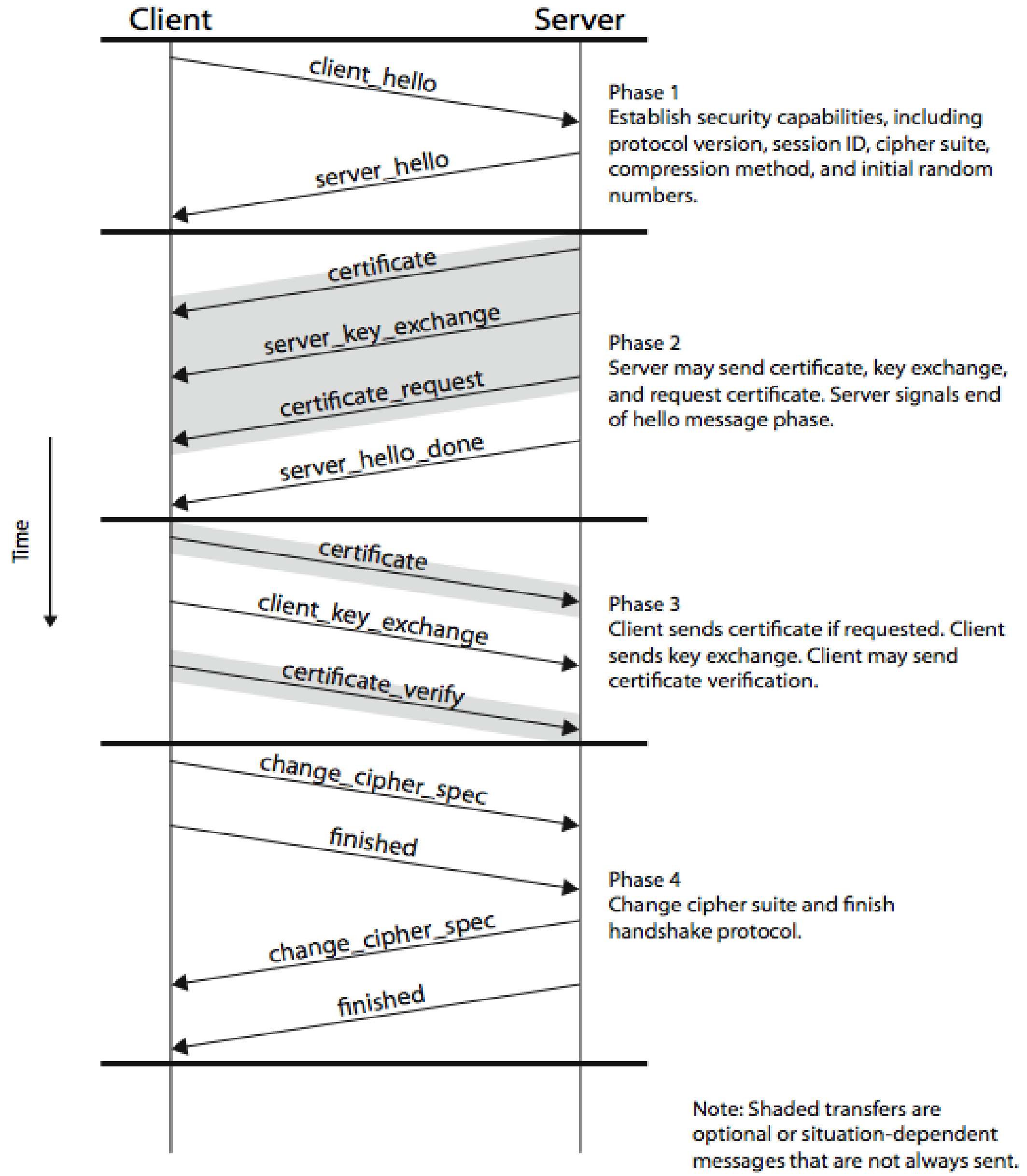
- Allows server and client to:
  - Authenticate each other
  - To negotiate encryption & MAC algorithms
  - To negotiate cryptographic keys to be used to protect data sent in an SSL record.

SSL/TLS Handshake Protocol

Comprises a series of messages in phases

1. Establish Security Capabilities:

   (a) The client initiates a logical connection and establish the security capabilities: protocol version, session ID, cipher suite (cryptographic algorithms supported by the client), compression method.

