CS-352 Final Projects
Spring 2024

Due: 5/18/2024 (No Extensions)
You may work in groups of 8

Overview

The final project is a chance for you to demonstrate your creativity and your skills in cryptography by tackling some real-world security problems. You have three options for the final project: (1) a programming project; (2) a practical skills project; or (3) a self-proposed project. If you choose option 1, you can select any one of the topics listed under the section *Programming Projects* below. You can use any existing cryptographic libraries, tools, programming languages or platforms to complete your project. The details and requirements of each topic are given below.

You can work on the final project individually or in a group of up to 8 students. The project descriptions below are not complete designs. You can use any reasonable design that meets the specifications. At the end of the semester, you need to submit your project and present it to the class. Each presentation should be 10 minutes long and should explain the design and implementation of your project. If there is not enough time for all presentations, you can either submit a video of your presentation or show your project to the instructor.

Each project topic can be chosen by **no more than 4 groups**. If your project has some extra features that go beyond the requirements, you may earn up to 10 points of extra credit. Please consult with the instructor to find out if feature XYZ qualifies for extra credit.

Programming Projects

Please complete one of the following programming projects.

1. Secure File Sharing

In this project, you are going to use the concepts of concurrent server to write a secure peer-to-peer file-sharing system. Concurrent server can service multiple clients at the same time.

There are 3 peers (hosts) connected to a network; each stores a number of files. Each file is associated a keyword that can be used to search for the file. There may be duplicates of the same file stored at multiple peers; these duplicates are associated with the same keyword. Different files can be associated with the same keyword. The goal of this project is to build a mechanism that enables these peers to securely share their files with each other. The overall structure of the file sharing system is given below.

An indexing server keeps track of which files are stored at which peers. Each peer has an account with the server. When a peer connects to the server, server prompts the peer to provide ID and password. The peer then sends its domain name and port number to the server. The peer also sends names of files it has and the corresponding keyword to the server; the server then updates its index database. A peer that wants a specific file queries the server with keyword. Server searches for the keyword in its index database and returns the <`domain_name, port_number`> where `domain_name` and `port_number` are the domain name and the port number of the peer who has the file.

Requesting peer then chooses one peer from the list returned by the indexing server and initiates a file transfer operation. Each peer contains a list of public keys of other peers of the form <`name, publickey`>. Files need to be transferred securely and should not be modifiable by unauthorized users during transmission.

Your implementation must provide both confidentiality and digital signature. For digital signature you must provide the user with a choice of using RSA or Digital Signature Algorithm (DSA; https://bit.ly/2TvvGSt). Both digital signature schemes must be supported.

2. Secure Chat

In this project you are to implement a system which enables a group of users to chat securely. All users are registered with the chat server. When the user wants to chat with another registered user, he first connects to the chat server and enters his/her user name and password. The server verifies the user name and password, and if correct, the user's status is changed to "online". Next, the user may enter the user ids of users with whom he wishes to chat (could be more then one). At any given time the user should be able to check what other users are online and invite them to the ongoing conversation.

Once the user specifies the users with whom he wishes to chat, the server generates a symmetric key, and securely distributes it to all the specified users and the user who initiated the chat. To achieve secure key distribution you must encrypt the symmetric key using the public keys of the respective users (you may assume that server knows the public keys of all users). If one of the specified users is not online, the requesting user is notified about this.

After the encrypted symmetric key has been distributed to all users, the users decrypt the symmetric key using their private keys, and the chat session may begin. All messages exchanged during the chat must be encrypted using the symmetric key provided by the server and must be delivered to all users participating in the chat. Any user may choose to leave the conversation. If the user disconnects from the chat server, his status should be changed to "offline". All users who are connected to the server, must have a way to check whether a given user is online.

You do not need to support multiple chat sessions.

