Security Handshake Pitfalls
Security Handshake Pitfalls

Client (K) \rightarrow \text{Hello} \rightarrow \text{Server (K)}

Client and Server share a secret K that no one else has. Everyone knows a hash or encrypt function $f$. 
Client and Server share a secret K that no one else has. Everyone knows a hash or encrypt function $f$. 

### Security Handshake Pitfalls
Security Handshake Pitfalls

Client and Server share a secret K that no one else has. Everyone knows a hash or encrypt function $f$.
Security Handshake Pitfalls

Concerns:
1. Handshake needs three messages, Server saves R, possible DoS
2. Off-line password guessing attack - if K is derived from password because both \( R \) and \( f(K,R) \) are observable.
3. Someone reading the Server database may be able to impersonate Client later or to another server that the client uses
4. Authentication is not mutual - only Server authenticates Client
   Anyone can send the challenge \( R \).
5. Connection can be hijacked – if the rest of the transaction is without cryptographic protection and attacker makes packets with Client addr
Client and Server share a secret K that no one else has. K\{R\} means encrypt R with key K. The encryption algorithm is known by everyone.
Client and Server share a secret $K$ that no one else has.

$K\{R\}$ means encrypt $R$ with key $K$.

The encryption algorithm is known by everyone.
Security Handshake Pitfalls

Client and Server share a secret $K$ that no one else has.

$K\{R\}$ means encrypt $R$ with key $K$.

The encryption algorithm is known by everyone.
Concerns:
1. Requires reversible cryptography - no hash, DSA cannot be used
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.
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
4. Server must still save $R$
Client and Server share a secret $K$ that no one else has.
$K\{time\}$ means encrypt time with key $K$.
Assume Client and Server are time synchronized to a few ms.
Security Handshake Pitfalls

Benefits:
1. One direction, more efficient
2. The server does not need to keep volatile state (such as R)
Security Handshake Pitfalls

**Benefits:**
1. One direction, more efficient
2. The server does not need to keep volatile state (such as R)

**Concerns:**
1. Attacker can use K\{time\} to impersonate Client - within acc clock skew
   But can be foiled if Server remembers all timestamps sent (oh boy)
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

Concerns:

4. Setting time requires a security handshake - better make it challenge/response (that is, not depending on time) or else there is a problem if clocks of machines are too far apart – the negotiation will fail.
Client (K) → Server (K)

Hello, time, hash{K,time}

Client and Server share a secret K that no one else has.

hash{K,time} means encrypt time with key K.

time is also sent in the clear.
Security Handshake Pitfalls

Benefit:
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 and Server have public and private keys

$S_c$ and $S_s$ are Client and Server private keys respectively

$K_c$ and $K_s$ are Client and Server public keys respectively
Security Handshake Pitfalls

Client and Server have public and private keys

$S_c$ and $S_s$ are Client and Server private keys respectively

$K_c$ and $K_s$ are Client and Server public keys respectively
Security Handshake Pitfalls

Client and Server have public and private keys

$S_c$ and $S_s$ are Client and Server private keys respectively

$K_c$ and $K_s$ are Client and Server public keys respectively
Security Handshake Pitfalls

Client \((K_c, S_c)\)

\[\text{signed}[R]_{S_c}\]

Server \((K_s, S_s)\)

**Benefit:**

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

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

**Serious 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.
Client and Server have public and private keys

$S_c$ and $S_s$ are Client and Server private keys respectively

$K_c$ and $K_s$ are Client and Server public keys respectively
Security Handshake Pitfalls

Client and Server have public and private keys
$S_c$ and $S_s$ are Client and Server private keys respectively
$K_c$ and $K_s$ are Client and Server public keys respectively
Security Handshake Pitfalls

Client and Server have public and private keys
$S_c$ and $S_s$ are Client and Server private keys respectively
$K_c$ and $K_s$ are Client and Server public keys respectively
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 decrypt messages sent to the Client, if Attacker can impersonate the Server.
   Attacker gets an encrypted message encrypt\(\{M\}_{K_c}\) that is sent to client
   Attacker waits for Client to initiate handshake with Server
   Attacker sends encrypt\(\{M\}_{K_c}\) as R to Client
   Client decrypts to get M, sends it to Attacker
Security Handshake Pitfalls

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 public key pair
2. Attacker chooses number $R$, prime relative to modulus $N$ ($R^{-1}$ exists)
3. Attacker creates bogus message $H = M^e \times R^e \mod N$ (recall $e,N$ are public)
4. Attacker impersonates server, sends $H$ to client for signing
5. Attacker gets $H^d \mod N = (M\times R)^{ed} \mod N = M \times R \mod N$
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 decrypt messages sent to the Client, if Attacker can impersonate the Server.