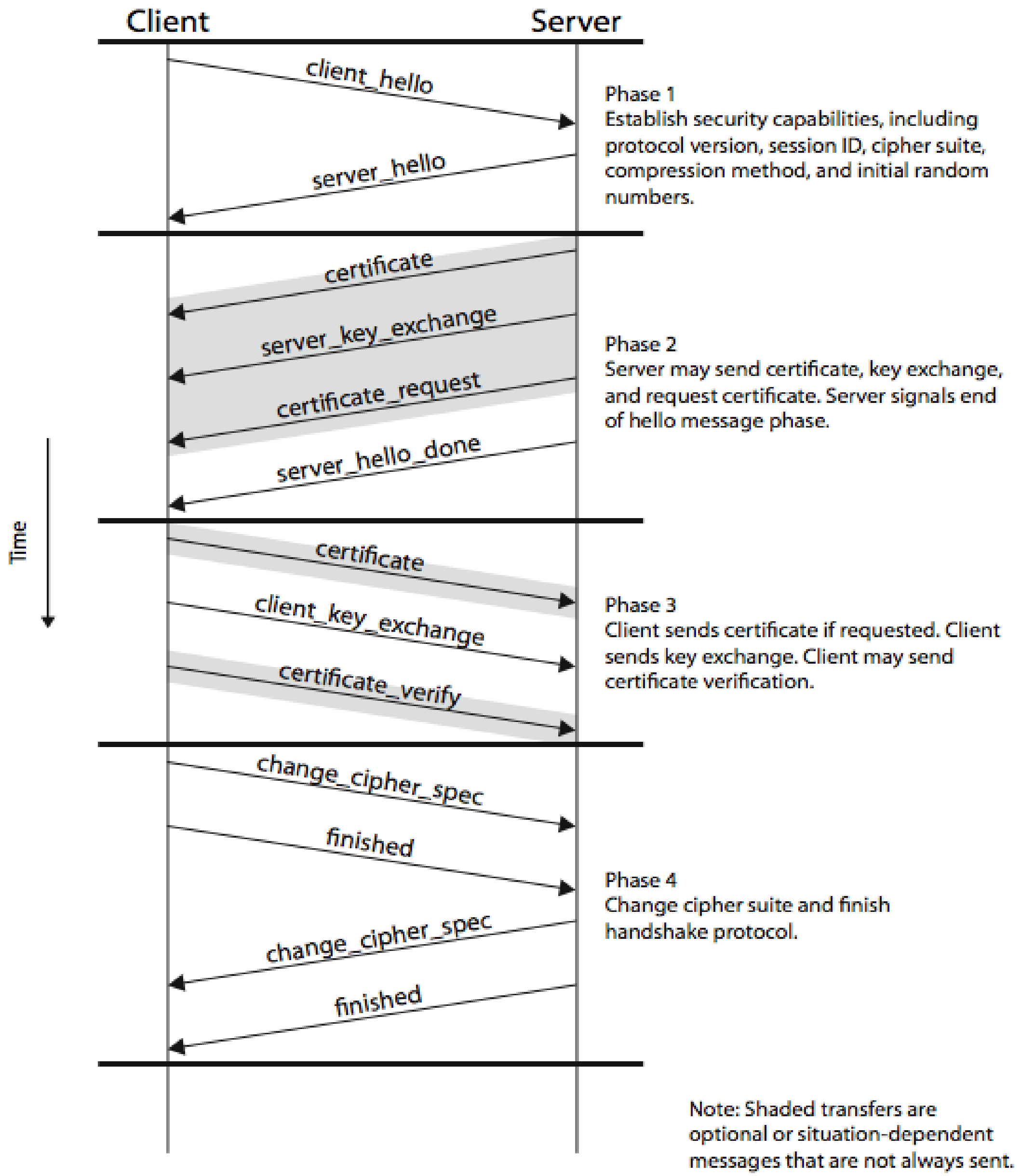


SSL/TLS Handshake Protocol

Comprises a series of messages in phases

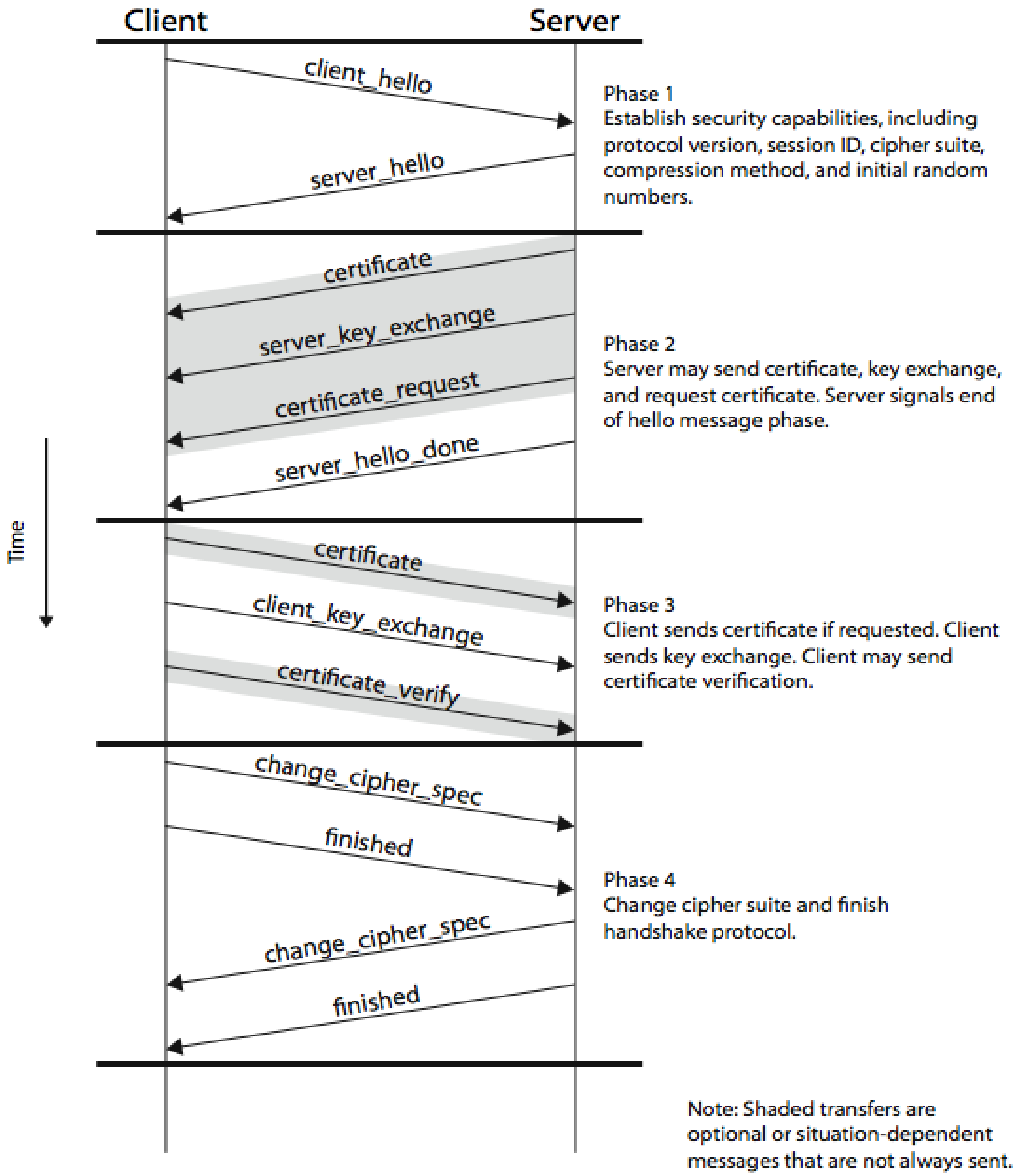
1. Establish Security Capabilities:

   (b) the server picks the strongest cipher and hash function that it also supports and notifies the client of the decision