Your implementation must provide both confidentiality and digital signature. For digital signature you must provide the user with a choice of using RSA or Digital Signature Algorithm (DSA; https://bit.ly/2TvvGSt). Both must digital signature schemes must be supported.

3. Secure Purchase Order

Implement a secure purchase order system that allows the user to enter a purchase request and routes it to Order Processing Department (OPD) for signature. Each customer has an account with the online purchasing system. The online purchasing system has the public-key of every customer.

First, the customer enters his/her ID and password. The system verifies his/her ID and password. Next, the customer sends the order as well as a timestamp to the OPD - confidentiality and digital signature must be provided. OPD verifies the signature of the customer and checks if the product is available. If the customer's signature is verified and the product is available, OPD prepares the order. After OPD processes the order, OPD retrieves the customer's email address and sends an email to the customer, indicating that the order has been shipped. OPD should be implemented as a concurrent server.

Your implementation must provide both confidentiality and digital signature. For digital signature you must provide the user with a choice of using RSA or Digital Signature Algorithm (DSA; https://bit.ly/2TvvGSt). Both digital signature schemes must be supported.

4. Secure Internet Poker

This project implements poker on the Internet. It will accept two players. The house has each player's public key.

1. Each player generates a session key and distributes the session key to the house securely. The session key is used to encrypt message sent between the house and the players.
2. The house randomly generates three numbers between 1 and 15 and sends the numbers to player.
3. The house randomly generates three numbers between 1 and 15 and sends the numbers to player.
4. There are three rounds. In each round, each player chooses a number out of the three numbers and sends the number to the server. The server compares the number. The player who chose larger numbers than the other for at least two rounds wins. At the end of the 3rd round, the house announces the winner.
5. The session key will be destroyed after a player leaves a current session.

Your implementation must provide both confidentiality and digital signature. For digital signature you must provide the user with a choice of using RSA or Digital Signature Algorithm (DSA; https://bit.ly/2TvvGSt). Both digital signature schemes must be supported.

5. Secure Banking

Implement the authentication and secure communications protocols for a distributed system consisting of a bank server and a number of automatic teller machines (ATMs). Assume one bank server and 2 ATMs. The bank should be implemented as a concurrent server.
Public key cryptography is employed for secrecy, integrity-protection, and authentication. The bank server's public key is stored by the ATM, with the corresponding private key stored by server the connected to that ATM. You may also assume that the bank has the public keys of the two ATMs.

ATM would work like this:

1. The customer enters his/her 6-digit ID (e.g. 124356) and password.
2. The ATM contacts the bank server to verify the customer's ID and password.

3. The customer selects an action to be performed, and that action is performed by the bank server.

There are five actions: display the amount of money in the account, deposit money, withdrawals, account activities (time and date when the user performed transactions and what transactions the user performs), and quit. Assume that the bank knows the ATM's public key and the ATM knows the bank's public key. The ATM communicates with the bank by running a protocol that satisfies the following requirements:

1. It authenticates the customer to the bank server (ID, passwd).
2. It authenticates the ATM to the bank server through public-key encryption.
3. It preserves the confidentiality of communications between the bank and ATM.
4. The bank server takes actions in response to customer uses of an ATM.

Your implementation must provide both confidentiality and digital signature. For digital signature you must provide the user with a choice of using RSA or Digital Signature Algorithm (DSA; https://bit.ly/2TvvGSt). Both digital signature schemes must be supported.

