Project 2: PSI protocol

This project is due on **February 25, 2024** at **11:59pm CST**. We strongly recommend that you get started early. You will get 80% of your total points if you turn in up to 72 hours after the due date. Late work will not be accepted after 72 hours past the due date.

The project is split into two parts. Checkpoint 1 helps you to get familiar with the language and tools you will be using for this project. You do NOT need to make a separate submission for each checkpoint. That is, the questions for the checkpoint 2 are within Coursera, and the code for checkpoint 1 should be pushed to your personal repository by **February 25, 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.

Solutions MUST be submitted electronically in the Github directory, following the submission checklist given at the end of each checkpoint. Details on the filename and submission guideline is listed at the end of the document.

Release date: February 2, 2024
Checkpoint 1 Recommended Due date: February 14, 2024
Checkpoint 2 Due date: February 25, 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.

Checkpoints

This machine problem is split into 2 checkpoints. Checkpoint 1 would help you get familiar with the PSI protocol. You will need to implement part of the protocol and test `Client.java` and `Server.java` on your own machine. In Checkpoint 2, you will run the protocol with your own machine in a few different settings and answer a few questions about your results.

Implementations

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`.

2.1 Checkpoint 1 (80 points)

This machine problem is split into 2 checkpoints. Checkpoint 1 would help you get familiar with the PSI protocol. You will need to implement part of the protocol and test `Client.java` and `Server.java` on your own machine.

2.1.1 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.

2.1.2 Implement the PSI Server (80 points)

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

2.1.2.1 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.

2.1.2.2 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`, `server_inputs.txt`, and `client_inputs.txt` in `mp2`.

2.2 Checkpoint 2 (10 points)

For this checkpoint, you'll need to test the PSI protocol on your own machine while considering both honest and malicious actors.

2.2.1 Test the Protocol

In the following tests, you should perform the role of the Client or Server at least once.

2.2.1.1 Honest Client and Honest Server

In this setting, both Client and Server group behave honestly and help each other to find the common incidents.

2.2.1.2 Malicious Client and Honest Server

In this setting, Client is ill intentioned and wants to learn as much information about the Server incidents as it can. In this case, Client is allowed to choose any incidents he wants and can run the protocol with the Server multiple times with different list of incidents.

2.2.1.3 Honest Client and Malicious Server

In this setting, Server is ill intentioned and wants to learn as much information about the Client incidents as it can. Client is honest.

What to submit No submission required.

2.2.2 Answer Questions (10 points)

For this part, please go to Coursera Week 4: "MP2: Private Set Intersection" to answer all the questions. **Please note that you can make an one-time submission only**.

What to submit

- Please go to Coursera to submit your answers.

2.3 Submission Instruction

2.3.1 Folder Structure

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

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

- `server_inputs.txt`
- `client_inputs.txt`

Remember to make sure there is a `mp2` folder in your main branch with 3 files `Server.java`, `client_inputs.txt` and `server_inputs.txt` in it. You will get error feedback from our auto-grader if you don't do so.

2.3.2 Git Upload

Then use the following commands to push your submission for grading (do this often for better version control). The grading will be based on the latest auto-grader submission before the deadline. Please push into your main branch rather than a self-created one.

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

2.3.3 Auto-grader

This semester we use auto-grader to eliminate any misalignment between student's working and grading.

Please use Java version 1.8 to write your Server.java.

To use the auto-grader please do as follows:

1: Upload your files into your git repo as mentioned above

2: Send HTTP POST request with your netid

We provide 2 methods, pick one as your preference. We recommend command line because it's more reliable.

Method1: Command Line

To grade cp1:

```
curl -X POST -H "Content-Type:  application/json" -d @submission.json
http:/128.174.136.25:8081/grade_cp1
```

`submission.json` is a json file in your local computer, with only one key-value pair "netid":"YOUR_NETID"

Replace "YOUR_NETID" with your own netid, such as "netid":"aj3".
It's fine to change "submission.json" into other names, just remember to ensure it is a json file and you are doing the same change to the command line.
Remember, it's HTTP request rather than HTTPS.

Method2: Web Tool

If you don't want to use the above command line (sometimes more nasty), you can also use web tool such as Postman to send HTTP POST request to http:/128.174.136.25:8081/grade_cp1 with a json body carring a key-value pair, "netid":"YOUR_NETID"

Replace "YOUR_NETID" with your own netid, such as "netid":"aj3".
Remember, it's an HTTP request rather than HTTPS.

3: Get Feedback

The feedback will be a web response to your HTTP request. The format is as follows (if no error message):


```
{
  "error": "",
  "result": {
    "actual": "1 is common.\n2 is common.\n3 is common.\n4 is common.\n5 is common.\n6
is common.\n7 is common.\n8 is common.\n9 is common.\n10 is common.\n",
    "expected": "1 is common.\n2 is common.\n3 is common.\n4 is common.\n5 is
common.\n6 is common.\n7 is common.\n8 is common.\n9 is common.\n10 is common.\n",
    "success": true
  }
}
```


Any error will be put into "error" field such as Java error trace, wrong folder structure, git error or wrong web request.
Your grading result and grade for this checkpoint will be put into the "result" field. The result field contains three subfields. **"actual"** subfield is the output generated by your code in Server.java. **"expected"** subfield is the expected output with your inputs. **"success"** subfield is a string match of the actual and expected outputs. Remember, your grade and grading report will be automatically recorded on our server.

If you see **"curl: (6) Could not resolve host: application"**, don't worry, just wait for the result.

2.3.4 Timeout

You have 5 min for checkpoint1. Auto-grader will throw an error if timeout value is reached. Please make sure your code is time-efficient.

2.3.5 Abuse

You should only submit your own NetID when querying the auto-grader. Please do not abuse the auto-grader in any form. Abuse behaviors include but are not limited to using other students' NetIDs to submit a query and get their grading results. We will closely monitor the NetIDs sent by each IP address. Abusing the auto-grader is considered a form of violation of the student code of conduct, and the corresponding evidence will be gathered and reported.

2.3.6 Important Note

1: Your highest grade and grading report will be automatically recorded in our server.

2: You **must** send request 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: If you see **"curl: (6) Could not resolve host: application"**, don't worry, just wait for the result to show up.

6: We have anti-cheating mechanisms, NEVER CHEAT OR HELP OTHERS CHEAT!

7: When auto-grader tests your code on dataset1, we will mask the location info for those who don't share their location even though it is provided in the dataset1! Be careful when you are testing in your local computer. Last semester lots of students overlooked this, causing misalignment between local and online testing result.

8: If you are using web tool such as Postman, sometimes you will fail and see "server hang up". Keep trying more times or use command line instead.

9: Please use Java version earlier than 1.8. Don't use any Java modules except for the ones that are specified in the release.

10: Read section 2.4.5 carefully. Never abuse the auto-grader.