SSL/TLS Handshake Protocol

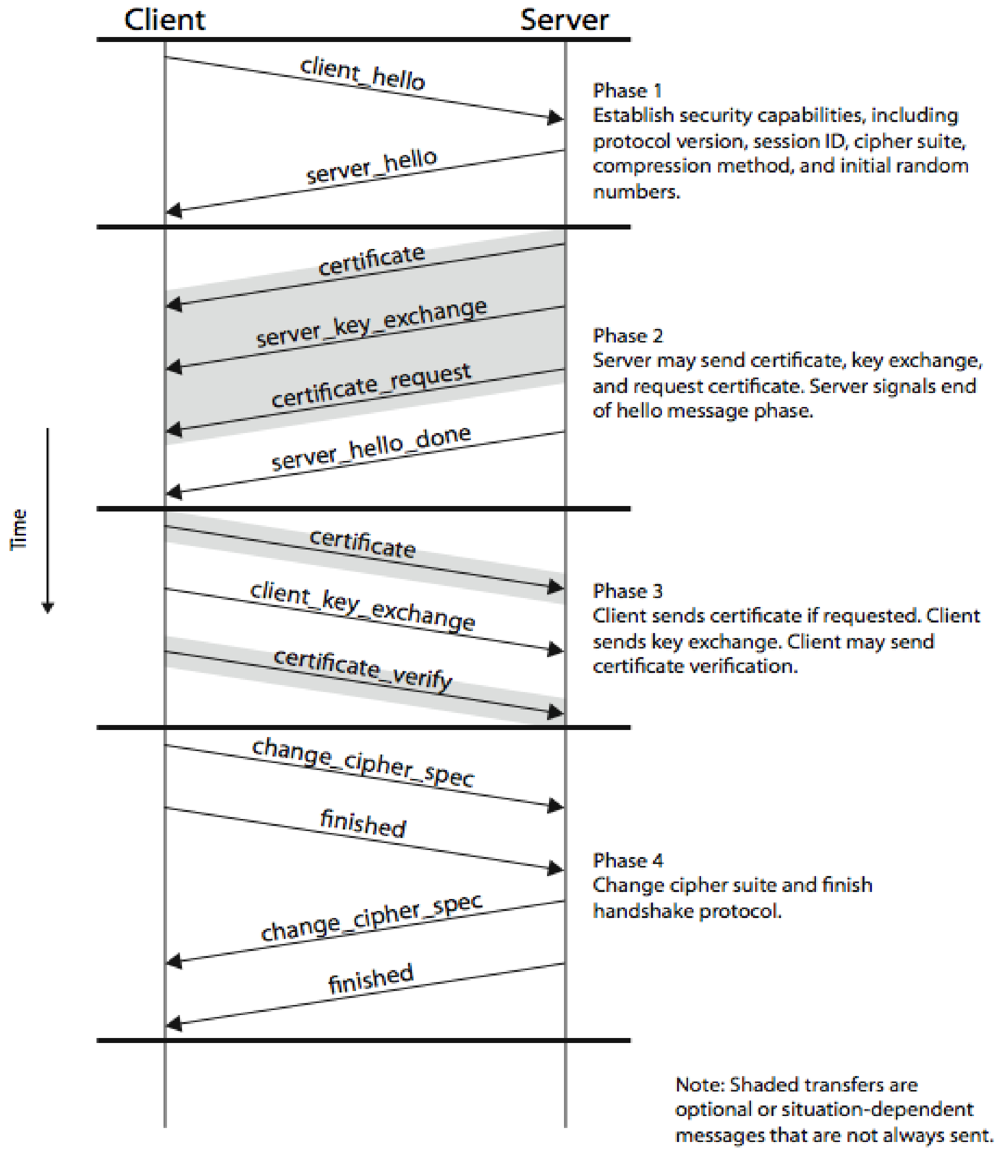
2. Server Authentication and Key Exchange: 1) sends certificate if it needs to be authenticated; 2) sends a server_key_exchange message, and request certificate; 3) signals the end of hello message phase.



SSL/TLS Handshake Protocol

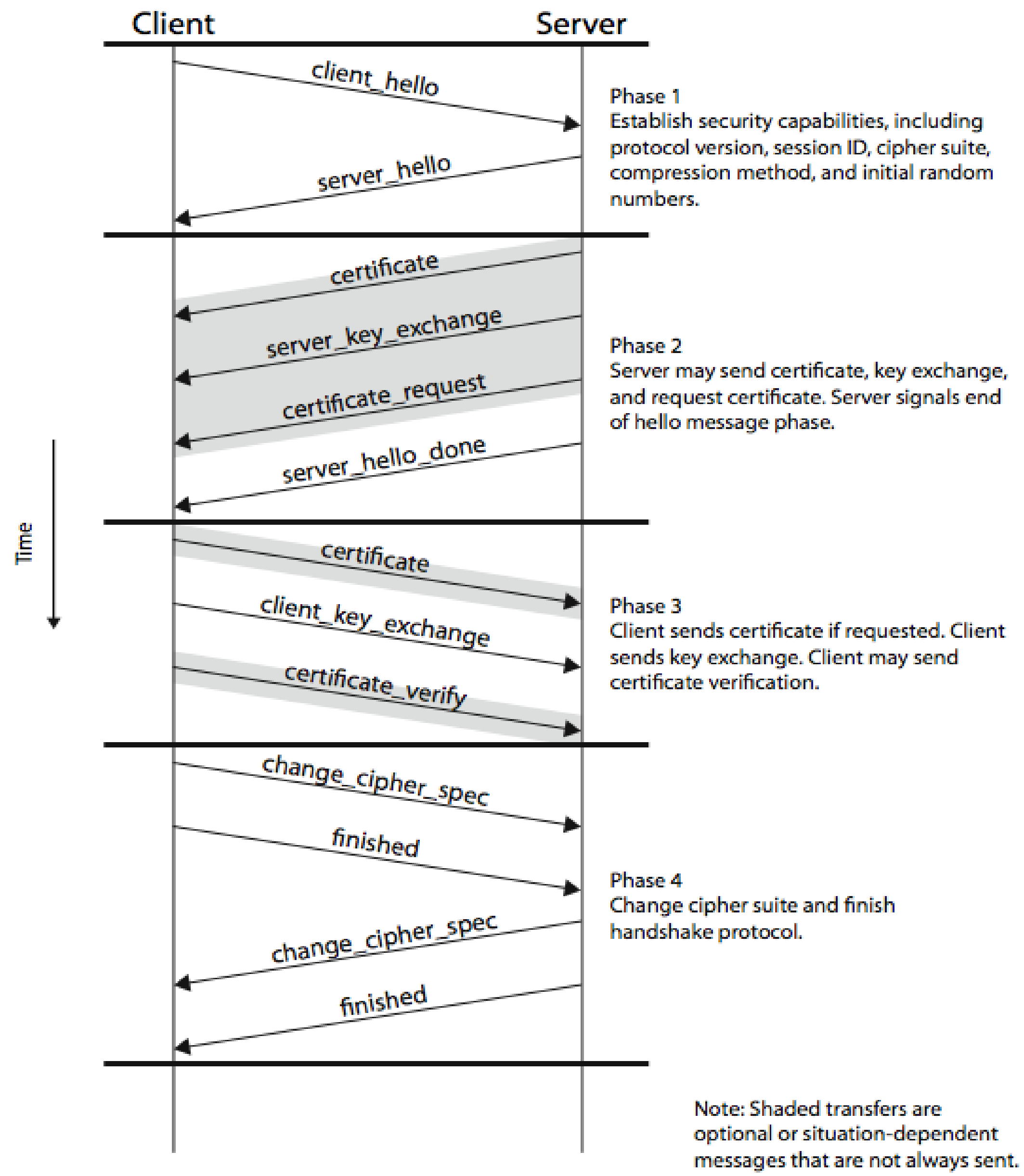
3. Client Authentication and Key Exchange:

sends certificate if requested and encrypts a random key with the server's public key, and sends the result to the server.



