Lecture 5:
Protocols - Authentication and Key Exchange*

CS 392/681: Computer Security
Fall 2006

Nitesh Saxena

*Adopted from a previous lecture by Gene Tsudik

What are your concerns?

- Class timing?
  - 392: until 8:50, 681: until 8:15
    - I cover material until 8:15 (plus some delay)
    - After that I discuss some miscellaneous stuff; like some questions that I have to recap the lecture
- Programming (in C)?
- Homeworks are too tough, there are too many of them?
- Lectures?
- Book?
- GRADES?
Course Admin

- HW#4 due at midnight coming Wednesday (10/11)
- 681 Project proposals:
  - Some of you would have got some comments
    - Do consider them
  - Work on the projects full throttle; keep me posted
- HW#5 will be posted by coming Monday
- HW#2 and #3 are being graded
  - Thanks for your patience!
- HW#3 solution will be posted soon
  - Thanks for your patience!

Exams – you love them the most 😊

- Midterm 1 is in-class on 10/19/06 (Thursday)
  - Includes lecture material until 10/12/06 (i.e., next class)
  - Closed book, closed notes
  - Emphasis, of course, is not on memorizing!
  - How long?
- Midterm 2 (tentatively) is on 11/30/06 (Thursday)
  - Covers everything, but focus will be on material covered after Midterm 1
  - No class a week prior to this -- thanksgiving
- No final
- 681 project reports and presentations due during the finals week: 12/18 – 12/22
Instructions

- HW submissions
  - Name your files “Lastname_Firstname_HW#”
  - Submit it on MyPoly
  - Also, send me an email with subject “392, 681 HW #”, containing a copy of your hw
    - For record-keeping
- Check the course website regularly
  - I am posting lectures/homeworks there
- Check your poly email regularly
  - I am sending out announcements there
    - e.g., when I post homeworks
- NO EXCUSES for not following instructions

Outline of Today’s lecture

- Today we try to put everything together
  - Encryption (public-key/private-key)
  - MACs
  - Signing
  - Key-Distribution
- Secure protocols (for secure communication)
  - Authentication
    - We studied it somewhat while talking about key distribution
  - (Authenticated-) Key Exchange
- Designing secure protocols is hard - we’ll only be able to learn the basics today
- We’ll use the white-board extensively today - be prepared to take notes
What are your concerns?

- Class timing?
  - 392: until 8:50, 681: until 8:15
    - I cover material until 8:15
    - After that I discuss some miscellaneous stuff; like some questions that I have to recap the lecture

- Programming?
- Homeworks are too tough, there are too many of them?
- Lectures?
- Book is of no use?
- GRADES?

Protocol

- A protocol is a set of rules using which *two or more* entities exchange messages
- It consists of *messages* and *rounds*
Messages and Rounds

- A message is a unit of information send from one entity to other
- A round is a basic unit of protocol time
  1. Wake up because of
     1. Alarm (or clock)
     2. Initial start or
     3. Receipt of message(s) from other(s)
  2. Compute something
  3. Send message(s) to other(s)
  4. Repeat 2-3 if needed
  5. Wait for message(s) or clock

Types of Adversaries

- Passive
  - Eavesdrop, delay, drop, replay messages

- Active
  - Eavesdrop, delay, drop, replay and modify messages
Properties of a Secure Protocol

- **Correctness**
  - If entities taking part in the protocol behave honestly, the protocol achieves its desired goal

- **Security**
  - No adversary can defeat the goal of the protocol (with a significantly high probability)
    - Adversary could be passive or active, depending upon the application (we consider the latter)
    - We won't consider denial-of-service attacks

Authentication

- **Origin Authentication**
  - Verification of the origin/source of the message

- **Entity Authentication**
  - Verification of the identity of the sender of the message

- We focus on the latter in a 2-party setting
- Authentication can be unilateral or mutual
Basis for Authentication

- Something you know
  - e.g., a PIN, a password

- Something you have
  - e.g., an access key or a card; a certificate; a smart card

- Something you are
  - e.g., biometric (such as fingerprint)

