Distributed Authentication in Kerberos Using Public Key Cryptography

Marvin A. Sirbu       John Chung-I Chuang
Carnegie Mellon University

Symposium on Network and Distributed System Security
February 10-11 1997
Outline

• Public Key Cryptography for Kerberos
• Alternative Approaches
• The PKDA Protocol
• Migration to PKDA
• Implementation and Progress
Why Public Key in Kerberos

- Reduce/eliminate sensitive information at KDC
- Distribute functions of TGS for scalability
  - on-line banking with millions of consumers in a single trust domain
PKDA

- Public-key based Kerberos for Distributed Authentication
- Public-key cryptography built upon certificate infrastructure
- Mutual authentication and key exchange
- Data integrity and privacy protection
PKDA

• Extension to Kerberos V5 Authentication Framework (RFC 1510)
• Builds upon X.509, PKCS standards
• Supports Rights Delegation
• Enhancement to User Privacy Protection over Kerberos V5
Alternative Approaches

- Secure Socket Layer (SSL 3.0)
- Public Key Cryptography for Initial Authentication in Kerberos (pk-init)
- PKDA
SSL 3.0

- Supports TCP but not UDP
- Client and server exchange certificates
- Both parties cache session key and session_id locally
- Reuse session key by resending session_id
- Choice of cryptographic algorithms
- Certificate revocation checking unspecified
pk-init

- Supports both TCP and UDP
- No client keys at KDC; server keys still stored
- TGS interaction required for every session ticket
- Session tickets reusable during lifetime
PKDA

• Supports both TCP and UDP
• Client and server exchange certificates
• Session ticket and key exchanged directly - no TGS involved
• Ticket reusable for subsequent interactions
• Certificate revocation checking unspecified
PKDA vs. SSL 3.0

• Protocol layer
• End-to-end message encryption
• Ticket reusability/session caching
• Rights delegation in PKDA
PKDA vs. pk-init

- PKDA is fully distributed; no centralized KDC/TGS
- PKDA enhances privacy of principals
- PKDA requires code modifications to clients and servers; pk-init requires code modifications for clients and KDC
Notation

C  Client
S  Server
K_r  random one-time symmetric key
K_{c,s}  symmetric key shared by C and S
\{M\}K_{c,s}  message encrypted using key K_{c,s}
\{M\}P_s  message encrypted using public key of S
\{M\}P_c^{-1}  message signed using private key of C
T_s#  time-stamps
T_{auth}  Initial Authentication Time
T_{c,s}  Ticket for session between S and C
Traditional Kerberos

1. **AS_REQ**: C, TGS, Ts1
2. **AS_REP**: \{K_{c,tgs}, TGS, Ts1\}K_c, T_{c,tgs}
3. **TGS_REQ**: C, S, Ts2, T_{c,tgs}, \{auth\}K_{c,tgs}
4. **TGS_REP**: C, \{K_{c,s}, S, Ts2\}K_{c,tgs}, T_{c,s}
5. **AP_REQ**: T_{c,s}, \{C, Ts3\}K_{c,s}

where

\[ T_{c,tgs} = TGS, \{K_{c,tgs}, C, T_{auth}\}K_{tgs} \]

is the ticket granting ticket (TGT);

\[ T_{c,s} = S, \{K_{c,s}, C, T_{auth}\}K_{s,tgs} \]

is the service ticket.
PKDA Protocol

1. SCERT_REQ: S
2. SCERT_REP: s-cert
3. PKTGS_REQ:
   S, \{C,c-cert,\{S, P_s, K_r, T_{auth}\}P_c^{-1}\}P_s
4. PKTGS_REP: \{C,S,K_{c,s},T_{auth}\}K_r, T_{c,s}
5. AP_REQ: T_{c,s}, \{C,TS1\}K_{c,s}

where ticket

\[ T_{c,s} = S,\{K_{c,s},C,T_{auth}\}K_s \]
Rights Delegation

1. SCERT_REQ: S
2. SCERT_REP: s-cert
3. PKTGS_REQ:
   S, {C,c-cert,{S, P_s, K_r,T_auth}P_c^{-1} }P_s
   with ‘PROXIABLE’ flag set
4. PKTGS_REP: {C,S,K_{c,s},T_auth}K_r, T_{c,s}
5. KRB_CRED: {T_{c,s}, {C,Ts1}K_{c,s},K_{proxy}}K_{c,g}
6. AP_REQ: T_{c,s}, {C,Ts1}K_{c,s}

where ticket is proxiable:

T_{c,s} = S,\{K_{c,s},C,T_{auth}\}K_s

and K_{c,g} is previously established symmetric key between C and G.
Accommodating Conventional Application Servers

If Server does not understand PKDA:

• Obtain conventional TGT from PKDA-enabled TGS
• Use TGT to request a service ticket for server S
• Capture all benefits of pk-init without need for server code change
Obtaining Session Tickets from a PDKA-Enabled TGS

0. SCERT_REQ: TGS
0. SCERT_REP: tgs-cert
1. PKTGS_REQ:
   TGS, \{C, ccert, \{TGS, P_{tgs}, T_{auth}, K_r\}P_c^{-1}\}P_{tgs}
2. PKTGS_REP: \{C, TGS, K_{c,tgs}, T_{auth}\}K_r, T_{c,tgs}
3. TGS_REQ: C, S, Ts1, T_{c,tgs}, \{auth\}K_{c,tgs}
4. TGS_REP: C, \{K_{cs}, S, Ts1\}K_{c,tgs}, T_{c,s}
5. AP_REQ: T_{c,s}, \{C, Ts2\}K_{c,s}

where
\[ T_{c,tgs} = TGS, \{K_{c,tgs}, C, T_{auth}\}K_{tgs} \]
is the ticket granting ticket;
\[ T_{c,s} = S, \{K_{c,s}, C, T_{auth}\}K_{s,tgs} \]
is the service ticket.
Implementation of PKDA

• Protocol Verification
• Working Implementation for CMU’s NetBill electronic payment system
  – Use DCE RPCs: enhancements to IDL compiler automatically adds PKDA RPCs to interfaces
• Protocol Specification in Internet Draft