CS-523: SecretStroll Project

You are an avid user of location-based applications such as Foursquare, Yelp, and Google Maps. These applications provide users with details and reviews of nearby Points of Interest (POIs) based on their location. You have installed and logged into Foursquare on your smartphone, and you always keep your location on to receive notifications about nearby places. However, you recently read an article on location tracking,[1] and it terrified you. Therefore, you decide that not only are you going to disable continuous location sharing but you are also going to build a better, more private, location-based service.

You would like your application (app) to work as follows: When a user opens the app, the app authenticates with the service provider. After authentication, the app sends the user's current location to the service provider. The service provider returns nearby POIs. Figure 1 shows an overview of the application.

Your app follows a privacy-friendly design and you do not want to earn revenue by monetizing users' personal data. Instead, you plan to use a subscription model. To make the app more specialized and affordable, you decide to implement interest-based subscriptions. For example, Bob can get a monthly subscription for restaurants ($2.50), gyms ($1.99), and dojos ($1.50), for a total of $5.99. The user selects a subset of subscriptions when sending the location to the server so that only POIs of that type are retrieved.

As you learn in the CS-523 lectures, there are three main information leaks in this app which could pose privacy risks to users:

1. Disclosing the sender of each query through authentication creates a time-stamped list of visited places for each user, and undermines their privacy. The adversary, in this case, is the service provider.

2. Disclosing user location and subscription data in each query enables the service provider to infer information about a user's behavior. The adversary, in this case, is the service provider.

3. Network traffic meta-data between the user and the service provider can reveal information about the query, and therefore about the user's whereabouts. The adversary, in this case, is someone who eavesdrops the network connection between the user and the service provider.

[1] `https://www.nytimes.com/interactive/2019/12/19/opinion/location-tracking-cell-phone.html`

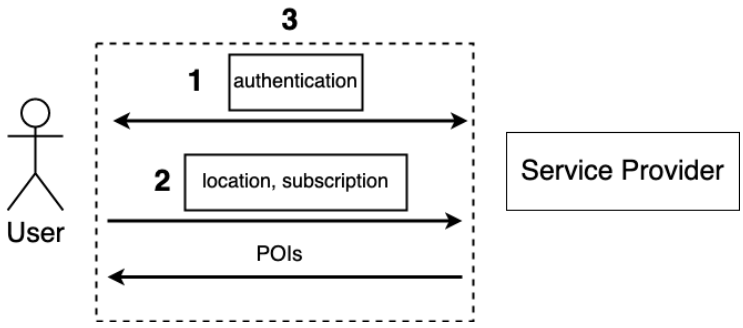

Figure 1: Overview of your location-based system. There are three information leaks to address. The numbers indicate where each leak occurs, and the project part corresponding to the leak.

We illustrate the points in which the leaks happen in Figure 1. This project is divided into three parts which tackle each of the information leaks mentioned above in each part.

Project Objectives

In this project, you have to do the following:

- **Part 1.** Design and implement an anonymous authentication mechanism using attribute-based credentials.
- **Part 2.** Conduct a privacy evaluation of a location-based service. Propose and evaluate a privacy defence.
- **Part 3.** Implement and evaluate a network-traffic fingerprinting attack when a user makes a location query.

Deliverables

By the due date specified on Moodle, you must submit:

- A PDF file with your IEEE-format two-column report of **at most 5 pages** (including figures, excluding references). The sections for each part contain the requirements for the report content. The LaTeX template for your report can be found at `https://github.com/spring-epfl/CS-523-public`. It is important that figures in the report have readable fonts (rule of thumb, font as big as the font of the main text).
- A `zip` or `tar.gz` code archive. The archive should contain three folders: `part1`, `part2`, `part3`. Each folder should contain all the code for the respective project parts. Code quality is *only* graded in Part 1. We will not grade code quality in Parts 2 and 3. However, all code needs to be

working, and the documentation for each part needs to include README files containing the instructions for running all the runnable artifacts in the respective folders.

Suggested Timeline

You have eight weeks in total for this project. We suggest the following timeline:

- Weeks 1–3: Complete attribute-based credentials (Part 1), and write the corresponding report section.
- Weeks 4–5: Complete (de)anonymization of user trajectories (Part 2), and write the corresponding report section. Start the network traffic data collection (Part 3).

  Important: Data collection for Part 3 is a time-consuming process. We **strongly** recommend that you begin data collection early. Data collection can be run in the background while you complete the other parts. Please read Part 3 for instructions on data collection.
- Weeks 6–7: Complete network traffic fingerprinting (Part 3), and write the corresponding report section.
- Week 8: Wrap up and report submission.

We **strongly** recommend that you finalize the code submission and the report for each part right away as soon as you finish working on it, as opposed to finalizing the archive and the report right before the deadline.

