CS458/CS558 Introduction to Computer Security

Course Project

Presentation Project

- Done individually
- Present 2 paper in April/end of March (around 25min each)
  - present the papers in the same or different day
  - Make powerpoint slides
    - Motivation
      E.g. Why role-based access control?
    - Background
      E.g. Syntax of xml
    - Technical details
      Use examples/pictures to illustrate technical details.
    - Related work
  - Present clearly and slowly
- Slides Submission deadline: May 3rd (Thurs)

Course Project: Presentation

- Topics
  - Blockchain
  - SGX-based security
  - Web security

Programming Project

- Done by a group of two students (10' extra credits if you choose to do the project alone)
- No presentation is required
- Submit the code through mycourses
  - deadline: 11:59pm May 3rd
- You can use existing implementations of RSA and digital signature, if necessary.
- You are not required to implement a graphical user interface.

Secure Electronic Voting

- This project implements a secure virtual election booth.
- The secure virtual election booth must meet the following requirements:
  - No one can vote more than once.
  - No one can determine the candidate for whom anyone else voted.

Secure Electronic Voting

- Assume that there are 3 voters (Alice, Bob, John) and 2 candidates (Tim, Linda).
- Each voter has a voter registration number. The voter registration number is given below, which is stored in a file **voterinfo**

Alice	**112550000**
Bob	**113880000**
John	**114660000**

Secure Electronic Voting

- The voter connects to the Voting Facility (VF) to vote.
- Assume that all voters have public keys of VF and VF has public keys of all voters.
  - You can manually generate public keys for VF and all voters and store them in files.
- The client is invoked (by the voter) as:
  voter-cli *<VF's domain name> <VF's port number> (C/C++)*
  java VoterCli <VF's domain> <VF's port> (Java)
- The VF server is invoked as:
  vf *<VF server's port number> (C/C++)*
  java Vf <VF server's port number> (Java)

- **CS558:** The VF server must be a concurrent server.
- **CS458:** The VF server can be ab iterative server.

Secure Electronic Voting

Detailed steps:

1. The voter invokes ***voter-cli*** to connect to the VF server

2. Upon connection, the voter is prompted to enter his/her name and voter registration number ***vnumber***

3. After the voter enters ***vnumber***, voter-cli sends ***E(pub(VF), name||vnumber)||DS(name)*** to the VF server

Secure Electronic Voting

4. VF checks whether the name and vnumber received match the information in file ***voterinfo***.
   - If not, VF sends 0 to voter-cli. voter-cli then prints "Invalid name or registration number" and terminates the connection.
   - Otherwise, VF sends 1 to voter-cli. voter-cli prints user's name and prompts the user to select an action:

Welcome, <user's name>
Main Menu
Please enter a number (1-4)
1. Vote
2. My vote history
3. Election result
4. Quit

Secure Electronic Voting

- 5. If the voter enters "1", then voter-cli sends 1 to the VF server. The VF checks whether the voter has voted (based on file ***history*** describe in Step 6).
  - If so, VF sends 0 to voter-cli, and voter-cli prints "you have already voted" and displays the Main menu.
  - Otherwise, VF sends 1 to voter-cli and voter-cli displays the following:

Please enter a number (1-2)
1. Tim
2. Linda

Secure Electronic Voting

6. After the voter enters the number, the client sends **E(pub(VF), number)** to VF. VF then updates the result in file ***result,*** which has the following Format (initially the total number is 0):

Tim <the total number of votes>
Linda <the total number of votes>

VF adds the date and time when the voter votes to a file ***history*** that has the following format (if ***history*** does not exist, then create the file):

<registration number> <date and time when the voter votes>

If the user is not the last user who voted, then voter-cli displays Main Menu in Step 4.

Otherwise VF server prints the results using the following format:

<Candidate's name> Win
Tim <the total number of votes>
Linda <the total number of votes>

Vote-cli then displays Main Menu.

