Security Handshake Pitfalls
(Chapter 11 & 12.2)

Outline

Login Only Authentication (One Way)
- Login w/ Shared Secret
- One-way Public Key
- Lamport’s Hash

Mutual Authentication
- Shared Secret
- Public Keys
- Timestamps

Login w/ Shared Secret: Variant 1

I’m Alice

a challenge R

f(K_{Alice-Bob}, R)

Bob

Protocol 11-1

- Bob: challenge R, Alice: f(K_{Alice-Bob}, R). f can be hash or secret key based
  - K_{Alice-Bob} could not be derived by the eavesdropper

- Problems:
  - Authentication not mutual
  - Connection hijacking after initial exchange if Alice’s address is spoofed
  - Off-line password guessing attack (assume K is derived from password)

Login With Shared Secret: Variant 2

I’m Alice

K_{Alice-Bob}(R)

R

Bob

Protocol 11-2

- Problems:
  - Requires reversible Cryptography
  - Vulnerable to dictionary attack (if key K_{Alice-Bob} derived from passwd)
  - For mutual authentication, R must be limited lifetime to foil the replaying of a K_{Alice-Bob}(R): if R is relevant to timestamp
**Login With Shared Secret: One Way**

- Alice: \( K_{AB} (\text{timestamp}) \); easy to add; efficient
- Problems:
  - Requires synchronized clocks
  - Replay attacks
    - Use for other server if several servers share the same secret; if add server name
  - Reuse if server clock can be set back

**One-way Public Key**

- Shared secret scheme is subject to impersonation attack if Bob’s database is disclosed
- Public key
  - Alice: \( R \); Alice: \( R \), (Alice signs \( R \))
  - Alice: \( R \); (Bob encrypts using Alice’s public key); Alice: \( R \)
  - Database at Bob only write-locked, not read-locked
- Problems: should not use the same key for two different purpose, unless...
  - Can trick Alice into signing or decrypting message (if bob’s address is spoofed)
  - To fix: explicitly specifies the message type, type field...(or, different keys)

**Lamport’s Hash** (Chapter 12.2)

- Safe from eavesdropping and database reading
- No public key cryptography
- Alice (human + workstation): password
- Bob (server): get “username, \( n \), hash(password)” during registration
- Authentication:
  - Human: <Alice, password> → workstation; Alice: name → Bob; Bob \( n \) → Alice;
  - Alice: \( x = \text{hash}^n (\text{password}) \) → Bob;
  - Bob: compares hash(x) with database, stores new \( <n-1, x> \); \( n \) decrease by 1
- Password reset if \( n \) gets to 1

**Lamport’s Hash: Small \( n \) Attack**

- No mutual authentication
- Trudy impersonates Bob:
  - Sends small \( n \), say 50, to Alice
  - Alice sends back hash\(^{50}\)(password)
- Trudy then impersonates Alice
  - The actual \( n \) in the real bob is greater than 50
  - Trudy can compute hash\(^{51}\), hash\(^{51}\), … hash\(^{n}\)
- Alice workstation display \( n \) to human Alice; If Alice remember
Mutual Authentication: Shared Secret
- **Straightforward**
  - Alice → Bob: I'm Alice;
  - Bob → Alice: $R_1$
  - Alice → Bob: $K_{AB}(R_1)$
  - Bob → Alice: $K_{AB}(R_2)$

- **Simplified:**
  - Alice → Bob: I'm Alice,
  - Bob → Alice: $R_1$, $K_{AB}$
  - Alice → Bob: $R_2$

Mutual Authentication: Reflection Attack
- **First login connection by Trudy:**
  - Trudy → Bob: I'm Alice, $R_2$.
  - Bob → Trudy: $R_1$, $K_{AB}$
  - Can't do Trudy → Bob: $K_{AB}(R_1)$ yet.

- **Second login connection by Trudy:**
  - Trudy → Bob: I'm Alice, $R_1$.
  - Bob → Trudy: $R_3$, $K_{AB}$
  - Go back to first connection and do:
    - Trudy → Bob: $K_{AB}(R_1)$
    - (Forget about the second connection).

