Key Distribution & Certificate Authority
The Problem

Key Distribution Problem
Publish public key with RSA
How do you trust/authenticate the public key?

How can they establish the common secret?
Solution for secret keys

- We will need $n^2$ keys for $n$ users!!
  - One per pair of users
- Employ a trusted server
- Server stores all the secret keys
- User A talks to TS before talking to B
- TS helps in establishing a secret key for A and B to communicate
KDC – Secret Keys

- Alice ––\(E_{ka}(\text{Bob})\) – KDC
- KDC chooses \(K_{ab}\)
- KDC –\(E_{ka}(\text{Bob, } K_{ab})\) – Alice
- KDC –\(E_{kb}(\text{Alice, } K_{ab})\) – Bob
- Alice and Bob can use their keys to decrypt and find \(K_{ab}\)
- Can use \(K_{ab}\) to communicate
KDC - secret keys

- Alternate: KDC can send $E_{kb}(Alice, K_{ab})$ to Alice who can send it along with her message to Bob.
- Bob can use $K_b$ to decrypt and get $K_{ab}$.
- Can use $K_{ab}$ to communicate.
- Difference from Diffie-Hellman?
  - Uses only secret keys, no PKI.
KDC problems

- One server with every one’s secret keys
- Trusted server broken!!!
Kerberos

- Authentication service for distributed services
- Restrict access to services to authorized users
- Authenticate requests for service
- Can’t trust workstation to identify users correctly
Threats to distributed service

- User accesses some one else’s machine
- Workstation’s network address can be altered
- User can eavesdrop on the LAN and replay packets
- Kerberos – entirely based on symmetric keys
Security Strategies?

- Rely on passwords for access to individual workstations – provide services based on user’s ID
- Require workstations/clients to authenticate themselves to servers, but trust the workstation in verifying the user’s ID
- Require users to prove identity for each service and require Servers to prove identity to clients – employed by Kerberos
Kerberos Requirements

- Secure (e.g., against replay attacks)
  - Should not be the weak link
- Reliable
  - Services depend on it
- Transparent
  - User should ideally do nothing extra for authentication
- Scalable – support large number of users
An authentication approach

- Employ an Authentication Server
- Client $- (ID_c, P_c, ID_v) \rightarrow AS$
  - ID of client, password, ID of server
- AS $- (Ticket) \rightarrow Client$
- Client $- (ID_c, Ticket) \rightarrow Server$
- Ticket $= E_{kv}(ID_c || AD_c || ID_v)$
  - Client ID, Client address, ID of server encrypted with Server’s key
Any problems?

- User has to enter a password each time for each service
  - Make Tickets reusable
  - Still need a password for different services
- Password being sent to AS in plaintext
  - Eavesdrop and replay attack
Second Approach

- Once per user logon Session
  - C – (ID_c||ID_tgs) → AS
  - AS – (E_{kc}[Ticket_{tgs}]) → C

- Once per type of Service
  - C— (ID_c || ID_v || Ticket_{tgs}) → TGS
  - TGS – Ticket_v → C

- Once per service session
  - C – (ID_c||Ticket_v) → V

- \(\text{Ticket}_{tgs} = E_{ktgs}(ID_c||AD_c||ID_{tgs}||TS_1||Time_1)\)

- \(\text{Ticket}_v = E_{kv}(ID_c||AD_c||ID_v||TS_2||Time_2)\)
Solved problem?

- Reduced the number of passwords
- But, replay attack on step 3 possible
  - Not Quite secure
- We need to verify that granted ticket belongs to the user requesting service
- Server not authenticated
  - Can redirect user requests to fake server
Authentication - tickets

- AS provides client and TGS a secret in a secure manner
- Client can prove its id by revealing the secret to TGS
- Use a session key as secret
- AS sends
  - Session key to client encrypted with client’s secret key
  - Ticket (contains session key) is encrypted with TGS’s secret key
- Client uses session key to encrypt messages to TGS
- TGS decrypts Ticket using its secret key
- Then uses session key to verify client
## Kerberos

### (a) Authentication Service Exchange: to obtain ticket-granting ticket

1. **C → AS:** \( ID_c \| ID_{tgs} \| TS_1 \)

2. **AS → C:** 
   \[
   E_{K_c} [ K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs} ]
   \]
   
   \[ Ticket_{tgs} = E_{K_{tgs}} [ K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2 ] \]