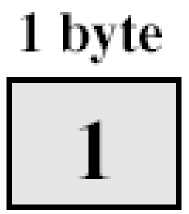
SSL/TLS Handshake Protocol

4. Change cipher suite and finish handshake protocol.



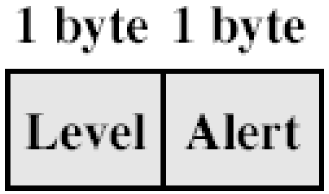
SSL/TLS Change Cipher Spec Protocol

- Notify the receiving party that subsequent records will be protected under the just-negotiated cipherspec and keys.
- Consists of a single message, which consists of a single byte with the value 1.
- Causes pending state to become current - updating the cipher suite in use



SSL/TLS Alert Protocol

- Conveys SSL-related alerts to peer entity
- Consists of two bytes – the first takes the value: warning (1) or fatal (2); the second contains a code that indicates the specific alert.



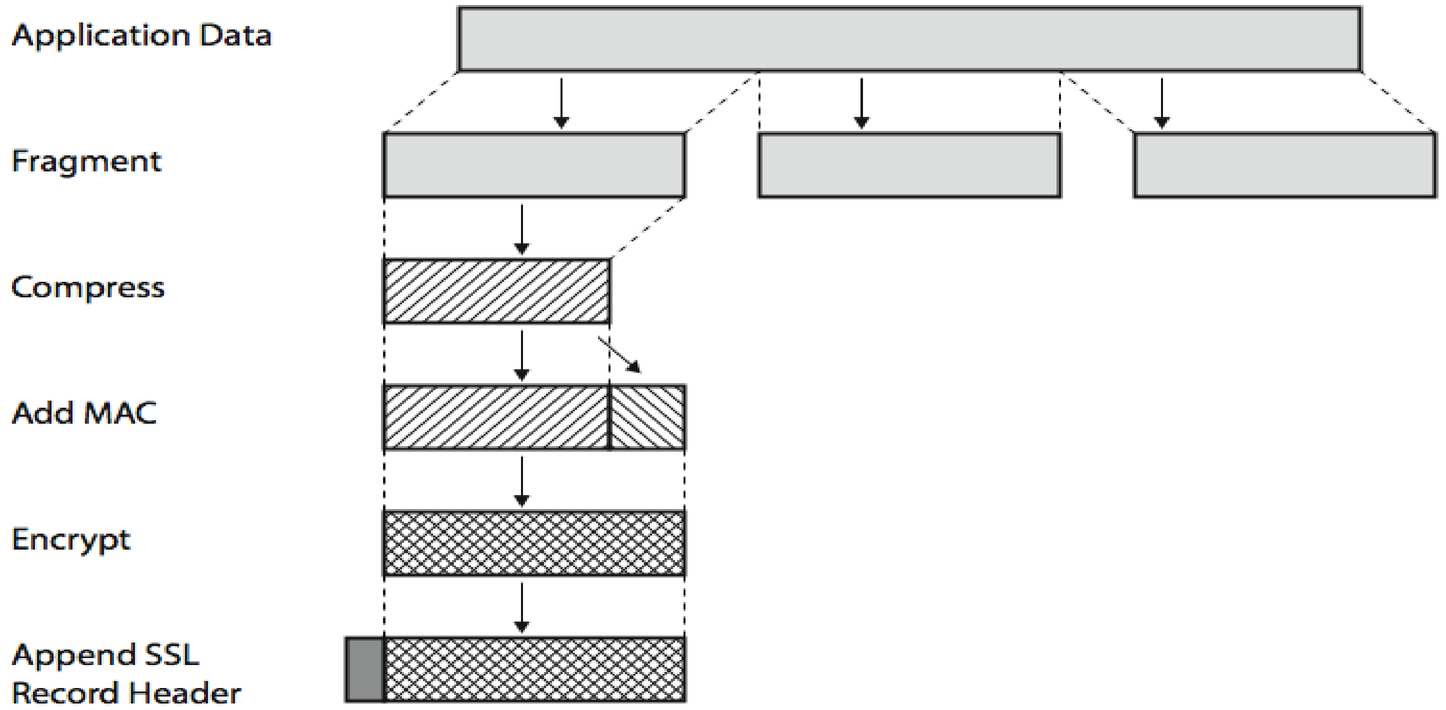
  - Fatal: unexpected message, decompression failure, handshake failure, illegal parameter
    - SSL immediately terminates the connection
  - Warning: close notify (the sender will not send any more message of this connection), bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown.

SSL/TLS Record Protocol Services

- Provides two services for SSL connections
  - Confidentiality:
    - encrypt SSL payloads
  - Message integrity:
    - use a shared secret key to form MAC.

