Project 2: PSI protocol

This project is due on **October 03, 2024** at **11:59pm CST**. We strongly recommend that you get started early. You have a total of 3 grace days for the 5 MPs, which allow for a 24-hour extension on your MP deadlines. You can use up to 2 grace days per MP, but no more than 3 in total across all MPs. If you submit your assignment after the due date and beyond the grace day period, you will lose 20% of the total points for each late day.

The project helps you to get familiar with the **Private Set Intersection (PSI)** protocol. The code should be uploaded to PrairieLearn and pushed to your personal repository by **October 03, 2024** at **11:59pm CST**. Detailed submission requirements are listed at the end of the document.

This is an individual project; you SHOULD work **individually**.

The code and other answers you submit must be entirely your own work, and you are bound by the Student Code. You MAY consult with other students about the conceptualization of the project and the meaning of the questions, but you MUST NOT look at any part of someone else's solution or collaborate with anyone else. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

This semester we have anti-cheating check on our autograder, so please do not take risk to cheat. We will routinely check for plagiarism. You will receive one warning if any is detected, and repeat offenses will result in a zero on the corresponding MP and a report to the appropriate authorities.

Solutions MUST be submitted electronically in your git directory, following the submission checklist given in this handout. Details on the filename and submission guideline is listed at the end of the document.

Release date: September 17, 2024
Due date: October 03, 2024 at 11:59pm CST

Introduction

Cybersecurity incidents are an ever-growing problem for companies. Recently, the President of the United States called for steps to improve cybersecurity, including new legislation that would limit liabilities of companies that report incidents that had happened to them. In this machine problem, we will study how companies can share cybersecurity incidents with each other without revealing unnecessary information about the incidents.

Each group represents a company that has experienced 10 standard cybersecurity incidents. You seek other companies that have experienced incidents similar to the ones you have experienced. You want to learn whether other companies have experienced similar incidents, but without sharing the set of all incidents you have experienced.
You are given a list of cybersecurity incidents in the file `incident_list.txt` labeled from 1 to 25. In this machine problem, each group will select exactly 10 cybersecurity incidents randomly from the list. You will use a private set intersection (PSI) protocol to see if other groups have experienced the same incidents. Each run of the protocol requires exactly two groups: a Client group and a Server group.

Please read the assignment carefully before starting the implementation.

Objectives

- Understand the PSI protocol.
- Be able to implement part of the PSI protocol.
- Gain familiarity with Crypto APIs in Java.

Problem Description

This machine problem would help you get familiar with the PSI protocol. You are provided with a complete implementation of the Client (`ClientPhase1.java`, `ClientPhase2.java`), a partial implementation of the Server (`Server.java`), and some useful classes (`Inputs.java`, `Paillier.java`, and `StaticUtils.java`). We will guide you through the implementation of server's side of the PSI protocol in `Server.java`.

You will need to implement part of the protocol and test `Client.java` and `Server.java` on your own machine.

Java 8 (0 points)

Java 8 is needed to run the code for this assignment. Check your Java environment with command `java -version` and `javac -version`. Please make sure you have Java 8 installed on your machine.

Reference

- JRE and JDK Installation (Java 8). `http://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html`

What to submit No submission required.

Implement the PSI Server (100 points)

In this section, you need to complete the implementation of the PSI Server and test it on your own machine.

Implement `Server.java`

Please place your code in the file `Server.java` starting at line 26. Detailed instructions are included as comments in the file.

Tips:

- Please refer to the slides of the Crypto Constructs lecture for more details on the PSI protocol. Note that you only need to implement the server's side of the protocol.
- Please use the provided class `Paillier.java` for the homomorphic operations.
- Please use the provided method `randomBigInt(BigInteger n)` to generate r_i mentioned in the slides.

Run the PSI protocol

In this section, you can test the protocol within yourself and act as a Server and a Client at the same time.
Before start running the protocol, please perform the following **initial steps**:

1. Compile the code using
    - Unix/Linux/Mac: `javac -cp .:flanagan.jar *.java`
    - Windows: `javac -cp ".;flanagan.jar" *.java` (note: if you are getting an error saying package flanagan.math does not exist, try javac -cp .;flanagan.jar *.java)
2. Select exactly 10 unique cybersecurity incidents from the list in file `incident_list.txt`.
3. Create another text file called `client_inputs.txt`, put the labels of the 10 incidents you selected in step 2, and separate the labels by new lines. See example `example_inputs.txt` to understand the format. DO NOT use the example `example_inputs.txt` file.

4. Repeat the above steps for the server, and create another text file called `sever_inputs.txt`. It is possible that `client_inputs.txt` and `server_inputs.txt` may share some common incidents.