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 decrypt messages sent to the Client, if Attacker can impersonate the Server.

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

Client
(K)

Hello, \( R_c \)

Server
(K)

Mutual Authentication:
Client and Server share a secret K that no one else has. Everyone knows a hash or encrypt function \( f \). \( R_c \) and \( R_s \) are “nonces” for authentication.
**Security Handshake Pitfalls**

**Mutual Authentication:**
Client and Server share a secret K that no one else has.
Everyone knows a hash or encrypt function $f$
$R_c$ and $R_s$ are “nonces” for authentication.
Mutual Authentication:
Client and Server share a secret $K$ that no one else has.
Everyone knows a hash or encrypt function $f$.
$R_c$ and $R_s$ are “nonces” for authentication.
Imposter

Hello, $R_i$

Server

(K)

**Reflection Attack:**
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Reflection Attack:
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Attacker gets back $f(K, R_i)$ and $R_s$. Server expects $f(K, R_s)$ from client.
**Reflection Attack:**
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Attacker gets back $f(K, R_i)$ and $R_s$. Server expects $f(K, R_s)$ from client.
Attacker opens second connection to Server using $R_s$. 

**Security Handshake Pitfalls**
Reflection Attack:
Attacker wants to impersonate Client to Server. Attacker sends Hello and a challenge.
Attacker gets back $f(K, R_i)$ and $R_s$. Server expects $f(K, R_s)$ from client.
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 $f(K, R_s)$ and $R_s$. Server expects $f(K, R_s)$ from client.
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
Imposter

Server

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:
Client and Server share a secret $K$ that no one else has
Everyone knows a hash or encrypt function $f$
$R_c$ and $R_s$ are “nonces” for authentication
Reflection Attack can't succeed with this protocol:
Client and Server share a secret K that no one else has
Everyone knows a hash or encrypt function \( f \)
\( R_c \) and \( R_s \) are “nonces” for authentication
Reflection Attack can't succeed with this protocol:
Client and Server share a secret K that no one else has
Everyone knows a hash or encrypt function $f$
$R_c$ and $R_s$ are “nonces” for authentication
Security Handshake Pitfalls

Reflection Attack can't succeed with this protocol:
Client and Server share a secret K that no one else has.
Everyone knows a hash or encrypt function \( f \).
\( R_c \) and \( R_s \) are “nonces” for authentication.
Security Handshake Pitfalls

**Reflection Attack can't succeed with this protocol:**
Client and Server share a secret $K$ that no one else has.
Everyone knows a hash or encrypt function $f$
$R_c$ and $R_s$ are “nonces” for authentication
Reflection Attack can't succeed with this protocol:
Reason: Initiator always is first to prove its identity!!
Security Handshake Pitfalls

Public key mutual authentication:
Client and Server have public and private keys
$S_c$ and $S_s$ are Client and Server private keys respectively
$K_c$ and $K_s$ are Client and Server public keys respectively
Security Handshake Pitfalls

Public key mutual authentication:
Client and Server have public and private keys
$S_c$ and $S_s$ are Client and Server private keys respectively
$K_c$ and $K_s$ are Client and Server public keys respectively
Public key mutual authentication:
Client and Server have public and private keys
$S_c$ and $S_s$ are Client and Server private keys respectively
$K_c$ and $K_s$ are Client and Server public keys respectively
Security Handshake Pitfalls

Public key mutual authentication:

Reflection attack will not succeed although the server proves its ID first. Having seen $\text{encrypt}\{R_s\}_{K_c}$ an attacker needs to extract $R_s$

It can't decrypt it because it does not know the client's private key.

It can't send $\text{encrypt}\{R_s\}_{K_c}$ to the server in a new login, server tries this:

$$\text{decrypt}\{\text{encrypt}\{R_s\}_{K_c}\}_{S_s}$$

to no avail.
Security Handshake Pitfalls

Public key mutual authentication:

Variant: \( R_s \) and \( R_c \) are signed by their respective parties
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
Security Handshake Pitfalls

Session Keys Are Important:
Use long term keys to produce session keys which are different for each transaction
Establish a session key (secret key system):
After authentication, $R$ and $K\{R\}$ are 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 $K\{R\}$ are 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.
Each party decrypts \([g^x \mod p]_S\), compares with also sent \(g^x \mod p\)
Attacker cannot produce \([g^x \mod p]_S\) in either direction
Use \(g^{ab} \mod p\) as the session key.