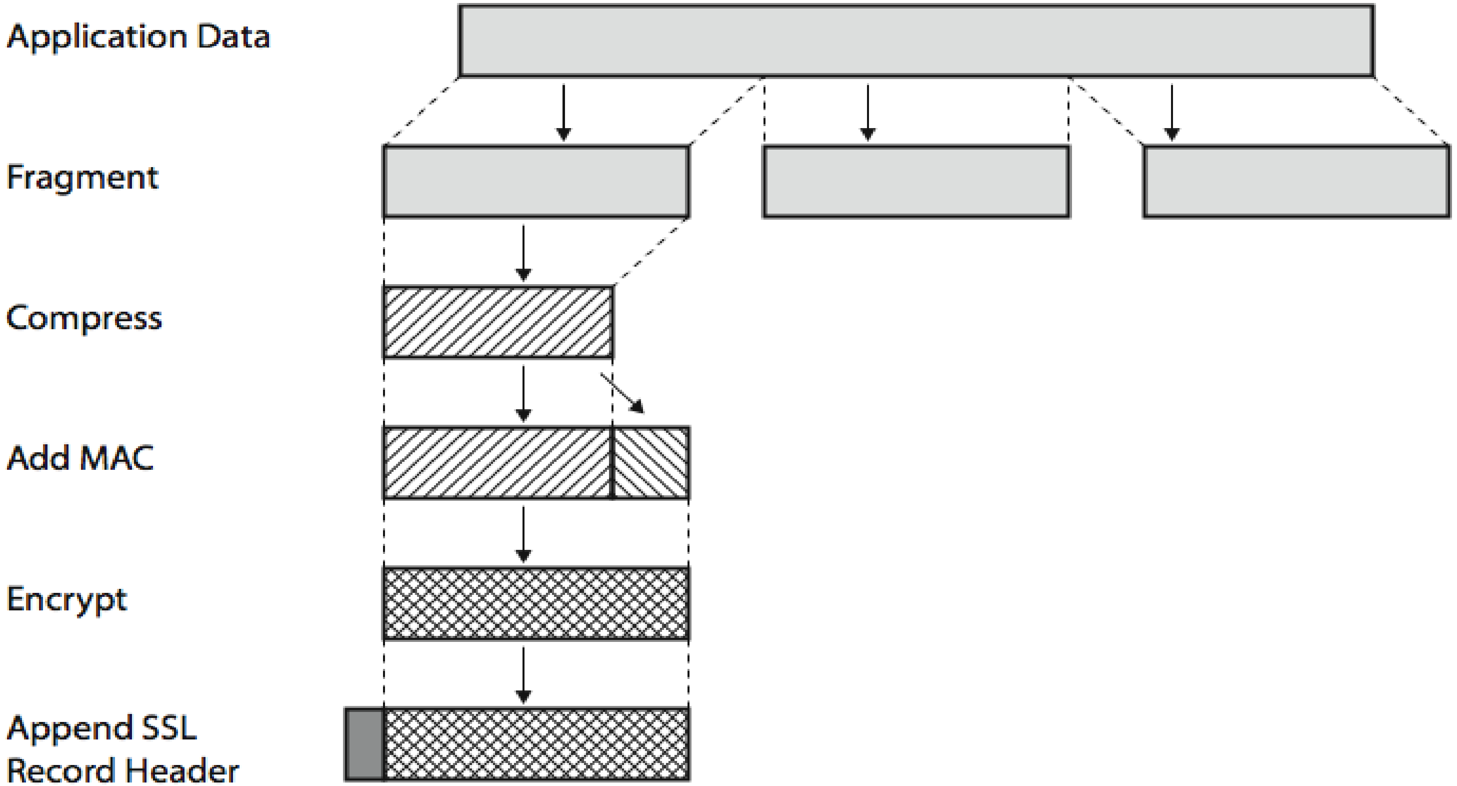
SSL/TLS Record Protocol Operation

- Fragmentation: message is fragmented into blocks of 2^{14} bytes or less
- Compression (optional): lossless and may not increase the content length by more than 1024 byte (for very short block, it is possible that the output is longer)



SSL/TLS Record Protocol Operation

- MAC: Compute the message authentication code over the compressed data.
- Encryption: the compressed message plus the MAC are encrypted using symmetric encryption.



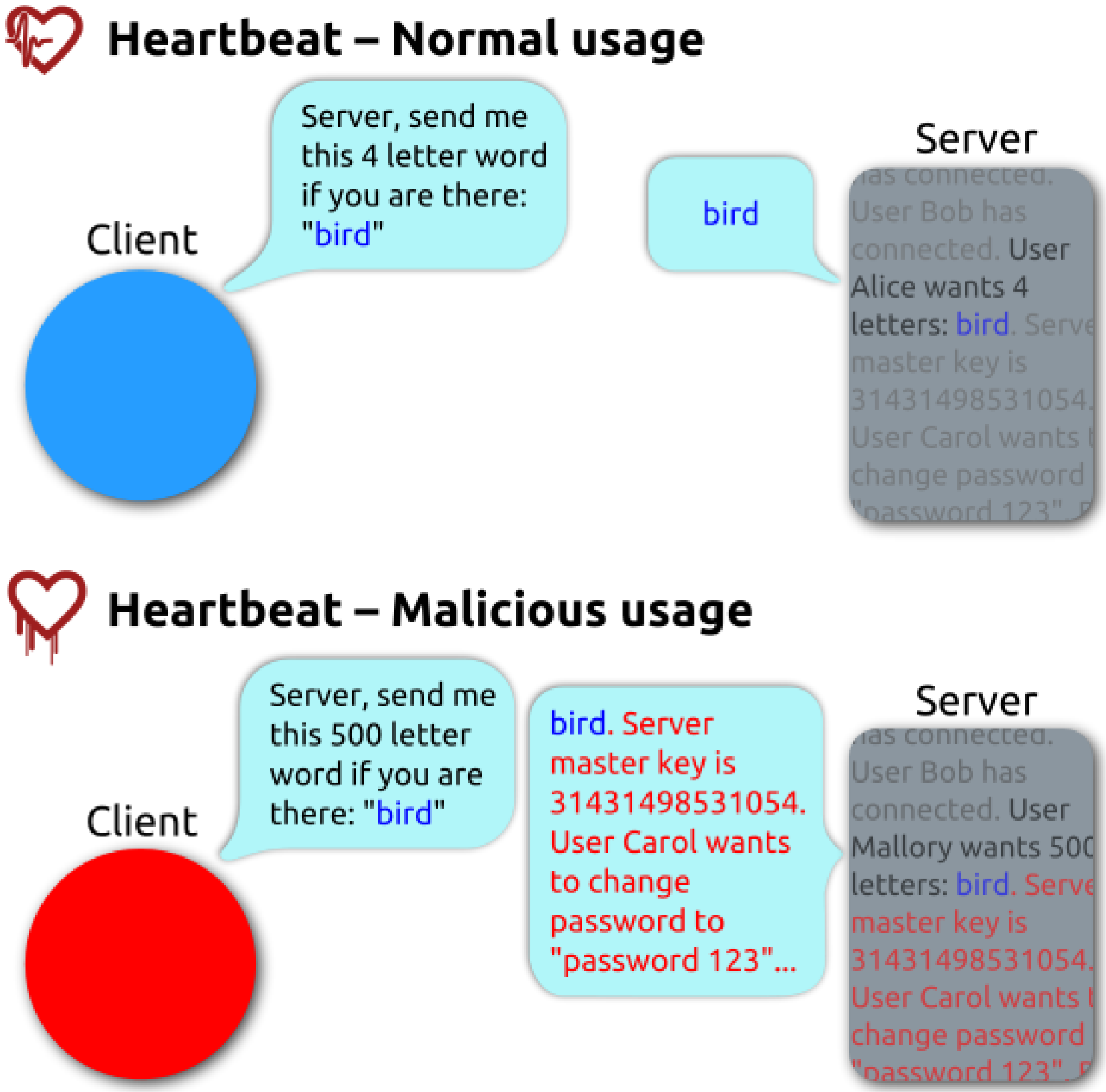
HeartBleed Vulnerability

- Introduced into OpenSSL code in 2011 by Robin Seggelmann (a Ph.D. student at the University of Duisburg-Essen).
  - Seggleman implemented a “heartbeat” function into OpenSSL which allows one side to check if the other side is still up and running.
- Stephen Henson, in charge of OpenSSL core development, did not spot Seggelmann’s bug.
  - Result: the vulnerable code was introduced into the production version…persisted till 2014.

