Authentication Protocols and Key Establishment

Peter Sjödin
psj@kth.se

Based on material by Vitaly Shmatikov, Univ. of Texas, and by the previous course teachers
Authentication & handshakes

• Authentication & pitfalls

• Trusted intermediates

• Performance & randomness
Basic Problem

How do you prove to someone that you are who you claim to be?

Any system with access control must solve this problem.
Authentication: first attempt

Alice says “I am Alice” and sends her secret password to “prove” it.

Failure scenario??
Authentication: Playback Attack

Alice says “I am Alice” and sends her secret password to “prove” it.

playback attack: Trudy records Alice’s packet and later plays it back to Bob
Authentication: yet another try

Alice says “I am Alice” and sends her encrypted secret password to “prove” it.

Failure scenario??
Authentication: another try

Alice says “I am Alice” and sends her encrypted secret password to “prove” it.
Authentication by Nonce Challenge

**Goal:** avoid playback attack

**Nonce:** number (R) used only *once-in-a-lifetime*

Bob sends Alice a nonce R. Alice must return R, encrypted with shared secret key

“*I am Alice*”

\[ R \]

\[ K_{A-B}(R) \]

Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice! Right?

Failures, drawbacks?
Authentication by Nonce Challenge

**Goal:** avoid playback attack

**Nonce:** number (R) used only *once-in-a-lifetime*

Bob sends Alice a nonce (R). Alice must return R, encrypted with a shared secret key

- Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice! Right?
- Bob isn’t authenticated
- If the key is derived from a password, Trudy can mount a dictionary attack
- Trudy can hijack connection after authentication, if she can send packets with Alice’s source address!

Failures, drawbacks?
Authentication: yet another try

**Goal:** avoid playback attack, efficiency

Alice encrypts a timestamp with shared secret key

"I am Alice, $K_{A-B}(\text{timestamp})$"

Only Alice knows key to encrypt current time.

Failures, drawbacks?
Authentication: yet another try

Goal: avoid playback attack, efficiency

Alice encrypts a timestamp with shared secret key

“I am Alice, KA-B(timestamp)”

- Requires synchronization
- In practice, there must be some margins for time skew
  - During the time-skew window, there is room for playback attacks and for Trudy to impersonate Alice
- Bob is still not authenticated

Failures, drawbacks?
Mutual Authentication With Symmetric Encryption

- **Mutual authentication**: Bob to Alice and Alice to Bob

- **Bob's reasoning**: I must be talking to Alice because...
  - Person who correctly encrypted $R_B$ is someone who knows $KEY$... Only Alice knows $KEY$... Alice must have encrypted $R_B$... Because $R_B$ is fresh, Alice can only know $R_B$ if she received my message
Reflection Attack

Bob’s reasoning: I must be talking to Alice because...
- Person who correctly encrypted $R_B$ is someone who knows KEY... Only Alice knows KEY... No! Bob himself knows KEY, too!

Security often fails because of flawed reasoning
Timestamp Reflection

- Problem: same key for Alice and Bob
  - Attacker can get Bob to encrypt using Alice’s key
- Problem: messages don’t include intended recipient
- Problem: Bob doesn’t remember his own messages
Authentication with Public Key

Use nonce, public key cryptography

“"I am Alice”

Bob checks that $K_A^+(K_A^-(R)) = R$

Only Alice could have encrypted $R$ that way!
Man-in-the-middle attack

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)

Difficult to detect:
- Trudy receives all messages as well
- Replay attacks
  - Must not allow Trudy to abuse the authentication to create bogus messages!
Encryption Tricking

Encryption tricking: Trudy tricks Alice to encrypt something with her private key

• Alice can be tricked into providing:
  - Digital signature
  - Message decryption
  - ...
• Key reuse should be avoided
• Data should have structure
  - (so Alice would know what kind of data she signs, encrypts, etc)

I am Alice

\[ K_A(M) \]

\[ M \]
Authentication & handshakes

• Authentication & pitfalls
• Trusted intermediates
• Performance & randomness
Trusted Intermediaries

**Symmetric key problem:**
- How do two entities establish shared secret key over network?

**Solution:**
- trusted key distribution center (KDC) acting as intermediary between entities

**Public key problem:**
- When Alice obtains Bob’s public key (from web site, e-mail, diskette), how does she know it is Bob’s public key, not Trudy’s?