Weak Authentication

- PINs, Passwords provide weak authentication
  - Security is based upon how hard the pin/password is to guess
  - Usually, the passwords are short and weak
  - Widely used in practice
    - Unix, kerberos, web emails,......
  - Protocol (A authenticates B using a password P, that A shares with B)

1. A → B: Hi, this is A!
2. B → A: r (random challenge)
3. A → B: H(p,r)

Problem?
Strong Authentication

- An entity authenticates to the other by proving the knowledge of a secret associated with that entity, without revealing anything meaningful about the secret itself.
- Can be achieved through:
  - Private/Public Key Encryption
  - MAC
  - Digital signatures
- Strong because the security reduces to the security of the underlying cryptographic mechanism, which is assumed to be hard to break.
- Our focus in the rest of the lecture.
- We’ll study both private-key and public-key based authentication.

Symm. Encryption-based authentication

- Uses encryption to authenticate Alice to Bob (assuming Alice-Bob have established a shared secret K, say through Kerberos).

**A auth B**

1. A → B: Hi Bob, this is Alice!
2. B → A: r (a challenge)
3. A → B: Enc\(_K\)(r, B) (response)

Why not simply send Enc\(_K\)(r) in msg 3?
What about sending Enc\(_K\)(r, A, B) instead of Enc\(_K\)(r, B) in msg 3?
Security of the previous protocol

- An attacker needs to come up with a valid response
  - Impossible if encryption is secure
- \( r \) must be un-predictable (by the attacker) and must not be re-used by Bob
  - Why?

Replay attack (predictable \( 'r' \))

1. \( A \rightarrow B: \text{Hi Bob, this is Alice!} \)
2. \( E \rightarrow A: \text{r} \)
3. \( A \rightarrow E: \text{M=Enc}_K(r,B) \)
4. \( E \rightarrow B: \text{Hi Bob, this is Alice!} \)
5. \( B \rightarrow E: \text{r} \)
6. \( E \rightarrow B: \text{M} \)
Freshness

- Assurance that message has not been used previously and originated within an acceptably recent timeframe
- Two methods:
  - Nonce (Number used once)
  - Timestamps

Nonces

- One-time random number
- We depended on B to make sure r is a good nonce
- Main property is “one-time-ness”, so could use even a counter (but must keep state)
  - Starting value must still be random and unpredictable
- Choose nonces “randomly” from a large space (such as $2^{128}$) to avoid re-use and for unpredictability – good RNG
**Timestamps**

- Inclusion of date/time-stamp in the message allows recipient to check it for freshness
  - Need to be protected with cryptographic means
- A $\rightarrow$ B: $\text{Enc}_K(T, B)$
  - Results in fewer messages
- But, requires synchronized clocks
  - hard to achieve in practice!

**Encryption-based Mutual Authentication (1)**

- Run two copies of the uni-lateral authentication protocol $\rightarrow$ 6 rounds
- We can piggyback common flows

1. A $\rightarrow$ B: A, rA
2. B $\rightarrow$ A: $\text{Enc}_K(rA, rB, A)$
3. A $\rightarrow$ B: $\text{Enc}_K(rB, B)$
Encryption-based Mutual Authentication (2)

1. A → B: A, $\text{Enc}_K(T, B)$
2. B → A: $\text{Enc}_K(T+1, A)$

MAC-based Authentication

1. A → B: A, $r_A$
2. B → A: $r_B$, $\text{HMAC}_K(r_A, r_B, A)$
3. A → B: $\text{HMAC}_K(r_B, B)$

- Faster than enc-based protocols (computationally)
Session Key Exchange with KDC - Needham-Schroeder Protocol (I’ll post a corrected version later!)

A -> KDC  ID_A || ID_B || N_1
(Hello, I am Alice, I want to talk to Bob, I need a session Key and here is a random nonce identifying this request)

KDC -> A  E_{K_A}( K || ID_B || N_1 || E_{K_B}(K || ID_A))
Encrypted(Here is a key, for you to talk to Bob as per your request N_1 and also an envelope to Bob containing the same key)

A -> B  E_{K_B}(K || ID_A)
(I would like to talk using key in envelope sent by KDC)

B -> A  E_{K}(N_2)
(OK Alice, But can you prove to me that you are indeed Alice and know the key?)