HeartBleed Vulnerability

- SSL heartbeats are used for one side (server or client) to check if the other side is alive and well.
  - Send an N byte message to the other side. The other side will echo the same N bytes back to the sender.
- The implementation bug:
  - The sender sends an X < N byte message, but tells the receiver that the message is actually N bytes.
  - The other side will echo the X bytes and N-X bytes in the memory adjacent to the first X bytes.
    - That memory can contain keys, certificates, passwords, and other information.
    - Can steal up to 64 KB of data per heartbeat message.

HeartBleed Vulnerability Example:



Password Management

Password Management

- Front-line defense against intruders
- Users supply both:
  - login – determines whether the user is authorized to gain access to a system, and the privileges of that user.
  - password – to identify them
- Should protect password file on system
  - One-way function: the system stores only the value of a function based on the user's password.
  - Access control: access to the password file is limited to one or a very few accounts.

Unix Password Management

- The user selects a password.
- Multiple encryption/hashing schemes supported for storing the password
- Good explanation: http://en.wikipedia.org/wiki/Shadow_password

Linux Password and Verification: Basic Idea

- A system should *never* store passwords in plaintext
  - If access controls fail, the passwords are divulged

- Better idea: we could store passwords in an encrypted database
  - Problem: if the attacker compromises the key, they will be able to obtain all the passwords

- Best idea: store the hashes of passwords!
  - Hash functions are one-way, so even if the attacker compromises the password database, they will not be able to directly obtain the passwords

Linux Password and Verification: Basic Idea

- Hashed passwords: basic idea:
  - When a user account is created, the users original password is hashed and stored in the password database that maps user IDs to password hashes

Linux Password and Verification: Basic Idea

- Example: user Bob creates a password:
  - Types password
  - The password is hashed and is stored in the database

Password Database

User ID	Password Hash
Bob	77d7f045034e630f2625a8f875abf801 679f78391bb58518a1eba3bbf9b92d72
Alice	...

SHA-256(m23dd_sz%45) m23dd_sz%45

Linux Password and Verification: Basic Idea

- Example: user Bob **tries to log in** a password:
  - Types username/password
  - The password is hashed
  - The hash of the entered password is matched against Bob's hash in the database

Password Database

User ID	Password Hash
Bob	77d7f045034e630f2625a8f875abf801679f78391bb58518a1eba3bbf9b92d72
Alice	…

Compare

77d7f450
34e630f26
25a8f875a
bf801679f
78391bb58
518a1eba3b
bf9b92d72

Username: Bob
Password: m23dd_sz%45

SHA-256(m23dd_sz%45)

Linux Password and Verification: Basic Idea

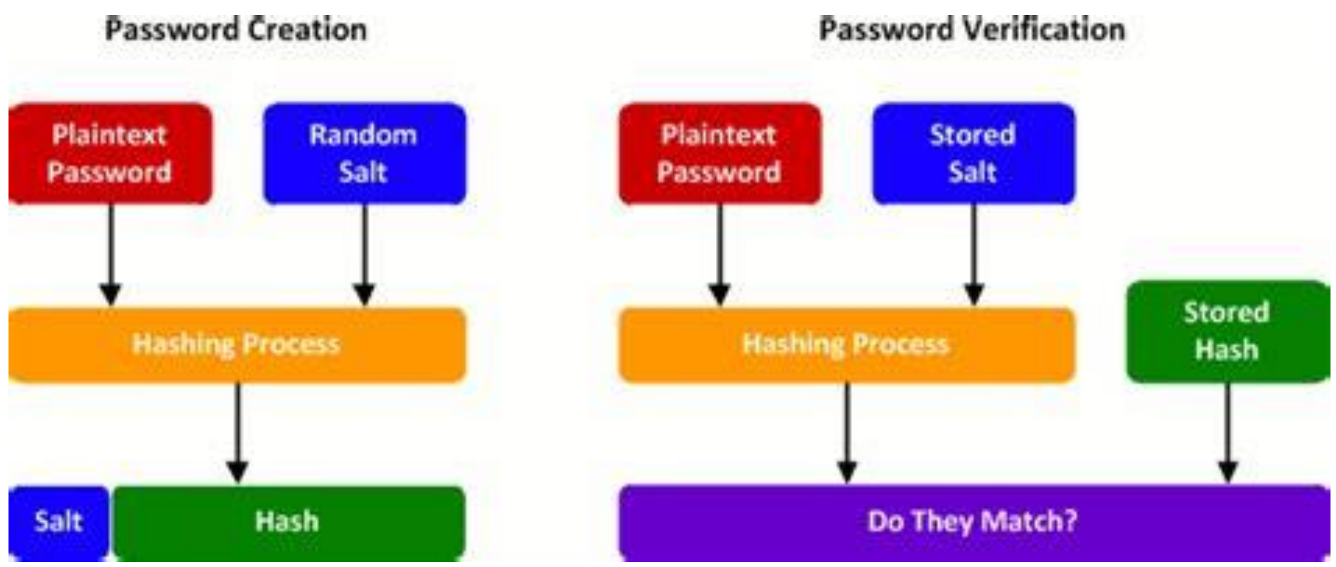
- Problem: since users tend to choose weak, predictable passwords, the attacker:
  - 1. Can create a dictionary of hashes of common passwords (a.k.a rainbow table)
    - E.g., a two-column mapping a password to a hash
  - 2. Steal the password database and match the hashes in the database against the hashes in the dictionary
  - 3. If a match is found, the attacker now knows the password
    - NOTE: if two users have the same hash, the attacker knows they have the same password

- Countermeasure? Next slide…

Linux Password and Verification: Basic Idea

- Solution: password salting: is a standard technique used to improve security of hashed password storage
  - A random value known as salt, is added to the password prior to hashing it
  - A column is added to the password database that contains the salt

Linux Password and Verification: Basic Idea



- Salting: Basic idea:
  - Storing the password: the user provides the password
    - Before hashing the password, a Salt value (a random number) is added to the password to help frustrate rainbow table attacks.
    - The password and salt are stored in the password database indexed by the user name
  - Verification: the user enters user name and password. The entry for the user is looked up in the database. If the associated hash(entered password || salt) == hash in the database, the password is correct

Managing Passwords - Education

- Benefits of salting:
  - If the attacker has only the salted hash:
    - 10 alphanumeric character password has 26^{10} possible hashes
    - 10 alphanumeric password + 12-bit salt has $26^{10} * 2^{12}$ possible hashes
  - If the attacker has the salted hash and the salt, two users with the same passwords will different hashes.