### (b) Ticket-Granting Service Exchange: to obtain service-granting ticket

3. **C → TGS:** \( ID_v \| Ticket_{tgs} \| Authenticator_c \)

4. **TGS → C:** 
   \[
   E_{K_{c,tgs}} [ K_{c,v} \| ID_v \| TS_4 \| Ticket_v ]
   \]
   
   \[ Ticket_{tgs} = E_{K_{tgs}} [ K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2 ] \]

   \[ Ticket_v = E_{K_v} [ K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4 ] \]

   \[ Authenticator_c = E_{K_{tgs}} [ ID_C \| AD_C \| TS_3 ] \]

### (c) Client/Server Authentication Exchange: to obtain service

5. **C → V:** \( Ticket_v \| Authenticator_c \)

6. **V → C:** 
   \[
   E_{K_{c,v}} [ TS_5 + 1 ] \text{ (for mutual authentication)}
   \]
   
   \[ Ticket_v = E_{K_v} [ K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4 ] \]

   \[ Authenticator_c = E_{K_{c,v}} [ ID_C \| AD_C \| TS_5 ] \]
Overview of Kerberos

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

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

5. Workstation sends ticket and authenticator to server.

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.

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

6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.
Certificate authority (CA) signs “certificates”

CA signs that “123456789 is the public key of Alice”
  - Signed using CA’s private key

Have CA’s public key, can decrypt the certificate to find the correct public key of Alice

Alice can pass this certificate around to authenticate herself
Authentication --PKI

- Alice – $E_{CA}(Alice, 12345678) \rightarrow Bob$
- Bob – $E_{CA}(Bob, 97654321) \rightarrow Alice$
- Bob and Alice are authenticated to each other
- Who authenticates CA?
KDC vs CA

- **KDC features**
  - All user secret keys in one database
  - Must be available all time
  - Need replication for performance, availability

- **CA features**
  - Simple, need not be online all the time
  - High processing complexity
Multiple CAs

- Alice gets certificate from CA1
- Bob gets certificate from CA2
- How do they authenticate each other?
  - Alice needs public key of CA2
  - Bob needs public key of CA1
  - CA2, CA1 exchange their public keys securely
  - Alice gets public key of CA2 from CA1
  - Bob gets public key of CA1 from CA2
- Chain of trust Alice → CA1 → CA2 → Bob
Authenticating CA?

- Build a chain of trust
- CA1 authenticated by CA2, CA2 authenticated by CA3...so on...
- How to trust the root?
- Root CA will be monopoly – can charge extra for Certificates
- Mutual authentication
- CA1 can certify CA2, CA2 can certify CA1
  - Any problems?
Chain of trust
Authenticating CA

- Employ a bunch of CAs
- No Monopoly
- Can get Certificates from anyone
  - Improve availability, reliability
- Security any weaker than single CA?
- Employ DNS style CAs?
  - Each CA responsible for part of the domain
Key Revocation

- Easy in KDC - modify KDC database
- Old key won’t work anymore
- Will need to alert existing sessions
- Employ Certificate Revocation List with CA
- CRL published regularly
  - Can issue deltas to CRL
X.509 authentication service

- X.509 certificate format used in S/MIME, IP Security, SSL/TLS
X.509 Formats

(a) X.509 Certificate

(b) Certificate Revocation List

Signature
- algorithm parameters
- encrypted

Issuer Unique Identifier

Signature algorithm identifier
- algorithm parameters

Subject Unique Identifier

Period of validity
-not before
-not after

Signature
- algorithm parameters

Revoked certificate
- user certificate serial number
- revocation date

Version

Subject's public key info
- algorithm parameters
- key

Version 1

Version 2

Version 3
X.509 Authentication procedures

(a) One-way authentication

1. A\{t_A, r_A, B, sgnData, E_{KUb} [K_{ab}]\}

(b) Two-way authentication

1. A\{t_A, r_A, B, sgnData, E_{KUb} [K_{ab}]\}

2. B\{t_B, r_B, A, r_A, sgnData, E_{KUa} [K_{pa}]\}

(c) Three-way authentication

1. A\{t_A, r_A, B, sgnData, E_{KUb} [K_{ab}]\}

2. B\{t_B, r_B, A, r_B, sgnData, E_{KUa} [K_{pa}]\}

3. A\{r_B\}
Summary

- Discussed Key Distribution for secret keys and public keys
- Kerberos solves key distribution and Authentication
- Certificate authorities solve this for PKI