Secure Electronic Voting

7. If the voter enters "2", VF retrieves the corresponding entry in file *history* and sends the entry to voter-cli. *voter-cli* then displays the entry to the user. Go to Main Menu in step 4

8. If the user enters "3", then VF checks whether all users have voted. If not, then VF sends 0 to voter-cli and voter-cli displays "the result is not available".

Otherwise, VF sends *voter-cli* the results. Voter-cli then displays the results using the following format:

<Candidate's name> Win
Tim <the total number of votes>
Linda <the total number of votes>

9. If the user enters "4", then voter-cli terminates.

Grading Guideline (CS558)

- Readme: 5'
- Makefile: 5'
- Encryption, decryption, digital signature: 25'
- Concurrent server: 10'
- Other functionality: 55'

Grading Guideline (CS458)

- Readme: 5'
- Makefile: 5'
- Encryption, decryption, digital signature: 25'
- Other functionality: 65'

Submission Guideline

- Submit source code, public and private keys generated, a readme, and a Makefile electronically.
- README (text file)
  - Name and email address of group members.
  - The programming language you use (C/C++/C#/Java)
  - Platform (Bingsuns/Linux/Windows)
  - How to execute your program.
  - Code for performing encryption/decryption
  - (Optional) anything special about your submission

Submission Guideline

- Place all your files under one directory with a unique name (such as proj-[userid] for the project).
- Tar the contents of this directory.
  tar –cvf proj-[userid].tar proj-[userid]
- Use the mycourses to upload the tared file

Systems Projects

- Done by a group of 2 students (10 points extra credits if done alone)
- Give 20-25 min presentation and show demo in the class on May 3rd
  - Design
  - Implementation
- Submit the code and slides on May 3rd

Systems Projects

1. Buffer Overflow Attack (language: C)
   Work through the shell example given in
   http://insecure.org/stf/smashstack.html

2. Kernel Rootkit II (lauguage: C)
   Design and implement a linux kernel rootkit that modifies the system call table to hide the file you create (ls).

Systems Projects

3. Virus (language: C)
   Design and implement a virus that can infect all executable files under a specific directory.

   If a program is infected with the virus, the program can still be executed.

   When the infected program executes, the virus will execute first, and then the program.

Systems Projects

4. Online secure checkout system
   Two options:

   (1) choose a secure checkout system and understand how the system is implemented

   (2) use paypal express checkout integration (URL is given below) to develop an online secure checkout system.
   https://developer.paypal.com/docs/integration/direct/express-checkout/integration-jsv4/

Systems Projects

5. Blockchain platforms

   You will use and compare two blockchain platforms at your choice.

   You will need to demonstrate how to use those blockchain systems and give a presentation to compare the two systems.

Course Projects

- If you choose to do a presentation project or a systems project,
  - Please email me 3 project topics you are interested in by Feb. 20th, indicating your first, second, and third choices.
- If you choose to do a programming project
  - Please do NOT email me the topic

Chapter 10
Key Management

Key Management

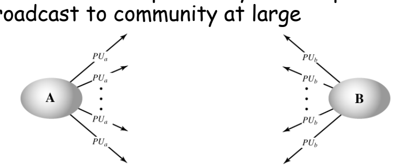
- Public-key encryption helps address key distribution problems:
  - Use of public-key encryption to distribute secret keys

Distribution of Public Keys

- Several techniques have been proposed for the distribution of public keys:
  - Public announcement
  - Public-key authority
  - Public-key certificates

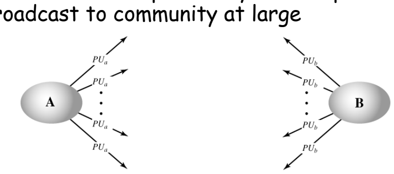
Public Announcement

- Users distribute public keys to recipients or broadcast to community at large



Public Announcement

- Users distribute public keys to recipients or broadcast to community at large



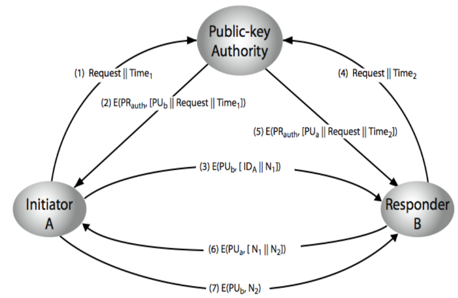
- Major weakness: forgery
  - Anyone can create a key claiming to be someone else and broadcast it
  - Until forgery is discovered can masquerade as claimed user