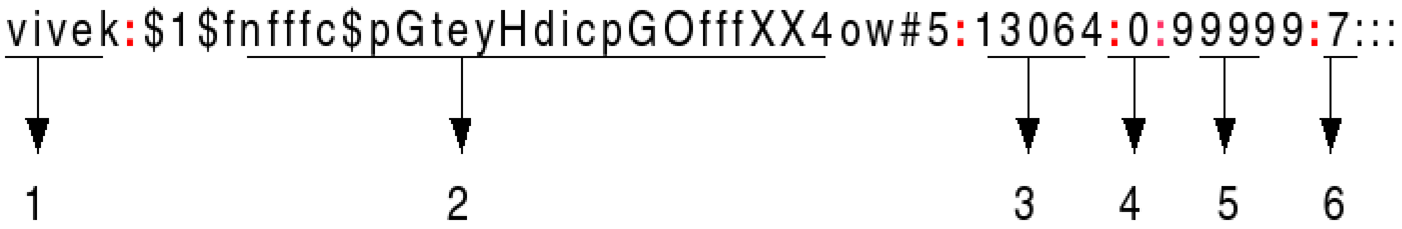
Linux Password and Verification: Basic Idea

- **Real-world:** In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- **Example:** each row represents an entry for a user:

```
_apt:*:18474:0:99999:7:::
tss:*:18474:0:99999:7:::
uuidd:*:18474:0:99999:7:::
tcpdump:*:18474:0:99999:7:::
avahi-autoipd:*:18474:0:99999:7:::
usbmux:*:18474:0:99999:7:::
rtkit:*:18474:0:99999:7:::
dnsmasq:*:18474:0:99999:7:::
cups-pk-helper:*:18474:0:99999:7:::
speech-dispatcher:!:18474:0:99999:7:::
avahi:*:18474:0:99999:7:::
kernoops:*:18474:0:99999:7:::
saned:*:18474:0:99999:7:::
nm-openvpn:*:18474:0:99999:7:::
hplip:*:18474:0:99999:7:::
whoopsie:*:18474:0:99999:7:::
colord:*:18474:0:99999:7:::
geoclue:*:18474:0:99999:7:::
pulse:*:18474:0:99999:7:::
gnome-initial-setup:*:18474:0:99999:7:::
gdm:*:18474:0:99999:7:::
student:$6$yJZ4XU5DkD/nllOZ$qpxaNRigM4SS4Odrt3vapZhR/7cj/WhZ2Q8YnJefdVMqrkdTSXxLrbWVVBNtixXPzL5aZ7d5nBSyrTJA/Ul27/:18488:0:99999:7:::
systemd-coredump:!!:18488::::::
sshd:*:18490:0:99999:7:::
_rpc:*:18490:0:99999:7:::
statd:*:18490:0:99999:7:::
vboxadd:!:18490::::::
pike:$6$eQDxGDHbWpTp8cb0$Li2ZPwdEJQ3xDggKUu5YZNYB0HZePqUwX.0sMmhy7f/Zu/14PmHjSjuiDZKVzZ/V.x9tB5dbKfXndo9efi/Jh.:19253:0:99999:7:::
jacob:$6$P87xg9ZdVLDfQv8W$Kfh1b0ViZ4y9pzabEgaZrVfq5dZuV7z9.TA.DvsnFChjt82i3Pa40VQDNbSomLhjBwB3IUrarz4f1Z4rUu.9v/:19254:0:99999:7:::
```

Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry (image source: https://www.cyberciti.biz/faq/understanding-etcshadow-file/):



- 1. **User name:** the user name to whom this password belongs

Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:

- 2. **Hashed password.** The format of the password is idsalt$hashed where:
  - Id is the hashing algorithm
    - 1 is MD5 (the one used in the example)
    - $2a$ is Blowfish
    - $2y$ is Blowfish
    - 5 is SHA-512
    - 6 is SHA-512

Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:

- 2. **Hashed password.** The format of the password is idsalt$hashed where:
  - $salt$ is the salt value (fnfffc in the example)

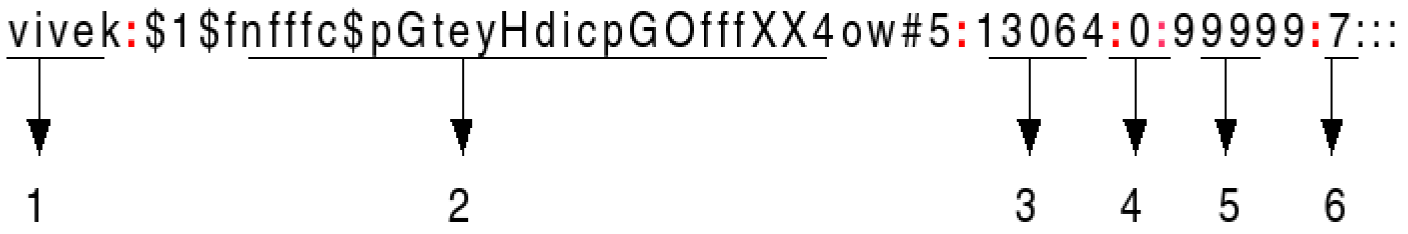
Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:

- 2. **Hashed password.** The format of the password is idsalt$hashed where:
  - $hashed$ is the hash value of the combined salt and hash (pGteyHdicpGOfffXX4ow#5 in the example)

Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:



- 3. **Last password change:** The date of the last password change as number of days elapsed since Jan 1, 1970 (Unix time). A value of 0 means the user must change the password at the next login.