6. Secure Blockchain

Blockchain is the uderlying technology of Bitcoin, Etherium, and other cryptocurrencies. Its power comes from the idea of distributed ledger which has many applications beyond cryptocurrencies.
If you choose to do this project, please take some time to read and learn about the basics of the blockchain. The following links provide good resources that will get you started:

- Blockchain high-level overview: https://bit.ly/2uHcWzd
- Blockchain tutorial: https://bit.ly/2GN1nzL
- Blockchain implementation tutorial https://bit.ly/2fOQsqt
- Blockchain implementation tutorial https://bit.ly/2Jazn7x

Cryptographic concepts, including public key cryptography and hashing, play central roles in the blockchain. The goal of this project to implement a simulation of the public blockchain similar to Bitcoin. Your implementation should include the following key parts of the blockchain:

1. Individual users should be able to broadcast transactions to the miners i.e., parties responsible for verifying the blocks.
2. All transactions within a block must be digitally signed by the user initiating a transaction.
3. Miners verify the transaction within a block by solving a computationally hard problem. For the purpose of this project you can simplify the difficulty of the problem in order to be able to test and demo your project. For example, finding a number that results in the block's hash containing a hexadecimal number "0x0a".
4. The miner who solves the problem first, gets a reward according to the rules of blockchain.

5. When the block is verified, the miner broadcasts the verified block to the users.

6. In order to deal with the issues associated with spoofed blocks (e.g., if the attacker beats the miners) the users must use the longest chain rule.

Your implementation should support at least three users and three miners and must provide both confidentiality and digital signature. For digital signature you must provide the user with a choice of using RSA or Digital Signature Algorithm (DSA; https://bit.ly/2TvvGSt). Both digital signature schemes must be supported.

7. Identity-Based Cryptograaphy

Identity-based cryptography (IBC) is a branch of public-key cryptography that allows users to use any unique identifier, such as their email, domain, or IP address, as their public key. This simplifies the process of encrypting messages or verifying signatures without having to distribute or store public keys in advance. However, IBC also requires a trusted third party, called the private key generator (PKG), that can create and deliver the matching private keys to the authorized users. The PKG has a master public key and a master private key, which it uses to generate any user's private key based on their identifier.
The goal of this project is to implement an IBC system that can support an arbitrary number of authorized users. The system should include a PKG server that maintains a list of authorized users and their credentials. A user can request a private key from the PKG by connecting to it and sending their username and password. These credentials should be encrypted with either the PKG's public key and its identifier, or using TLS, to ensure secure transmission. The PKG then verifies the credentials and generates the user's private key using the master private key and the user's identifier. The PKG then securely (e.g., using TLS) sends the private key back to the user. A user should then be able to securely send a message to another user by encrypting it with the receiver's public key and the PKG's master public key. Similarly, the system should support the creation and verification of digital signatures using IBC.
You may find the following links useful:

- IBC basics
- More IBC basics
- Even more IBC basics
- Sample IBC library codes (Github)

Presentation Guideline

If time permits, each group will give an in-class presentation. If the time does not permit, then please submit a video of your presentation or demo your project to the instructor. When making your video, demo, or in-class presentation, please adhere to the following guidelines:

- Give 5-10min presentation.
- Show and explain how your code runs.

- Discuss the design (how do you provide confidentiality, authentication, digital signature, integrity etc)
- Implementation (language, libraries, implementation issues).
- What you have learned by doing the project.

What to Submit

- A well written document clearly detailing your design. The document shall contain the following sections:
  - **Abstract:** a brief paragraph succinctly describing your project and its salient features.
  - **Introduction:** a section describing the services your project delivers e.g. secure exchange of chat messages for all parties involved. Users can also check the authenticity of messages they receive, etc.
  - **Design:** a section outlining the design of your system e.g. components, their interactions, use cases, etc. Use diagrams in order to illustrate your points.
  - **Security Protocols:** a section explaining the cryptographic protocols used for e.g. exchanging keys, secure communications, etc; similar to the protocol diagrams and discussions you seen in class. Please be sure to explain how your protocol achieves confidentiality, authenticity, etc.
  - **Implementation:** a section discussing the implementation details of your projects. E.g. what language, libraries, platform, etc. did you use and why? Comment on any special implementation techniques. Use screenshots.
  - **Conclusion:** a section giving a brief recap of the project.
- Your code.
- README file containing:
  - Names of group members.
  - Contributions of each member.
  - Instructions on how to run the code.
- Anything else you think is relevant.

