Security Handshake Pitfalls
Security Handshake Pitfalls

Client (K) → Server (K)

Hello
Security Handshake Pitfalls

Client (K) → Challenge R → Server (K)
Security Handshake Pitfalls

Client (K) \rightarrow f(K, R) \rightarrow Server (K)
**Security Handshake Pitfalls**

**Problems:**
1. Authentication is not mutual - only Server authenticates Client. Anyone can send the challenge $R$. 

Security Handshake Pitfalls

Problems:

1. Authentication is not mutual - only Server authenticates Client
   Anyone can send the challenge $R$.
2. Connection can be hijacked – if the rest of the transaction is without
   cryptographic protection and attacker makes packets with Client addr
Security Handshake Pitfalls

Problems:
1. Authentication is not mutual - only Server authenticates Client.
   Anyone can send the challenge $R$.
2. Connection can be hijacked – if the rest of the transaction is without cryptographic protection and attacker makes packets with Client addr.
3. Off-line password guessing attack - if $K$ is derived from password because both $R$ and $f(K,R)$ are observable.
Security Handshake Pitfalls

Problems:
1. Authentication is not mutual - only Server authenticates Client
   Anyone can send the challenge $R$.
2. Connection can be hijacked – if the rest of the transaction is without cryptographic protection and attacker makes packets with Client addr
3. Off-line password guessing attack - if $K$ is derived from password because both $R$ and $f(K,R)$ are observable.
4. Someone reading the Server database may be able to impersonate Client later or to another server that the client uses
Security Handshake Pitfalls

Client (K) \rightarrow \text{Hello} \rightarrow \text{Server (K)}
Security Handshake Pitfalls
Security Handshake Pitfalls

Client
(K)

Server
(K)

$R$
Security Handshake Pitfalls

Concerns:
1. Requires reversible cryptography - no hash
Security Handshake Pitfalls

Concerns:
1. Requires reversible cryptography - no hash
2. No need to eavesdrop if $R$ is recognizable - Attacker says hello and gets back $K\{R\}$, if $K$ derived from password can generate lots of $K$s for a dictionary attack.
Security Handshake Pitfalls

Concerns:
1. Requires reversible cryptography - no hash
2. No need to eavesdrop if $R$ is recognizable - Attacker says hello and gets back $K\{R\}$, if $K$ derived from password can generate lots of Ks for a dictionary attack.
3. If $R$ is recognizable by Client, then protocol authenticates Server but only if life of $R$ is short, otherwise $K\{R\}$ can be replayed by attacker
Security Handshake Pitfalls

Client (K) → Server (K) → Hello, K{time}
Security Handshake Pitfalls

Advantage:
1. One direction, more efficient - is drop-in replacement for existing prot.
Security Handshake Pitfalls

**Advantage:**
1. One direction, more efficient - is drop-in replacement for cleartext prot.
2. The server does not need to keep volatile state (such as R)
Security Handshake Pitfalls

Client (K)  

Hello, K{time}  

Server (K)

**Advantage:**
1. One direction, more efficient - is drop-in replacement for cleartext prot.
2. The server does not need to keep volatile state (such as R)

**Problems:**
1. Attacker can use K{time} to impersonate Client - within acc clock skew
**Security Handshake Pitfalls**

---

**Advantage:**
1. One direction, more efficient - is drop-in replacement for cleartext prot.
2. The server does not need to keep volatile state (such as R)

**Problems:**
1. Attacker can use K{time} to impersonate Client - within acc clock skew
   But can be foiled if Server remembers all timestamps sent.
Security Handshake Pitfalls

**Advantage:**
1. One direction, more efficient - is drop-in replacement for cleartext prot.
2. The server does not need to keep volatile state (such as R)

**Problems:**
1. Attacker can use K\{time\} to impersonate Client - within acc clock skew
   But can be foiled if Server remembers all timestamps sent.
2. If multiple Servers, same K, K\{time\} on one can work on others
Security Handshake Pitfalls

**Advantage:**
1. One direction, more efficient - is drop-in replacement for cleartext prot.
2. The server does not need to keep volatile state (such as R)

**Problems:**
1. Attacker can use K{time} to impersonate Client - within acc clock skew
   But can be foiled if Server remembers all timestamps sent.
2. If multiple Servers, same K, K{time} on one can work on others
   Can be foiled by sending K{time | server}
Security Handshake Pitfalls