Public-Key Authority

- A central authority maintains a dynamic directory of public keys of all participants {name, public-key}.
- Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure communication.
- Requires users to know public key for the directory. Only the authority knows the corresponding private key.
- Users interact with directory to obtain any desired public key securely.

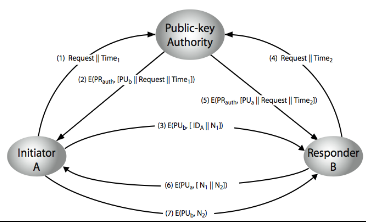
Public-Key Authority

1. A sends a timestamped mesg to the public-key authority containing a request for the public key of B.



Public-Key Authority

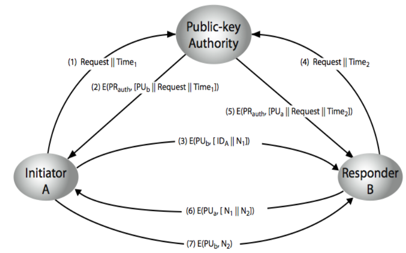
2. The authority responds with a mesg that is encrypted using the authority's private key.
   - B's public key
   - The original request: match with the request
   - The original timestamp: A can determine that this is not an old message from the authority.



Public-Key Authority

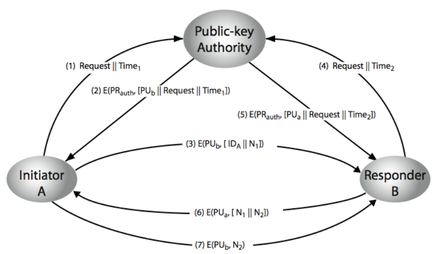
3. A stores B's public key and uses it to encrypt a mesg to B containing an identifier of A and a nonce N1.

4.5. B retrieves A' public key from the authority in the same manner as A retrieved B's public key



Public-Key Authority

6. B sends a mesg to A encrypted using A's public key that contains A's nonce and a nonce generated by B.
7. A returns N2 encrypted using B's public key, to ensure B that its correspondent is A.



Drawback: Public-Key Authority

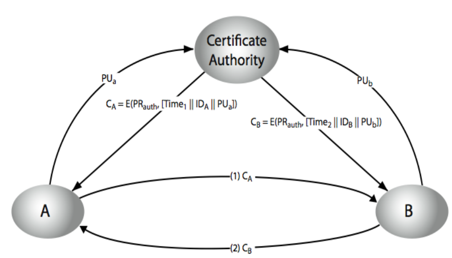
- Drawback of public-key authority: the public-key authority is a bottleneck because a user must appeal to the authority for a public key for every other user that it wishes to contact.

Public-Key Certificates

- Certificates allow key exchange without contacting a public-key authority
- A certificate consists of a public key plus an identifier of the key owner, with the whole block signed by a trusted third party (certificate authority).
- A user can present his/her public key to the authority in a secure manner and obtain a certificate.
- The user then publishes the certificate.
- Other participant can verify that the certificate was created by the authority.

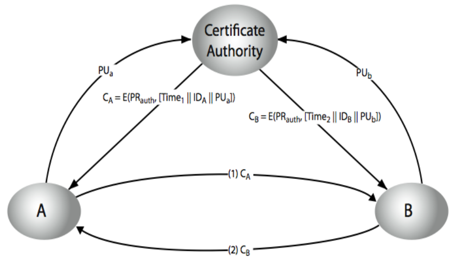
Public-Key Certificates

1. Participant A applies to the certificate authority, supplying a public key and requesting a certificate.
   - Application must be in person or by some form of secure authenticated communication.