A -> B  E_{K}(f(N_2))  (Sure I can!)

Last two steps - challenge-response. Commonly used to thwart replay attack.

Session Key Exchange with KDC - Needham-Schroeder Protocol (corrected version 1)

A -> KDC  ID_A || ID_B || N_1
(Hello, I am Alice, I want to talk to Bob, I need a session Key and here is a random nonce identifying this request)

KDC -> A  E_{K_A}( K || ID_B || N_1 || E_{K_B}(K || ID_A))
Encrypted(Here is a key, for you to talk to Bob as per your request N_1 and also an envelope to Bob containing the same key)

A -> B  E_{S}(TS, B), E_{K_B}(K || ID_A)
(I would like to talk using key in envelope sent by KDC; here is an authenticator)

B -> A  E_{K}(TS+1, A)
(OK Alice, here is my authenticator)
Session Key Exchange with KDC - Needham-Schroeder Protocol (corrected version 2)

- A -> KDC  \( ID_A \ || \ ID_B \ || \ N_1 \)
  (Hello, I am Alice, I want to talk to Bob, I need a session Key and here is a random nonce identifying this request)

- KDC -> A  \( E_{K_A}(K \ || \ ID_B \ || \ N_1 \ || \ E_{K_B}(K \ || \ ID_A)) \)
  Encrypted(Here is a key, for you to talk to Bob as per your request \( N_1 \) and also an envelope to Bob containing the same key)

- A -> B  \( N_2, E_{K_B}(K \ || \ ID_A) \)
  (I would like to talk using key in envelope sent by KDC; here is a nonce)

- B -> A  \( E_K(N_3, N_2, A) \)
  (OK Alice, But can you prove to me that you are indeed Alice and know the key?)

- A -> B  \( E_K(N_3, B) \)  (Sure I can!)

Version 4 summary

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

1. \( C \rightarrow TGS: ID_c \ || \ ID_{TGS} \ || \ TS_1 \)
2. \( TGS \rightarrow C: \ E_{K_{TGS}} [K_c \ || \ ID_c \ || \ TS_1 \ || \ Lifetime_c] \)

Ticket_{TS} = \( E_{K_{TGS}} [K_c \ || \ ID_c \ || \ AD_c \ || \ ID_{TGS} \ || \ TS_2 \ || \ Lifetime_c] \)

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

3. \( C \rightarrow TGS: ID_c \ || \ Ticket_{TS} \ || \ Authenticator_c \)
4. \( TGS \rightarrow C: \ E_{K_{TGS}} [K_c \ || \ ID_c \ || \ TS_1 \ || \ Ticket_{TGS} \ || \ Lifetime_c] \)

Ticket_{TGS} = \( E_{K_{TGS}} [K_c \ || \ ID_c \ || \ AD_c \ || \ ID_{TGS} \ || \ TS_2 \ || \ Lifetime_c] \)

Ticket_{TGS} = \( E_{K_{TGS}} [K_c \ || \ ID_c \ || \ AD_c \ || \ ID_{TGS} \ || \ TS_1 \ || \ Lifetime_c] \)

Authenticator_{TGS} = \( E_{K_{TGS}} [ID_c \ || \ AD_c \ || \ TS_1] \)

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

5. \( C \rightarrow K: \ Ticket_s \ | \ Authenticator_s \)
6. \( K \rightarrow C: \ E_{K_{Auth}} [TS_5 + 1] \)  (for mutual authentication)

Ticket_{s} = \( E_{K_{c}} [K_{c} \ || \ AD_{c} \ || \ ID_{s} \ || \ TS_{s} \ || \ Lifetime_{s}] \)

Authenticator_{s} = \( E_{K_{c}} [ID_{s} \ || \ AD_{s} \ || \ TS_{s}] \)
Public-key based authentication (Needham-Shroeder (NS) pk-based)

- Assuming public keys are distributed through CA(s)

1. \( A \to B: \) \( \text{Enc}_{pkb}(rA, A) \)
2. \( B \to A: \) \( \text{Enc}_{pka}(rA, rB) \)
3. \( A \to B: \) \( \text{Enc}_{pkb}(rB) \)

Attack and fix on PK-based NS protocol

- Attack:

- Fix:

1. \( A \to B: \) \( \text{Enc}_{pkb}(rA, A) \)
2. \( B \to A: \) \( \text{Enc}_{pka}(rA, rB, B) \)
3. \( A \to B: \) \( \text{Enc}_{pkb}(rB) \)
**Signature-based authentication**
(assuming public keys are distributed through CA)

**A auth B**
- A → B:  
  Hi Bob, this is Alice!
- B → A:  
  r (a challenge)
- A → B:  
  Sig_{SKa}(r,B) (response)

**A auth B, B auth A** (run two copies; piggyback common flows)
- A → B:  
  A, rA (could sign this too)
- B → A:  
  rB, Sig_{SKb}(rA, rB, A)
- A → B:  
  Sig_{SKa}(rB, B)

---

**Authenticated Key Exchange**

- Public-key operations are costly

- Why not
  1. use public-key mutual authentication protocols to exchange a symmetric key
  2. use this symmetric key with a symmetric encryption to secure subsequent communication
1. A \rightarrow B: A, r_A, \text{Enc}_{PK_b}(K) (\text{could sign this too})
2. B \rightarrow A: r_B, \text{Sig}_{SK_b}(r_A, r_B, A)
3. A \rightarrow B: \text{Sig}_{SK_a}(r_B, B)
4. A and B output K as the authenticated key

- Such a protocol can be instantiated using RSA encryption/signing
  - The way SSL/SSH establishes key
    - But, generally only the server authenticates the client, not vice versa

X.509 Authentication – One-way.

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

- Establishes the following
  - Identity of A and message was generated by A
  - Message was intended for B
  - Integrity and originality of message.
X.509 Authentication – Two-way.

1. A(t_A, r_A, B, sgnData, E_{K_{Ub}} [K_{ab}])
2. B(t_B, r_B, A, r_A, sgnData, E_{K_{Ua}} [K_{ba}])

- One-Way plus the above which establishes the following:
  - Identity of B and message was generated by B
  - Message was intended for A
  - Integrity and originality of message.

X.509 Authentication – Three-way.

1. A(t_A, r_A, B, sgnData, E_{K_{Ub}} [K_{ab}])
2. B(t_B, r_B, A, r_A, sgnData, E_{K_{Ua}} [K_{ba}])
3. A(r_B)
**Diffie-Hellman (DH) Key Exchange**

1. \( A \rightarrow B: \quad K_a = g^a \mod p \)
2. \( B \rightarrow A: \quad K_b = g^b \mod p \)
3. \( A \) outputs \( K_{ab} = K_b^a \)
4. \( B \) outputs \( K_{ba} = K_a^b \)

- Note \( K_{ab} = K_{ba} = g^{ab} \mod p \)

**Security of DH key exchange**

- No authentication of either party
- Secure only against a passive adversary
  - Under the computational Diffie-Hellman assumption
    - Given \( (g, g^a, g^b) \), hard to compute \( g^{ab} \)
  - Not secure against an active attacker
    - Man-in-the-middle attack...
Authenticated DH Key Exchange  
aka: Station-to-Station Protocol

1. A → B: \( K_a = g^a \mod p \)
2. B → A: \( \text{Cert}_b, K_b = g^b \mod p \)  
   \( \text{Sig}_{SK_b}(K_a, K_b) \)
3. A → B: \( \text{Cert}_a, \text{Sig}_{SK_a}(K_b, K_a) \)
4. A outputs \( K_{ab} = K_b^a \)
5. B outputs \( K_{ba} = K_a^b \)

Summary

- Designing secure protocols is not easy
  - Becomes harder in a concurrent setting, where there are multiple parties, executing multiple instances of the protocols simultaneously
  - Become even harder as the number of parties increase; n-party or group setting
  - Use the protocols that are well-studied and standardized

- While designing a protocol, consider
  - Reflection attacks
  - Replay attacks
  - Eliminating any symmetry in the messages
Further Reading

- Unix password security
  http://www.ja.net/CERT/Belgers/UNIX-password-security.html
- MIT Kerberos site:
  http://web.mit.edu/kerberos/www/
- Kerberos RFC: RFC-1510