Mutual Authentication: Reflection Attack (Cont’d)
- **Fixes:**
  - **General principle 1:** Alice & Bob won’t do same thing
    - Different keys for initiator and responder
      - Trudy can’t get Bob to encrypt using Alice’s key
      - How?
    - Different types of challenges for initiator and responder
      - e.g., concatenate the name of party to the challenge
      - e.g., initiator challenge be odd number…
  - **General principle 2:** the initiator should prove its identity first (e.g. Protocol 11-7 is not subject to reflection attack)

Mutual Authentication: Public Keys
- **Variant:**
  - Sign instead of encrypt
- **Challenges:**
  - Public key distribution and storage
  - Difficulty of deriving private key (e.g., RSA) from password
    - Encrypt private key by password
Mutual Authentication: Timestamps (Shared Secret)

- Eliminate exchanges of challenges
  - Alice → Bob: I'm Alice, \( K_{ab}(t) \)
  - Bob → Alice: \( K_{ab}(t+1) \)
- Problem:
  - Trudy eavesdrops, gets \( K_{ab}(t+1) \), and uses it to impersonate Alice (i.e., sends it to Bob...)
  - Fixes:
    - Bob remembers messages (timestamps)
    - Includes direction flag

Integrity/Encryption after Authentication: Establishing Session Keys

- Authentication handshakes to securely establish session keys: recall Kerberos!
  - Using shared secret
  - Using public keys
  - One-way public key (only Alice needs to have keys)

Session Key: via Shared Secret

- Alice → Bob: \( \{R\}_B \)
- Bob → Alice: \( \{R\}_A \)
- \( R \) is session key (after public key based mutual authentication)
- Alice → Bob: \( \{R\}_A \)
- Trudy can impersonate Alice and send her own \( \{R\}_B \) to Bob (hijack com., pick her own \( R \), & spoof Alice's address)
- Alice → Bob: \( \{R\}_B \)
- Trudy can record conversations, later break into Bob, and decrypt (reading: p271)
- \( R \) is session key, Trudy needs to break into both Alice and Bob
- Diffie-Hellman with signing: \( g^{Ra \mod p} \), \( g^{Rb \mod p} \). Session key: \( g^{Ra \mod p} \)
Session Key: via One-Way Public Key

- Only one of the parties has a public/private key
- Alice → Bob: \( \{K_B \} \)
  - Trudy can record conversations, break into Bob, and decrypt
- Diffie-Hellman with signing: \( g^{KA} \mod p \), \( [g^{KB} \mod p]_B \), Session key: \( g^{KA} \cdot KB \mod p \)

Mediated Authentication

- Alice wants Bob
  - KDC: invents \( K_{AB} \)
  - KDC → Alice: \( K_{AB} \) (use \( K_{AB} \) for Bob)
  - KDC → Bob: \( K_{BA} \) (use \( K_{AB} \) for Alice)
  - Avoid "race" condition:
    - KDC sends "ticket" = \( KB \) (use \( K_{AB} \) for Alice) to Alice, who then uses the ticket to contact Bob

Needham-Schroeder

- Alice wants Bob
  - KDC: invents \( K_{AB} \)
  - KDC → Alice: \( K_{AB} \) (ticket to Bob)
  - ticket = \( KA \) (Alice)
  - KDC: ticket = \( KA \) (Alice)
  - Ensure that both KDC and Bob are legit

- Bob wants Alice
  - KDC: invents \( K_{AB} \)
  - KDC → Bob: \( K_{BA} \) (ticket)
  - Bob: ticket = \( KB \) (Alice)

- KDC uses \( K_{AB} \) to authenticate Bob

- Alice wants Bob
  - KDC: invents \( K_{AB} \)
  - KDC → Alice: \( K_{AB} \) (ticket)
  - Alice: ticket = \( KA \) (Bob)

- Flaws: (Reading assignment: p275-277)
  - Trudy can record conversations, break into Bob, and decrypt

Otway-Rees (reading assignment: p277-280)

- Idea: suspicious party should generate a challenge
  - Alice → Bob: \( N_C \), "Alice", "Bob"; \( KA(N_C, N_C, "Alice", "Bob") \)
  - Bob: \( KA(N_D, N_D, "Alice", "Bob") \)
  - KDC uses \( N_C \) to authenticate Bob
  - KDC→Bob: \( N_D \), \( KA(N_D, N_D, "Alice", "Bob") \)

- Bob: \( KA(N_D, N_D, "Alice", "Bob") \) (ticket)

- Flaws: (Reading assignment: p277-278)
Performance Considerations

- Metrics to evaluating performance
  - No. of crypt. operations (blocks) using a private key
  - No. of crypt. operations (blocks) using a public key
  - No. of bytes encrypted/decrypted using a secret key
  - No. of bytes to be hashed
  - No. of messages transmitted

- Reading Assignment (Chapter 11.8): pp284-287
  - Authentication Protocol Checklist: organized around what Trudy might attempt to do.