Advantage:
1. One direction, more efficient - is drop-in replacement for cleartext prot.
2. The server does not need to keep volatile state (such as R)

Problems:
1. Attacker can use K\{time\} to impersonate Client - within acc clock skew
   But can be foiled if Server remembers all timestamps sent.
2. If multiple Servers, same K, K\{time\} on one can work on others
   Can be foiled by sending K\{time | server\}
3. Vulnerable to attacker setting time back on Server!!
Security Handshake Pitfalls

4. Setting time requires a security handshake - better make it chllng/resp (that is, not depending on time) or else there is a problem if clocks of machines are too far apart – the negotiation will fail.
Security Handshake Pitfalls

Client (K) → Server (K)
Hello, time, hash\{K, time\}
Security Handshake Pitfalls

Client (K) → Server (K)

Hello, time, hash{K,time}

Advantage:
1. Otherwise, if time not sent, for Server to check hash, must try lots of different acceptable times before one matches the hash
Security Handshake Pitfalls

Client
\((K_c, S_c)\)

Hello

Server
\((K_s, S_s)\)
Security Handshake Pitfalls

Client $(K_c, S_c)$

Server $(K_s, S_s)$

Challenge $R$
Security Handshake Pitfalls

Advantage:
Like secret-key protocol except Server database not vulnerable to peeking
Security Handshake Pitfalls

**Advantage:**
Like secret-key protocol except Server database not vulnerable to peeking

**Problem:**
Can trick someone into signing something. Wait for Client to say Hello, send back something you want client to sign, get the signed item - use it.
Security Handshake Pitfalls

Client (K_c, S_c) → Hello → Server (K_s, S_s)
Security Handshake Pitfalls

Client
$(K_c, S_c)$

Server
$(K_s, S_s)$

$\text{encrypt} \{ R \}_{K_c}$
Security Handshake Pitfalls

Concerns:
1. Some public key systems can only do signatures, not reversible encryption
Security Handshake Pitfalls

Concerns:
1. Some public key systems can only do signatures, not reversible encryption
2. Attacker can get an encrypted message sent to client, impersonate Server, send the message to Client who decrypts it. Gets back the decrypted message.
Should the same keys be used for encryption and signing?

0. Someone encrypts message $M$ with client public key, sends to client.
1. Attacker intercepts $M^e \mod N$ where $<e,N>$ is the client's key pair.
2. Attacker chooses number $R$, prime relative to modulus $N$.
3. Attacker creates bogus message $H = M^e \cdot R^e$ (recall $e$ is public).
4. Attacker impersonates server, sends $H$ to client for signing.
5. Attacker gets $H^d \mod N = (M^e \cdot R)^d \mod N = M^e \cdot R \mod N$.
6. Attacker multiplies by $R^{-1}$ to get $M$. 
Security Handshake Pitfalls

Concerns:

1. Some public key systems can only do signatures, not reversible encryption
2. Attacker can get an encrypted message sent to client, impersonate Server, send the message to Client who decrypts it. Gets back the decrypted message.

Never use the same key for signing and encryption/decryption
Security Handshake Pitfalls

Concerns:
1. Some public key systems can only do signatures, not reversible encryption
2. Attacker can get an encrypted message sent to client, impersonate Server, send the message to Client who decrypts it. Gets back the decrypted message.

Never use the same key for signing and encryption/decryption
Observe: several secure independent systems may be insecure when put together (attacker may use one protocol to break another).
Security Handshake Pitfalls

Mutual Authentication:
Security Handshake Pitfalls

Mutual Authentication:

Client
(K)

Server
(K)

\[ R_s, f(K, R_c) \]
Security Handshake Pitfalls

Mutual Authentication:

Client
(K)

\( f(K, R_s) \)

Server
(K)
Reflection Attack:
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Security Handshake Pitfalls

**Reflection Attack:**
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Attacker gets back $f(K, R_i)$ (who cares) and $R_s$. 
**Reflection Attack:**
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Attacker gets back $f(K, R_i)$ (who cares) and $R_s$.
Attacker opens second connection to Server using $R_s$. 
**Reflection Attack:**
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Attacker gets back $f(K, R_i)$ (who cares) and $R_s$.
Attacker opens second connection to Server using $R_s$.
Attacker gets back $R_s$ encrypted.
Reflection Attack:
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Attacker gets back $f(K, R_i)$ (who cares) and $R_s$.
Attacker opens second connection to Server using $R_s$.
Attacker gets back $R_s$ encrypted.
Attacker abandons second connection and sends back $f(K, R_s)$ to complete.
Reflection Attack, how to foil it:

1. Instead of the same key $K$, use different keys $K_s$, $K_c$
   - $K_s$ may be almost $K_c$ (say times -1 or 1 different etc.)
2. Insist the challenge from the Client is different from that of the Server

Security Handshake Pitfalls
Reflection Attack, how to foil it:
1. Instead of the same key $K$, use different keys $K_s$, $K_c$
   $K_s$ may be almost $K_c$ (say times -1 or 1 different etc.)
2. Insist the challenge from the Client is different from that of the Server

Also, off-line password guessing attack vulnerability: Attacker sends message to Server claiming to be Client and enclosing a challenge $R$, Server returns the challenge encrypted $f(K, R)$, this gives the attacker ability to check passwords
Reflection Attack can't succeed with this protocol:
Reflection Attack can't succeed with this protocol:
Reflection Attack can't succeed with this protocol:
Reflection Attack can't succeed with this protocol:
Reflection Attack can't succeed with this protocol:

Reason: Initiator always is first to prove its identity!!
Security Handshake Pitfalls

Hello, encrypt\{R_c\}_{K_s}

Public key mutual authentication:
Security Handshake Pitfalls

Client
$(K_c, S_c)$

Server
$(K_s, S_s)$

Public key mutual authentication:

$R_c, \text{ encrypt}\{R_s\}_{K_c}$
Security Handshake Pitfalls

Public key mutual authentication:

Client
\[(K_c, S_c)\]

\[R_s\]

Server
\[(K_s, S_s)\]
Public key mutual authentication:

Variant: $R_s$ and $R_c$ are signed by their respective parties
Security Handshake Pitfalls

Public key mutual authentication:

Variant: $R_s$ and $R_c$ are signed by their respective parties

How does the client's workstation retrieve the client's private key when all the client knows is a password? – usually keys cannot be recovered from passwords, e.g. RSA

Use a directory service - client workstation retrieves the following:
1. Client's private key which is encrypted with Client's password
2. Certificate for Server's public key, signed with Client's private key
Establish a session key (secret key system):
After authentication $R$ and $K\{R\}$ is known by Client and Server
Can't use $K\{R\}$ as session key - someone may have seen it
Can't use $K\{R+1\}$ as session key - Attacker seeing transmission using session key $K\{R+1\}$ impersonates Server's network address to Client and sends $R+1$ as the challenge to Client. Client responds with $K\{R+1\}$. So attacker can decrypt previous transmission.
Establish a session key (secret key system):

After authentication $R$ and $K\{R\}$ is known by Client and Server
Can't use $K\{R\}$ as session key - someone may have seen it
Can't use $K\{R+1\}$ as session key - Attacker seeing transmission using session key $K\{R+1\}$ impersonates Server's network address to Client and sends $R+1$ as the challenge to Client. Client responds with $K\{R+1\}$. So attacker can decrypt previous transmission.

Session key should not be an encrypted value which can be predicted or extracted later.
Establish a session key (public key system):
Authenticate with Diffie-Hellman exchange, RSA signing.
Use $g^{ab} \mod p$ as the session key.
Attempt to get root access on either machine does not reveal the $a$ and $b$ needed to generate session key to decrypt recorded messages - they were never recorded and the session key was tossed. If session key had been sent across, it could have been recorded and decrypted after getting root access. If session keys get sent across without being signed, attacker can send an $R$ while impersonating Client's network address.
Security Handshake Pitfalls

KDC operation, in principle

- Client A: $K_a$ (K DC operation, in principle)
- KDC: $K_s$
- Client B: $K_b$

Request B:
- $K_a \{K_{AB}\}$
- $K_b \{K_{AB}\}$

Makes $K_{AB}$
Needham-Schroeder authentication and session key (secret key system):
Client A sends a nonce $N_a$ and request to connect to B to the KDC
Needham-Schroeder authentication and session key (secret key system):

Client A sends a nonce $N_a$ and request to connect to B to the KDC.
KDC makes a session key $K_{ab}$ for the connection.
Needham-Schroeder authentication and session key (secret key system):
Client A sends a nonce $N_a$ and request to connect to B to the KDC.
KDC makes a session key $K_{ab}$ for the connection.
KDC encrypts $N_a$, B, a ticket T with key $K_a$ and sends all to A.
To make T the KDC encrypts the session key and A with with $K_b$. 