Public-Key Certificates

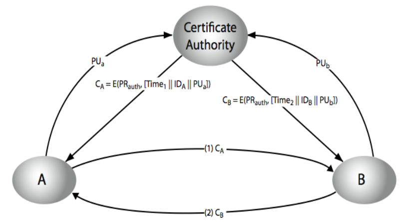
2. The authority provides the certificate of the form CA = E(PRauth, [Time1||IDa||PUa]), where PRauth is authority's private key and Time1 is a timestamp.



Public-Key Certificates

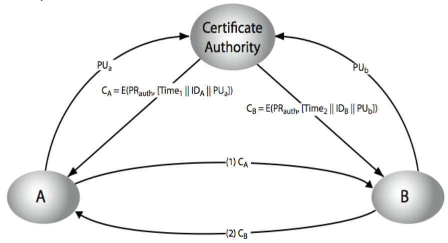
3. A may then pass this certificate on to any other participant, who reads and verifies that the certificate comes from the certificate authority (by decrypting the certificate using authority's public key):

D(PUauth, CA) = D(PUauth, E(PRauth, [T||IDA||PUa]))= (T||IDA||PUa)



Public-Key Certificates

- Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
- Only the certificate authority can create and update certificate.



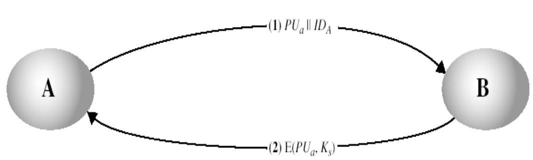
Distribution of Secret Keys Using Public-Key Cryptography

- Use previous methods to obtain public-key
- Can use for secrecy or authentication
- But public-key algorithms are slow
- So usually want to use symmetric-key encryption to protect message contents - need to distribute the session key
- Public-key cryptography can be used to distribute the session keys.

Simple Secret Key Distribution

- Proposed by Merkle in 1979

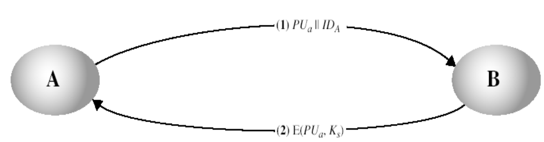
1. A generates a new temporary public key pair
2. A sends B the public key and their identity
3. B generates a session key K, sends it to A encrypted using the supplied public key
4. A decrypts the session key



Simple Secret Key Distribution

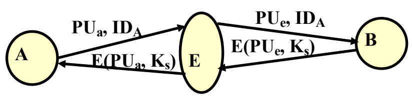
- Proposed by Merkle in 1979

5. A discards the public key pair and B discards A' public key
6. At the completion of the exchange, A and B discard the session key



Simple Secret Key Distribution: Attack

- Problem: man-in-the-middle attack – an adversary can intercept messages and then replay the intercepted message or send another message.



- A transmits a message intended for B consisting of PU_a and A's identifier ID_A
- E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e||ID_A$ to B
- B generates a secret key K_s and transmits $E(PU_e,K_s)$
- E intercepts the message, and learns Ks by computing $D(PR_e, E(PU_e, K_s))$
- E transmits $E(PU_a,K_s)$ to A
- Both A and B know K_s and are unaware that K_s has been

Chapter 10.2 Diffie-Hellman Key Exchange

44

Diffie-Hellman Key Exchange

- The first public-key algorithm proposed by Diffie & Hellman in 1976
- The purpose is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages.
- Used in a number of commercial products

45

Diffie-Hellman Key Exchange

- An integer a is a primitive root of a prime number q if $a \bmod q$, $a^2 \bmod q$, …, $a^{q-1} \bmod q$ are distinct and consist of the integers from 1 through q-1 in some permutation.
- Is 2 a primitive root of 5?