The **Client** needs to perform the following steps:

1. Run the following program (Use your netid and the `client_inputs.txt` you generated in the last step as input):
   - Unix/Linux/Mac: `java -cp .:flanagan.jar ClientPhase1 <client_inputs.txt> <netid>`
   - Windows: `java -cp ".;flanagan.jar" ClientPhase1 <client_inputs.txt> <netid>`
2. This will generate two files with filename being `netid` and `ClientPK.out`
3. Email both of these files to the Server (you don't need to send an actual email, just put them in the Server's folder).
4. Wait for the server to run its program and send you the file `netid.out`.
5. After you receive the file from Server named `netid.out`, put it in the same directory as the code and run the following program:
   - Unix/Linux/Mac: `java -cp .:flanagan.jar ClientPhase2 <client_inputs.txt> <netid.out>`
   - Windows: `java -cp ".;flanagan.jar" ClientPhase2 <client_inputs.txt> <netid.out>`
6. The program will output the common incidents.

The **Server** needs to perform the following steps (Note: You must implement the server PSI sub-protocol correctly in order to perform the following steps):

1. Put the files (`ClientPK.out` and `netid`) received from the Client in the same directory as the code.
2. Run the following program (Use the `server_inputs.txt` you generated in the last step as input) :
   - Unix/Linux/Mac: `java -cp .:flanagan.jar Server <server_inputs.txt> <netid>`
   - Windows: `java -cp ".;flanagan.jar" Server <server_inputs.txt> <netid>`
3. This will generate a file with the filename `netid.out`
4. Email `netid.out` back to the Client.

The output file at Client in Step 6 should contain all the common incidents between `client_inputs.txt` and `server_inputs.txt`. You may test your code with different sets of incidents.

What to submit

- Place `Server.java` in `mp2`.

Autograder

We have set up an autograder on PrairieLearn. To use it, you must first enroll in the course CS 463: Computer Security II by visiting `https://us.prairielearn.com/pl`. After enrolling, navigate to the **Assessments** section within CS 463: Computer Security II, where you will find **MP2**. Upload your `Server.java` file to receive your scores.

Important Note

1. Your highest grade and grading report will be automatically recorded in our server.
2. You **must** submit through PrairieLearn to our auto-grader **at least once** before the deadline to get a grade. We don't grade your code manually, the only way is to use our auto-grader.
3. Try auto-grader as early as possible to avoid the heavy traffic before the deadline. **DON'T DO EVERYTHING IN THE LAST MINUTE!**
4. If the auto-grader is down for a while or you are encountering weird auto-grader behaviour, don't be panic. Contact us on Campuswire, we will fix it and give some extension if necessary.
5. We have anti-cheating mechanisms in place. **NEVER CHEAT OR HELP OTHERS CHEAT!**
6. Please use Java version earlier than 1.8. Don't use any Java modules except for the ones that are specified in the release.
7. Never abuse the auto-grader.

Submission Instructions

There are two tasks you need to complete for submitting your work:

1. Upload to PrairieLearn

You must upload your code to PrairieLearn, which serves as our autograder. Specifically, upload `Server.java` to the **MP2** on PrairieLearn. It is essential to complete these uploads to receive and record your MP2 score.

Submission Limit

Please note that you are limited to a maximum of 30 submissions for MP2 on PrairieLearn. Be mindful of each submission you make, as your submission count is limited.

Timeout

The MP2 autograder on PrairieLearn has a timeout limit of 600 seconds. If your code exceeds this time limit, the autograder will throw an error and assign a score of 0 for that submission. Ensure your algorithm is time-efficient to avoid timeouts.

2. Upload to GitHub

In addition to uploading to PrairieLearn, you must also upload your code to GitHub. After the assignment deadline, the course staff will compare the code you submitted to PrairieLearn with the code you uploaded to GitHub to ensure they match. We will also perform anti-cheating checks to detect any plagiarism. Please note that any form of code plagiarism is strictly prohibited, including copying parts of someone else's code or code from previous students.

Folder Structure

Place the following files in the `mp2` folder of your git repository.

- `Server.java` (80 points) [Implementation of the PSI server]

Remember to make sure there is a `mp2` folder in your main branch with `Server.java` in it. Your score will be deducted if you fail to do so.

Git Upload

Use the following commands to push your submission for grading (it is advisable to do this frequently for better version control). Grading will be based on the latest submission to the autograder before the deadline. Please push to your main branch rather than creating a separate branch.

```
git add -A
git commit -m "REPLACE THIS WITH YOUR COMMIT MESSAGE"
git push origin main
```