Security Handshake Pitfalls
Needham-Schroeder authentication and session key (secret key system):

Client A sends a nonce $N_a$ and request to connect to B to the KDC.
KDC makes a session key $K_{ab}$ for the connection.
KDC encrypts $N_a$, B, a ticket T with key $K_a$ and sends all to A.
To make T the KDC encrypts the session key and A with with $K_b$.
Client A encrypts a nonce $N_b$ with the session key $K_{ab}$, sends with ticket to B.
Needham-Schroeder authentication and session key (secret key system):

Client A sends a nonce $N_a$ and request to connect to B to the KDC.
KDC makes a session key $K_{ab}$ for the connection.
KDC encrypts $N_a$, B, a ticket T with key $K_a$ and sends all to A.
To make T the KDC encrypts the session key and A with with $K_b$.
Client A encrypts a nonce $N_b$ with the session key $K_{ab}$, sends with ticket to B.
Client B subtracts 1 from nonce, make new nonce, encrypts with session key sends encrypted message to Client A.
Needham-Schroeder authentication and session key (secret key system):

Client A sends a nonce $N_a$ and request to connect to B to the KDC.
KDC makes a session key $K_{ab}$ for the connection.
KDC encrypts $N_a$, B, a ticket T with key $K_a$ and sends all to A.
To make T the KDC encrypts the session key and A with with $K_b$.
Client A encrypts a nonce $N_b$ with the session key $K_{ab}$, sends with ticket to B.
Client B subtracts 1 from nonce, make new nonce, encrypts with session key and sends encrypted message to Client A.
Client A subtracts 1 from nonce, encrypts with session key, sends to Client B.
**Security Handshake Pitfalls**

*Needham-Schroeder: What is going on?*

Nonce $N_a$ protects against: attacker, having stolen previous $K_b$ and message from Client A requesting it, waits for Client A to request a session key from the KDC and immediately plays back KDC's reply to Client A, and gets the session key encrypted with Client B's old $K_b$, which it knows, so it can impersonate Client B to Client A.

$$K_a \{N_a, B, K_{ab}, T\}$$

$$T=K_b \{K_{ab}, A\}$$

*Attacker need not know key of Client A*
Needham-Schroeder: What is going on?
With the identity of Client B inserted, attacker id substitution is detected by A.
Nonce is returned to check authenticity of KDC.
Double encryption of session key (K_b for ticket, then K_a on the ticket) is not considered necessary.
Needham-Schroeder: What is going on?
Client A knows that only someone with Client B's key $K_b$ can decrypt and get the session key. With session key, Client B decrypts $N_b$. 

Security Handshake Pitfalls
**Needham-Schroeder: Vulnerability**

If attacker finds out Client A's key $K_a$, then attacker can impersonate itself to the KDC. But if Client A changes $K_a$, then we require attacker not to be able to impersonate. But the ticket to Client B remains valid after a such a change. Attacker can save $K_a\{N_a, B, K_{ab}, K_b\{K_{ab}, A\}\}$ until it knows the old $K_a$ key, then decrypt to get $K_b\{K_{ab}, A\}$ and send that to Client B to convince it that it is connecting with Client A using $K_{ab}$.
Needham-Schroeder: Vulnerability
If attacker finds out Client A's key $K_a$, then attacker can impersonate itself to the KDC. But if Client A changes $K_a$, then we require attacker not to be able to impersonate. But the ticket to Client B remains valid after a such a change. Attacker can save $K_a\{N_a, B, K_{ab}, K_b\{K_{ab}, A\}\}$ until it knows the old $K_a$ key, then decrypt to get $K_b\{K_{ab}, A\}$ and send that to Client B to convince it that it is connecting with Client A using $K_{ab}$.
Fixed by Client A connecting to Client B requesting an encrypted nonce and using that nonce throughout the exchange.
Security Handshake Pitfalls

Otway-Rees:

\[ N_c, A, B, K_a \{ N_a, N_c, A, B \} \]
Otway-Rees:

If \( N_c \) is not the same in both messages, KDC will stop the transaction.
Security Handshake Pitfalls

Client A
\( (K_a) \)

Kab
KDC
\( (K_s) \)

Client B
\( (K_b) \)

Otway-Rees:
Otway-Rees:

Client B is reassured that it is talking to KDC since its nonce was extracted using $K_b$ and sent back encrypted.
Otway-Rees:

This message reassures Client A that both the KDC and Client B are OK because it can check its nonce, which it had encrypted with other things, and to get it back means the KDC used $K_a$ and the KDC validated Client B.
Otway-Rees: $K_{ab}\{\text{anything recognizable}\}$

Client A proves identity to Client B by showing it knows $K_{ab}$. 
Security Handshake Pitfalls

Nonce types:

Random number - will probably never be reused
Timestamp - requires synchronized clocks
Sequence number - requires non volatile memory (system crash?)
Security Handshake Pitfalls

Protocol Checklist:

**Eavesdropping:** attacker should not be able to do any of the following
- learn the contents of messages between connecting parties
- learn information enabling impersonation in a future exchange
- learn anything that permits off-line password-guessing
Security Handshake Pitfalls

Protocol Checklist:

**Eavesdropping:** attacker should not be able to do any of the following
- learn the contents of messages between connecting parties
- learn information enabling impersonation in a future exchange
- learn anything that permits off-line password-guessing

**Impersonation of Originator:** attacker should not be able to do any of these:
- convince other party it is the real originator
- learn information that would enable impersonator to do an off-line password-guessing attack against Client or Server's secret information
- learn information that would enable impersonation of Client in the future
- learn information that would enable impersonation of Server to Client
- trick Server into signing or decrypting something
Security Handshake Pitfalls

Protocol Checklist:

**Eavesdropping:** attacker should not be able to do any of the following
- learn the contents of messages between connecting parties
- learn information enabling impersonation in a future exchange
- learn anything that permits off-line password-guessing

**Impersonation of Originator:** attacker should not be able to do any of these:
- convince other party it is the real originator
- learn information that would enable impersonator to do an off-line password-guessing attack against Client or Server's secret information
- learn information that would enable impersonation of Client in the future
- learn information that would enable impersonation of Server to Client
- trick Server into signing or decrypting something

**Pounce** - attacker gets part way through authentication but should not:
- convince Client the attacker is the Server
- learn info enabling an off-line password-guessing attack
- learn info enabling impersonation of Server in future or Client to Server
- trick Client into signing or decrypting something
Security Handshake Pitfalls

Protocol Checklist:

**Read Client Database:** Bad news! Then attacker can convince Server it is the Client. Attacker can do off-line password guessing attack against Server's secret (if it is derived from a password) since Client must have enough info to know if party really is the Server. But:
- Attacker should not be able to impersonate the Server to the Client
- Attacker should not be able to decrypt recorded messages between S and C
Security Handshake Pitfalls

Protocol Checklist:

**Read Client Database:** Bad news! Then attacker can convince Server it is the Client. Attacker can do off-line password guessing attack against Server's secret (if it is derived from a password) since Client must have enough info to know if party really is the Server. But:

- Attacker should not be able to impersonate the Server to the Client
- Attacker should not be able to decrypt recorded messages between S and C

**Read Server Database:** Bad News!! Then attacker can convince Client that it is the Server. Attacker can do off-line password-guessing attack against Client's secret. But:

- Attacker should not be able to impersonate the Client to the Server
- Attacker should not be able to decrypt recorded messages between S and C
Security Handshake Pitfalls

Protocol Checklist:

**Read Client Database:** Bad news! Then attacker can convince Server it is the Client. Attacker can do off-line password guessing attack against Server's secret (if it is derived from a password) since Client must have enough info to know if party really is the Server. But:

Attacker should not be able to impersonate the Server to the Client
Attacker should not be able to decrypt recorded messages between S and C

**Read Server Database:** Bad News!! Then attacker can convince Client that it is the Server. Attacker can do off-line password-guessing attack against Client's secret. But:

Attacker should not be able to impersonate the Client to the Server
Attacker should not be able to decrypt recorded messages between S and C

**Sit on net and modify messages** – Attacker should not be able to:

do an off-line password-guessing attack on anybody's secret information
read any messages
hijack a conversation without the other side knowing this
cause messages between Client and Server to be misinterpreted