Programming Projects (C/C++/C#/Java)
No presentation is required
Submission deadline: 11:59pm May 6th
(source code + readme)

The programming project is done by a group of 2 students. You can use existing implementations of RSA and digital signature (e.g. provided in java.security, openssl, etc), if necessary. You are not required to implement a graphical user interface. If you work on the programming project by yourself, you will get 15 points extra credits.

This project implements a secure virtual election booth. The implementation should provide a secure way for people to vote online. 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.

Assume that there are 4 voters (Alice, Tim, Chris, Mike) and 2 candidates (Bob and John). The voter first connects to the Legitimization Agency (LA) to get a validation number and then connects to the Voting Facility (VF) to vote. Assume that all voters have the public keys of LA and CF. Also, assume that LA and VF have the public keys of all voters. You can manually generate the public keys for LA, CF, and all voters, store them in files, and then used them in your program.

The client is invoked (by the voter) as:
voter-cli *<LA/VF server's domain name> <LA/VF server's port number> (C/C++)*
java VoterCli <LA/VF server's domain> <LA/VF server's port> (Java)
The LA server is invoked as:
la-*ser <LA server's port number> <VF server's domain> <VF server's port> (C/C++)*
java LaSer <LA server's port number> <VF server's domain> <VF server's port> (Java)
The VF server is invoked as:
vf-ser *<VF server's port number> (C/C++)*
java VfSer <VF server's port number> (Java)

Note that the LA server can be either an iterative or a concurrent server. The VF server must be a concurrent server that enables multiple voters to connect to the LA server simultaneously.

The LA server maintains a file **"Status"** which contains the name, the SSN, and the citizenship status of the voter (only citizens can vote):

Alice	**114000000**	**citizen**
Tim	**114000001**	**permanent resident**
Chris	**114000002**	**citizen**
Mike	**114000003**	**citizen**

The LA server also maintains a file **verify** which contains the SSN and the validation number of the user. If the user does not yet have the validation number, then the validation number is 0. Initially, file verify has the following content:

114000000	**0**
114000001	**0**
114000002	**0**
114000003	**0**

Let || represent concatenation, pub(X) represent the public-key of X, priv(X) represent the private-key of X, and E(K,M) represent encrypting message M using key K. The detailed steps are given below:

1. The voter invokes voter-cli to connect to the LA server.
2. Voter-cli prompts the voter to enter his/her name (**NAME(voter)**) and SSN number (**SSN(voter)**), and sends **E(pub(LA), NAME(voter) ||SSN(voter))** to the LA server.
3. The LA server decrypts the message and prints the name of the voter on the screen.
4. The LA server checks whether the voter is a citizen (based on file "Status").
   If not, LA sends "no" to voter-cli and voter-cli prints "you are not eligible to vote" and terminates the connection with LA.
   Otherwise, the server checks whether the user already has a validation number.
   If not, the server randomly generates an 8-digit validation number **vnumber.** sends **E(pub(voter), vnumber)** to the voter, stores vnumber in file verify, connects to VF, and sends **E(priv(LA), E(Pub(VF), vnumber))** to the VF server (**you can also use digital signature instead of encryption with priv(LA)**)
   Otherwise, the server sends the vnumber in file verify to voter-cli.
5. voter-cli decrypts the message, prints vnumber, and terminates the connection with LA.
6. (this step is skipped if LA does not send VF the vnumber in step 4): The VF server decrypts the message and stores vnumber in file **"Voternumber" ((**if **Voternumber** does not exist, the VF creates the file). Voternumber has the following format:
   <validation number> <1/0>

where <validation number> is the validation number of the user and 1/0 indicates that the voter has voted/not voted, respectively.

VF terminates connection with LA

7. The voter invokes voter-cli to connect to the VF server
8. Upon connection, the voter is prompted to enter the validation number vnumber
9. After the voter enters vnumber, voter-cli sends **E(pub(VF), vnumber)** to the VF server
10. The VF server checks whether vnumber received matches a validation number stored in file "Voternumber". If not, the VF server sends a message "invalid" to voter-cli and voter-cli prints "Invalid verification number" and terminates the connection. Otherwise, voter-cli prompts the user to select an action to be performed
    Please enter a number (1-4)
    1. Vote
    2. My vote history
    3. View the latest results
    4. Quit
11. If the voter enters "1", then voter-cli sends 1 to the VF server. The VF server checks whether the voter has voted (based on file History describe in Step 12). If so, VF server sends "voted" to voter-cli, and voter-cli prints "you have already voted" and displays the menu in Step 10. Otherwise, the VF server sends a message "notvoted" to voter-cli andvoter-cli displays the following:
    Please enter a number (1-2)
    1. Bob
    2. John
12. After the user enters the number, the client sends the number to VF **encrypted using the public-key of VF**. The VF server then updates the result in file **"Result"** which has the following Format (initially the total number of votes is 0):
    Bob <the total number of votes>
    John <the total number of votes>

VF also 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):

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

Display the menu at step 10

13. If the user enters “2” (i.e. My vote history), the VF server retrieves the corresponding entry in file “History” and sends the entry to voter-cli. voter-cli then displays the entry to the user. Display the menu at Step 10
14. If the user enters “3”, then VF sends the result stored in “Result” to voter-cli. voter-cli then displays the results. Display the menu at step 10
15. If the user enters “4”, then the connection is terminated.

Grading Guideline

- Readme – 5’
- Encryption/decryption – 25’
- Concurrent server – 10’
- Other functionality: -- 60’

Submission guideline

- If you use C/C++/Java, please hand in your **source code, public and private keys,** and a **Makefile** electronically (**do not submit .o or executable code**). If you use C#, please handin your source code and executables. Please make sure that this code compiles and runs correctly on bingsuns.binghamton.edu.
- Write **a README** file (text file, do not submit a .doc file) which contains
    - Your name and email address.
    - The programming language you use (C/C++/C#/Java)
    - Platform (Bingsuns/Linux)
    - How to execute your program.
    - Core code for encryption/decrption
    - Core code for implementing the concurrent server
    - (Optional) Anything special about your submission that the TA should take note of.
- Place all your files under one directory with a unique name (such as proj-[userid] for assignment 1, e.g. proj-pyang).
- Tar the contents of this directory using the following command.
  tar –cvf [directory_name].tar [directory_name]
  E.g. tar -cvf proj-pyang.tar proj-pyang/
- Use the Blackboard to upload the tared file you created above.

Other Projects

Presentation + demo: May 6th (in class)
Submission deadline: 11:59pm May 6th
(source code + slides)

Each project is done by a group of 2 students. Each group will give a 20-25 min presentation and show demo in the class on May 6th, describing the design and implementation of the project. If you choose to do the project by yourself, you will get 15 points extra credits. You will also need to submit your code and presentation slides by 11:59pm on May 6th.

1. Buffer Overflow Attack (language: C)

Work through the shell example given in
http://www.maths.leeds.ac.uk/~read/bofs.html
Note: Use a linux machine (instead of bingsuns) to do this project.