46

Diffie-Hellman Key Exchange

- An integer a is a primitive root of a prime number q if $a \bmod q$, $a^2 \bmod q$, …, $a^{q-1} \bmod q$ are distinct and consist of the integers from 1 through q-1 in some permutation.
- Is 2 a primitive root of 5?
  Yes

47

Diffie-Hellman Key Exchange

- An integer a is a primitive root of a prime number q if $a \bmod q$, $a^2 \bmod q$, …, $a^{q-1} \bmod q$ are distinct and consist of the integers from 1 through q-1 in some permutation.
- Agree on two numbers:
  - A prime number q
  - An integer a that is the primitive root of q
- Each user generates his/her key
  - Chooses a private key (number): $x < q$
  - Computes their public key: $y = a^x \bmod q$
- Each user keeps the x value private and makes the y value available publicly

48

Diffie-Hellman Key Exchange

Property of modular arithmetic

[(a1 mod n) *...* (am mod n)] mod n = (a1*...*am) mod n

- Shared session key for users A & B is K_{AB}:

$K_{AB} = a^{x_A . x_B} \bmod q$

$= y_A^{x_B} \bmod q$ (which B can compute)

$= y_B^{x_A} \bmod q$ (which A can compute)

- K_{AB} is used as session key in private-key encryption scheme between A and B
- Question: How to prove that $y_A^{x_B} \bmod q = y_B^{x_A} \bmod q$

Diffie-Hellman Key Exchange

Property of modular arithmetic

[(a1 mod n) *...* (am mod n)] mod n = (a1*...*am) mod n

- Question: How to prove that $y_A^{x_B} \bmod q = y_B^{x_A} \bmod q$

$y_A^{x_B} \bmod q$

$= (a^{x_A} \bmod q)^{x_B} \bmod q$

$= [\underbrace{(a^{x_A} \bmod q)* \ldots *(a^{x_A} \bmod q)}_{x_B}] \bmod q$

$= a^{x_A x_B} \bmod q = a^{x_B x_A} \bmod q$

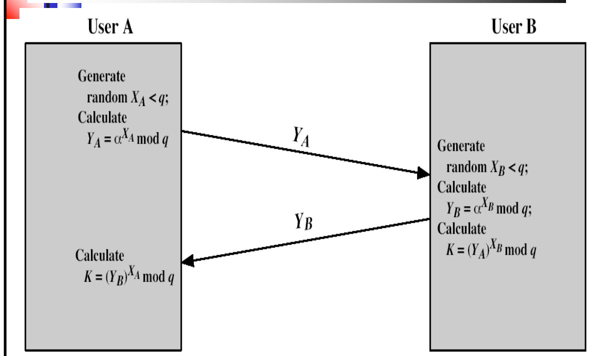
$= [\underbrace{(a^{x_B} \bmod q)* \ldots *(a^{x_B} \bmod q)}_{x_A}] \bmod q$

$= y_B^{x_A} \bmod q$

Diffie-Hellman Example

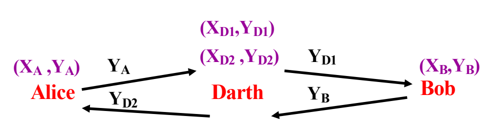
- Users A & B who wish to swap keys:
- Agree on prime $q=353$ and $a=3$
- Select random private keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- Compute respective public keys:
  - $y_A=3^{97} \bmod 353 = 40$ (A)
  - $y_B=3^{233} \bmod 353 = 248$ (B)
- Compute shared session key as:
  - $K_{AB}= y_B^{x_A} \bmod 353 = 248^{97} \bmod 353 = 160$ (A)
  - $K_{AB}= y_A^{x_B} \bmod 353 = 40^{233} \bmod 353 = 160$ (B)

Man-in-the-Middle Attack



Man-in-the-Middle Attack

- Man-in-the-Middle-Attack



$K2 = (Y_{D2})^{X_A} \bmod q$

$K2 = (Y_A)^{X_{D2}} \bmod q$

$K1 = (Y_B)^{X_{D1}} \bmod q.$

$K1 = (Y_{D1})^{XB} \bmod q$

- Now, Alice and Bob think that they share a secret key, but instead Bob and Darth share secret key K1. Darth and Alice share secret key K2.