**Solution:**
- trusted certification authority (CA)
Key Distribution Center (KDC)

- Alice, Bob need shared symmetric key.
- **KDC**: server shares different secret key with *each* registered user (many users)
- Alice, Bob know own symmetric keys, $K_{A-KDC}$ $K_{B-KDC}$, for communicating with KDC.
Key Distribution Center (KDC)

Alice knows $KA-B$ for communication with Bob.

Alice and Bob can communicate using $KA-B$ as session key for shared symmetric encryption.

KDC generates $KA-B$.

Ticket to Bob.

Bob knows $KA-B$ for communication with Alice.
Needham-Schroeder Protocol

\[ K_{A-KDC}(N_1, A, B) \]

\[ K_{A-KDC}(N_1, B, K_{A-B}, K_{B-KDC}(A, K_{A-B})) \]

\[ K_{B-KDC}(A, K_{A-B}), K_{A-B}(N_2) \]

\[ K_{A-B}(N_2-1, N_3) \]

\[ K_{A-B}(N_3-1) \]
Needham-Schroeder Protocol

nonce to prevent against replay attacks

Includes “B” so Alice knows $K_{A-B}$ is for communication with Bob (and not with Trudy...)

Bob could decrypt $N_2$ $\Rightarrow$ he could decrypt $K_{A-B}$ $\Rightarrow$ he knew $K_{B-KDC}$ and must be Bob!

Alice could decrypt $N_3$ $\Rightarrow$ she knows $K_{A-B}$
Ticket Invalidation Problem

1. Trudy sees the message from KDC to Alice
2. Trudy steals Alice's key $K_{A-KDC}$
3. Alice changes to a new key
4. Trudy can still trick Bob to communicate with her, using Alice's old key. Bob cannot detect that the key is invalid!
Key Distribution Center (KDC)

• Many subtle problems to consider
  - How to prevent key reuse by attacker?
  - What kind of nonce do we use (more later)
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides “proof of identity” to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E’s public key digitally signed by CA
    - CA says “this is E’s public key”

\[
\begin{align*}
\text{Bob’s public key} & \quad K_B^+ \\
\text{digital signature (encrypt)} & \quad \text{CA private key} \quad K_{CA}^-
\end{align*}
\]

\[
\begin{align*}
\text{certificate for Bob’s public key, signed by CA} & \quad K_B^+
\end{align*}
\]
Certification Authorities

- When Alice wants Bob’s public key:
  - gets Bob’s certificate (Bob or elsewhere).
  - apply CA’s public key to Bob’s certificate, get Bob’s public key

\[ K_B + \text{digital signature (decrypt)} \]

\[ K_{CA} \]

\[ K_B \]
A certificate contains:

- Serial number (unique to issuer)
- Info about certificate owner, including algorithm and key value itself (not shown)
- Info about certificate issuer
- Valid dates
- Digital signature by issuer
Certificates & CAs

- No prior arrangement needed with peer
- CA does not need to be online for verification
  - Peers can distribute their own certificates
  - Only need to have received CA's public key securely at some point in the past
  - **Note:** Online CA required to support revocation!
Authentication & handshakes

• Authentication & pitfalls
• Trusted intermediates
• Performance & randomness
Performance and randomness

- Performance issues force us to encrypt only as much as is absolutely necessary
  - Encryption and decryption times can differ
    - RSA 2048 decrypt = 60 * RSA 2048 encrypt
  - Public key cryptos often orders of magnitude more expensive than symmetric crypto

- Message exchanges are also very expensive
  - ~100ms RTT to the US
  - Limit number of messages, piggybacking
    - Sometimes leads to weaknesses!
Performance and randomness

• Randomness is very important!
  – True randomness is hard to achieve
    • Radioactive decay
    • Source of white noise
  – Pseudo-randomness = pseudo-security
    • Need good seeds to pseudorandom generators
    • Seeds must not be exposed!
    • “Normal” rand()-functions not random enough!

• Who controls source of randomness?
Nonces - Numbers used once

- Nonces injects “noise” into protocols
  - Sequence numbers
  - Time stamps
  - Large random numbers

- Sequence numbers and time stamps can be attacked by guessing and may repeat
  - If you know $N_x$, you can guess $N_{x+1}$
  - Setting back clock/lost state
Summary

- Like cryptographic algorithms, the design of an authentication algorithm requires a lot of skill and care - *don’t do this at home!*

- Using KDCs and CAs allows communication with unrelated peers - **trusted intermediate**

- Performance considerations impact on the design of authentication algorithms

- Randomness is both tricky to achieve and important for the result