Attempt to get root access on either machine does not reveal the \(a\) and \(b\) - 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 DH halves get sent across without being signed, Attacker can succeed with MinM attack.
Security Handshake Pitfalls

KDC operation, in principle

Client A
(K_a)

K_a{K_{AB}}

Request B

KDC
(K_s)

K_{s}{K_{AB}}

Client B
(K_b)

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, $K_{ab}$, a ticket T with key $K_a$ and sends all to A.

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

**Needham-Schroeder:** What is going on?  
Nonce $N_a$ is checked by A in message from KDC to protect against:
Needham-Schroeder: What is going on?
Nonce $N_a$ is checked by A in message from KDC to protect against:
   Attacker steals a $K_b$ and holds onto to it – maybe it is changed by B.
**Security Handshake Pitfalls**

**Needham-Schroeder:** What is going on?

Nonce $N_a$ is checked by A in message from KDC to protect against:
- Attacker steals a $K_b$ and holds onto to it – maybe it is changed by B.
- Attacker waits for A to ask for session key with B
Security Handshake Pitfalls

**Needham-Schroeder:** What is going on?

Nonce $N_a$ is checked by $A$ in message from KDC to protect against:

Attacker steals a $K_b$ and holds onto to it – maybe it is changed by $B$.

Attacker intercepts KDC response $M$ to $A$ requesting a ticket for $B$, holds it

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

$T = K_b \{K_{ab}, A\}$
**Needham-Schroeder:** What is going on?

Nonce $N_a$ is checked by A in message from KDC to protect against:

- Attacker steals a $K_b$ and holds onto it – maybe it is changed by B.
- Attacker intercepts KDC response $M$ to A requesting a ticket for B, holds it
- Attacker waits for A to request a session key for B and plays back $M$ to A
- A sends session key encrypted with B's $K_b$, Attacker impersonates B to 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?

Nonce $N_a$ is checked by A in message from KDC to protect against:

- Attacker steals a $K_b$ and holds onto to it – maybe it is changed by B.
- Attacker intercepts KDC response $M$ to A requesting a ticket for B, holds it
- Attacker waits for A to request a session key for B and plays back $M$ to A
- A sends session key encrypted with B's $K_b$, Attacker impersonates B to A

Attacker need not know key of Client A

Nonce $N_a$ will be different from $N_a$
**Needham-Schroeder:** What is going on?

- Identity of B returned so A knows who this key is for
  - (there may be multiple requests for keys by A)
- Nonce is also 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 B's key $K_b$ can decrypt and get the session key. With session key, B decrypts $N_b$. 
**Needham-Schroeder: Vulnerability**

If attacker finds out A's key $K_a$, then Attacker can impersonate A to the KDC.

If A changes $K_a$, then we require that Attacker is not to be able to impersonate A.

But the ticket to 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 B
to convince it that it is connecting with Client A using $K_{ab}$.
Needham-Schroeder: Vulnerability
If attacker finds out A's key $K_a$, then Attacker can impersonate A to the KDC.
If A changes $K_a$, then we require that Attacker is not to be able to impersonate A.
But the ticket to 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 B to convince it that it is connecting with Client A using $K_{ab}$.

Fixed by Client A first 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.

Only someone with knowledge of B's current $K_b$ can encrypt $N_c$ properly.

Only someone with knowledge of A's current $K_a$ can encrypt A, B so it can successfully be decrypted by the KDC, verifying the value of the decrypted $N_c$. 
Security Handshake Pitfalls

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.
Security Handshake Pitfalls

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:
1. learn the contents of messages between connecting parties
2. learn information enabling impersonation in a future exchange
3. 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:
1. learn the contents of messages between connecting parties
2. learn information enabling impersonation in a future exchange
3. learn anything that permits off-line password-guessing

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

Protocol Checklist:

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

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

**Pounce** - attacker gets part way through authentication but should not be able to:
1. convince a Client that the attacker is a legitimate Server
2. learn information enabling an off-line password-guessing attack
3. learn info enabling impersonation of a Server in the future or a Client to a Server
4. trick a 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:
   1. Attacker should not be able to impersonate the Server to the Client
   2. 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:
   1. Attacker should not be able to impersonate the Server to the Client
   2. 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:
   1. Attacker should not be able to impersonate the Client to the Server
   2. 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:
  1. Attacker should not be able to impersonate the Server to the Client
  2. 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:
  1. Attacker should not be able to impersonate the Client to the Server
  2. 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:
  1. do an off-line password-guessing attack on anybody's secret information
  2. read any messages
  3. hijack a conversation without the other side knowing this
  4. cause messages between Client and Server to be misinterpreted