Practical Skills

Please complete **all** of the following:

1. Configure a system to function as a certificate authority. You may use any system. You may find the following link helpful: https://help.ubuntu.com/community/OpenSSL. Please note: you must actually set up your own authority; not simply purchase an SSL certificate from somewhere.

Set up a website (locally on your machine or some remote server) which uses the services of your certificate authority. When setting up a website with a certificate, please do so using **all** of the following:

- Microsoft IIS Web Server (https://bit.ly/2F8Ryuh). If you do not have access to a Microsoft Windows operating system, please contact the instructor.
- Apache Web Server (https://bit.ly/1TgO6Ag)
- Node.js Web Server (https://nodejs.org/en/)

2. Set up and configure a Virtual Private Networking Server (VPN) server. You may find the following link helpful: https://help.ubuntu.com/community/OpenVPN.

3. Set up an SSH server. Describe how to configure the server such that the user can securely connect to the server without having to enter the password every time.

4. Set up an IPSec channel between two systems.

5. Set up an email account that uses Pretty Good Privacy (PGP) for security. The sender must be able to send encrypted and digitally signed emails and receiver should be able to verify them. You may find the following link helpful https://support.mozilla.org/en-US/kb/digitally-signing-and-encrypting-messages

6. Setup a DNSSEC-based DNS Server https://www.digitalocean.com/community/tutorials/how-to-setup-dnssec-on-an-authoritative-bind-dns-server–2. Demonstrate the function of your server using a DNSSEC client. Network traffic captures with tools such as Wireshark can also be used to further illustrate the function.

7. Setup a simple Kerberos Server and a Simple Kerberos Client that logs onto the Kerberos server. You may find the following resource helpful https://www.atlantic.net/dedicated-server-hosting/how-to-setup-kerberos-server-and-client-on-ubuntu-20-04/

Presentation Guideline

If time permits, each group will give an in-class presentation. If the time does not permit, then please submit a video of your presentation or demo your project to the instructor. Please see the guidelines below:

- Give 5-10 min presentation.
- Show your systems in action.
- Discuss how you set up and configured each type of server. What tools did you use? What configuration choices did you make? Why? Any challenges you encountered? Solutions?
- What platform did you use? Anything else relevant?

What to Submit

- A well written document clearly detailing your design. The document shall contain the following sections:
  - **Abstract:** a brief paragraph succinctly describing your project and its salient features.
  - **Introduction:** a section briefly describing the systems you set up and the underlying principles of their function.
  - **Configuration:** a section outlining the configuration of your systems e.g. system components, their interactions, different cases for which you configured your systems, etc. Use diagrams, snippets of configuration files, etc in order to illustrate your points.
  - **Security Protocols:** a section explaining the cryptographic protocols which make creation of SSL certificates, SSH, VPN, and IPSec, and DNSSEC possible.
  - **Testing and Field Evaluation:** Describe the tests you devised in order to evaluate the function of the systems you configured. What functionality was each test designed to evaluate? What were the results of each test? Did the results cause you to take any corrective actions? If so, then explain. Use screenshots to illustrate the functioning of your systems.
  - **Conclusion:** a section giving a brief recap of the project.
- Configuration files (if any).
- README file containing:
  - Names of group members.
  - Contributions of each member.
  - Instructions on how to run the code.
  - Anything else you think is relevant.

Propose Your Own Project

Propose your own project idea! Be creative! The proposed idea should be at least as challenging as the projects listed above and require application of principles and practices of cryptography. Please make sure you get your idea approved by the instructor. All proposed projects must include a presentation, demo, or a video component and a written document similar to that required in programming and practical skills projects.

Important Dates

- Please email me your project choice by: **3/1/2024**.
- All material i.e. reports, codes, etc must be submitted via CANVAS by **5/18/2024 (No Extensions)**.