Introduction to Security

Luis Garcis
USC CSCI 430

Module 8 – Key Management – Shared Key Exchange

(Ch 20.6, 21.5 and 23.1)
Crypto Recap

• One-way-hashes
  – Ensure data integrity

• Symmetric Cryptography
  – Uses a shared secret key for encryption and decryption
  – Efficient for large amounts of data

• Asymmetric Cryptography
  – Uses a pair of keys: public and private
  – Slower, but provides non-repudiation

• Shared Key Exchange
  – Combines strengths to provide secure and efficient method
Shared Key Exchange Problem

• How do Alice and Bob exchange a shared secret?
• Offline
  – Doesn’t scale
• Using specially crafted messages (Diffie-Hellman)
• Using a trusted third party (KDC)
  – Secrets should never be sent in clear
  – We should prevent replay attacks
  – We should prevent reuse of old keys
• Using public key cryptography (possible)
Diffie-Hellman Key Exchange
(Ch 21.5)

• Exchange a secret with someone you never met while shouting in a room full of people
  • Alice and Bob agree on $g$ and large $n$
  • Alice chooses random $a$, sends $g^a \mod n$
  • Bob chooses random $b$, sends $g^b \mod n$
  • Alice takes Bob’s message and calculates $g^{ab} \mod n = (g^b \mod n)^a \mod n = \text{shared secret}$. Bob does the same.
Discrete Logarithms

• Why is it hard for anyone to guess \( a \) and \( b \) in Diffie-Hellman key exchange?
  
  – Why is brute force the only possible way of guessing?

• Let \( g \) be a “primitive root” of \( n \)
  
  – Powers of \( g \) generate all numbers 1..\( n-1 \)

• For any integer \( x<n \) one can find \( a \) s.t.:
  
  \[ x=g^a \pmod{n} \]

• \( a \) is called the “discrete logarithm” of \( x \)

• There is no algorithm to efficiently calculate \( a \) given \( x, g \) and \( n \)
  
  • \( x, g, \text{ and } n \text{ are visible to everyone} \)
Uses of Diffie-Hellman

• Alice and Bob can exchange a shared key
• Multiple users can:
  – Select their “private” key Pr_i
  – Calculate their “public” key x_i = g^{Pr_i} \mod n
    and publish it in a central directory
  – When another user wants to talk to them, they can calculate a shared key for this communication
  – On-demand shared key exchange
Diffie-Hellman

• What can go wrong?

\[ g^{ab} \mod n \]

Alice

Bob

\[ g, n, g^a \mod n \]

\[ g^b \mod n \]
Mallory sits between Alice and Bob and replaces their messages with her own.

When Alice and Bob send messages, Mallory decrypts them and re-encrypts them.

This works because there’s no authentication.
KDC Based Key Distribution
(Ch 20.6)

• Building up to Needham Schroeder/Kerberos
• Client sends req. to KDC (key distrib. center)
• KDC generates a shared key: $K_{c,s}$

Keys $K_{KDC,C}$ and $K_{KDC,S}$ are preconfigured
• No keys ever traverse net in the clear
• Why are identities in tickets?
KDC Based Key Distribution

- KDC does not have to talk both to C and S

\[
ticket_s = EK_{KDC,S}\{C, K_{c,s}\}
\]

What happens when Mallory comes in to play?
KDC Based Key Distribution

- KDC does not have to talk both to C and S
  \[ \text{ticket}_S = EK_{KDC,S}\{C, K_{c,s}\} \]

- Messages 2 or 3 can be replayed by Mallory
  - Force C and S to use same secret for a long time
  - Cause S to have an old ticket, break communication with C
  - Can we just add timestamps?

![Diagram of KDC-Based Key Distribution](image)
Needham-Shroeder Key Exchange

- Use nonces to prevent replay attacks
  - "nonce" – random number used once

\[
ticket_s = EK_{KDC,S}\{C, K_{c,s}\}
\]

1. \(N_1, C, S\)
2. \(EK_{KDC,C}\{N_1, S, K_{c,s}, ticket_s\}\)
3. \(EK_{C,S}\{N_2\}, ticket_s\)
4. \(EK_{C,S}\{N_2-1\}EK_{C,S}\{N_3\}\)
5. \(EK_{C,S}\{N_3-1\}\)
Challenge-Response

• Used when C wants to check if S knows the same shared key $K_{c,s}$

• 3 Flavors:
  – C selects a random number $R$ and sends $E_{K_{c,s}}(R)$, S decrypts this and sends back $R$
  – C selects a random number $R$ and sends to S, S has to return $E_{K_{c,s}}(R)$
  – C selects a random number $R$ and sends $E_{K_{c,s}}(R)$ to S, S decrypts, transforms the number (usually calc $R-1$), reencrypts the result and sends back to C (Needham-Shroeder)
Problem

• What happens if attacker gets session key?
  – Can reuse old session key to answer challenge-response, generate new requests, etc
  – Need validity period in ticket to ensure freshness = tickets expire after some time

```
1. N1, C, S → KDC
2. EK_{KDC,C}{N1, S, Kc,s, ticket_s} → C
3. EK_{C,S}{N2}, ticket_s → S
4. EK_{C,S}{N2-1}EK_{C,S}{N3} → S
5. EK_{C,S}{N3-1} → S
```

ticket_s = EK_{KDC,S}{C, Kc,s}
Introduce Ticket Granting Server (TGS)
- Issue tickets for clients to talk to servers

Authentication server (AS) authenticates users
- Issues a ticket for clients to talk to TGS

TGS+AS = KDC (2 servers)

Each ticket has a validity period: timestamp and lifetime

Each service request (client to server) has an authenticator (nonce): timestamp + client identity, encrypted with a session key
Kerberos

1. User logs on to workstation and requests service on host.

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.

3. Workstation prompts user for password to decrypt incoming message, then send ticket and authenticator that contains user's name, network address and time to TGS.

4. TGS decrypts ticket and authenticator, verifies request then creates ticket for requested application server.

5. Workstation sends ticket and authenticator to host.

6. Host verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.
Kerberos

What to remember:
- Similar Needham-Shroeder, except 2 key exchanges (NS had one)
- Additional info in messages that prevents replay attacks and reuse of old keys
Key Exchange Using Public Keys

• Alice selects a shared key, encrypts it with Bob’s public key – only Bob can read

\[ \text{EPub}_B(K_{AB}) \]

• Why not just use public keys?
  – It would be much slower (roughly 1500x)