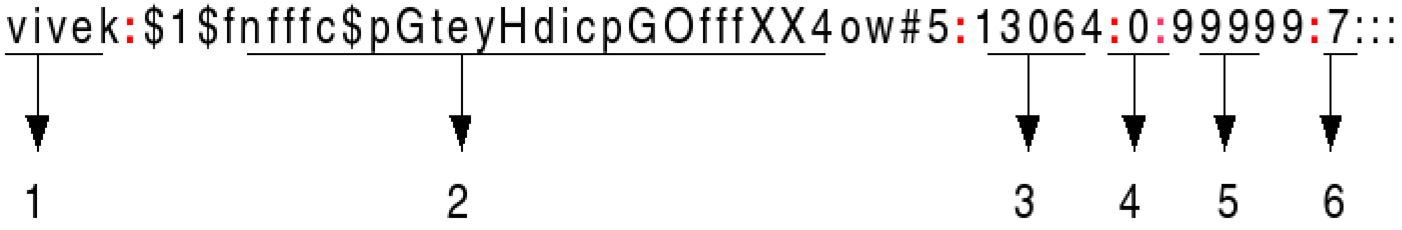
Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:

- 4. **Minimum:** The minimum number of days required between password changes (the number of days left before the user must change their password)

Linux Password and Verification: Basic Idea

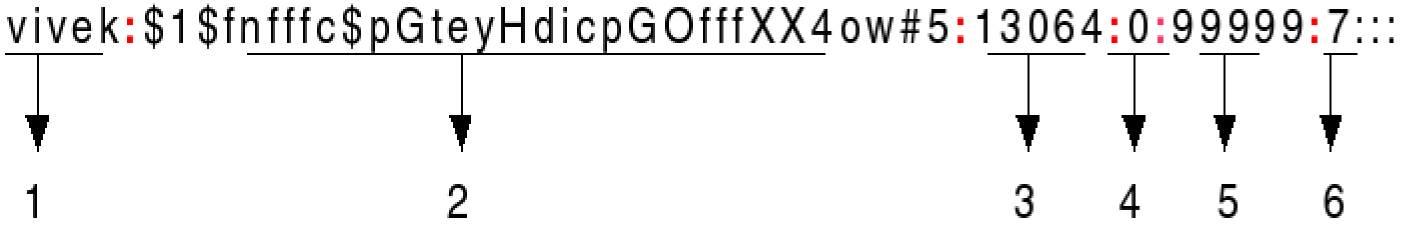
- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:



- 5. **Maximum:** The number of days the password is valid before the user must change it

Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:



- 6. **Warn:** The number of days before the password is to expire that the user is warned about the need to change the password.

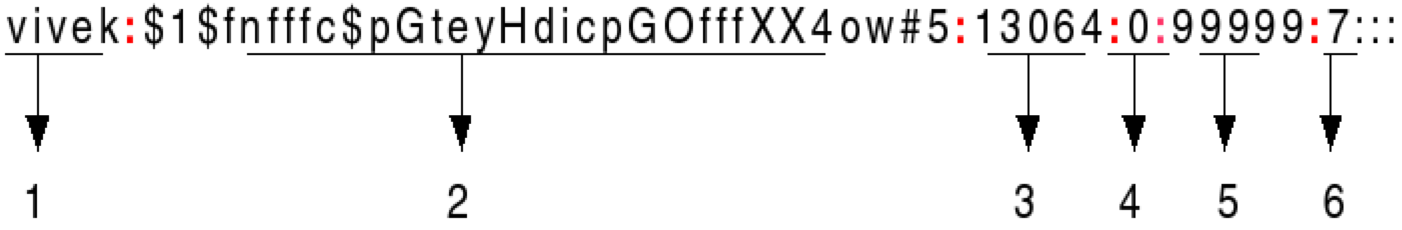
Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:

- 7. **Inactive:** The number of days after password expiration after which the account is disabled.

Linux Password and Verification: Basic Idea

- Real-world: In Linux the passwords and salts are stored in the root-only accessible /etc/shadow file
- Example: Meaning of an entry:



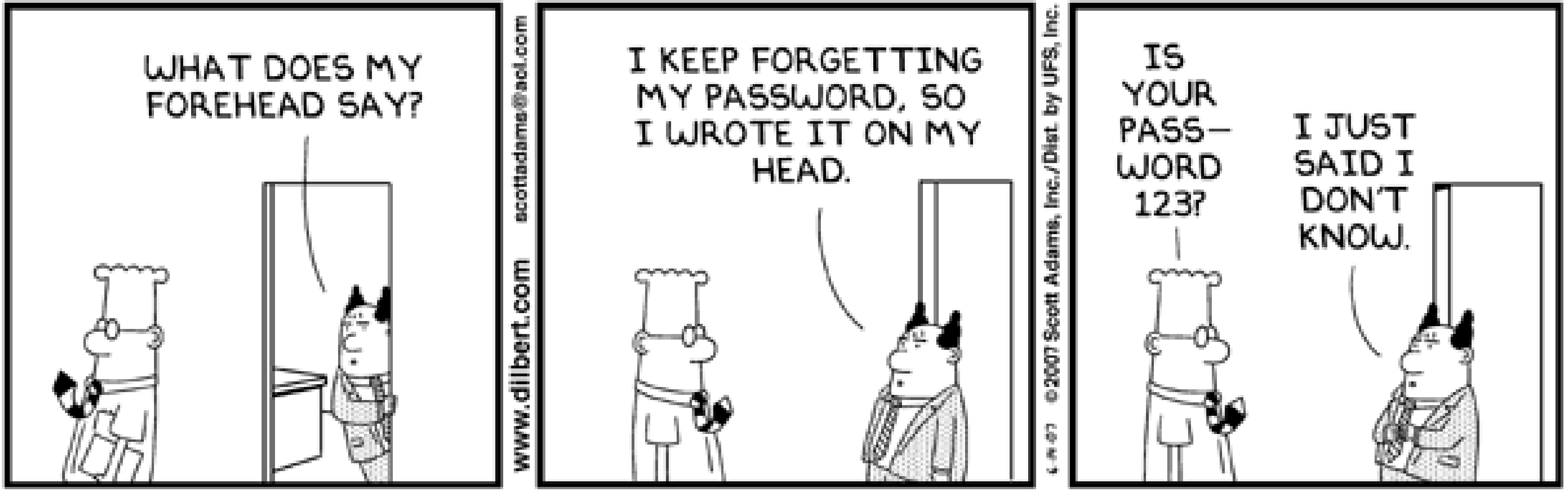
- 8. **Expire**: The date of the account expiration (as days since January 1, 1970)

Managing Passwords - Computer Generated

- Let computer create passwords

Managing Passwords - Computer Generated

- Let computer create passwords
  - If the passwords are quite random in nature, users will not be able to remember them.
  - Even if the password is pronounceable, the user may have difficulty remembering it.
  - Have history of poor user acceptance.



Managing Passwords - Reactive Checking

- Periodically run password guessing tools
- Cracked passwords are disabled
- But is resource intensive
- Bad passwords are vulnerable till found

Managing Passwords - Proactive Checking

- Most promising approach to improving password security
- Allow users to select own password
- But have system verify it is acceptable
  - Simple rule enforcement
  - Compare against dictionary of bad passwords

Acknowledgements

- Some slides borrowed from Dr. Ping Yang from State University of New York at Binghamton.