1 Attribute-Based Credentials

A common practice in location-based apps is to ask users to sign up and create an account. Users need to be logged in to their account to search for places, and search requests are therefore linked to a user's account. This enables the service provider to create a profile of visited places for each user. This is a privacy risk (see Figure 1, box 1). While SecretStroll does not need the user's identity to search for nearby POIs, it does need some way to verify that the user paid for a subscription. Requiring users to log in is the easiest mechanism, but lacks privacy. In Part 1 of the project, we will use attribute-based credentials (ABCs) instead to let the service provider check that the user has a valid subscription, without requiring users to identify.

SecretStroll uses ABCs as follows. When users sign up, the SecretStroll server issues a credential for them (step 1 in Figure 2). Unlike classic authentication systems, there is no "login" protocol in SecretStroll. Instead, with each request, users provide a proof that they have a valid subscription using their subscription credential. To link the request for PoIs at a location to this proof, we

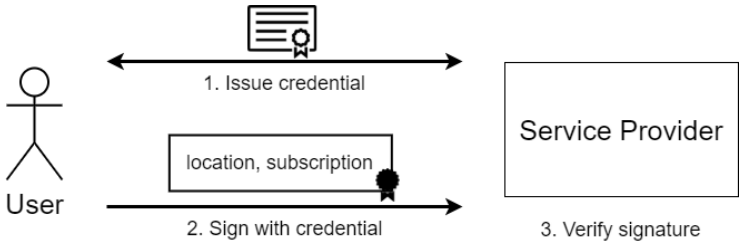

Figure 2: Using attribute-based credentials as an authentication mechanism.

will use the proof to "sign" the search request (step 2 in Figure 2). The SecretStroll server verifies this proof/signature before processing the corresponding request (step 3 in Figure 2). The properties of ABCs ensure signatures are unlinkable, and thus that users can remain anonymous when using the service.

You decide to use the Pointcheval and Sanders (PS) attribute-based credential scheme [2]. You have seen this scheme in the CS-523 lecture slides (Lecture 2 - Anonymous Authentication). It takes a bit of effort to extract the ABC scheme from the original paper [2]. Therefore, we refer to the lecture notes about PS credentials [1] for the details.

In Part 1 of this project, you have to implement PS attribute-based credentials, test them, integrate them into the SecretStroll system, and evaluate their performance.

1.1 Use the skeleton

To help you with the project as a whole, we have prepared a skeleton for SecretStroll which handles the communication parts of SecretStroll. In Part 1 you will integrate the ABC scheme into this skeleton. Later on, Part 3 uses the finished system of Part 1 as a black-box.

You are only allowed to change the `stroll.py` and `credential.py` files in the skeleton. But you can create and edit new files at will.

Using the skeleton. The skeleton is documented and guides you towards what you have to do. We repeat some important notes and hints in this document, but you will need to read the README for further instructions on how to use the SecretStroll system.

1.2 Implementing the ABC scheme

We recommend that you first implement and test the Pointcheval-Sanders ABC scheme, before integrating it with the rest of the SecretStroll system. You can find a suggested structure in the file credential.py in the provided skeleton. However, you are free to adopt your own structure.

Debugging cryptographic systems is difficult. Therefore we suggest the following smaller steps.

1. Implement the PS signature scheme by implementing the KeyGen, Sign, and Verify functions described in Section 2 of the lecture notes.

2. Implement a basic version of the issuance protocol described in Section 3 of the lecture notes. For simplicity, first omit the zero-knowledge proof that the commitment C has been computed correctly. The resulting credential should contain a valid standard PS signature on the messages you specified.

3. Extend the issuance protocol to also compute the non-interactive zero-knowledge proof that commitment C has been correctly formed as well as verification of this proof by the issuer.

4. Implement the showing protocol described in Section 3 of the lecture notes.

To ensure that each step is successful, make sure to write a small test suite that tests the new functionality. This test suite is part of the deliverables for this part of the project. See below.

1.3 Integrating ABCs into SecretStroll

Your next task is to integrate the ABC scheme you just implemented in the bigger SecretStroll system. To do so, you should integrate your ABC scheme with the skeleton by filling out the functions defined in `stroll.py`.

A common ABC practice is to include a secret key in the credential as an attribute. You should follow this practice and include a secret key in the credential. (*Hint:* Users should not reveal their secret key.)

Furthermore, assuming that two users A and B perform a search request and reveal the same set of subscriptions, given one of these two signatures, the SecretStroll server should not be able to distinguish the credential used to produce the signatures or learn anything beyond the revealed attributes about the signer.

Finally, recall that you should link the disclosure proof to the location request. You should (slightly) adjust the schemes you defined in the previous section to let them "sign" the location request.

We expect you to implement the zero-knowledge proofs yourself. You are not allowed to use external libraries to help you with the zero-knowledge proofs (e.g., the `zksk` library).

To help you with the integration, we provide you with six options of attributes. Your task is to choose one among them, and justify your choice. You still need to decide on the encoding of the attributes, and which ones to reveal and when.

Option 1

- User attributes = [secret key]
- Issuer attributes = [all subscriptions; username]

Option 2

- User attributes = [ ]
- Issuer attributes = [all subscriptions; username]

Option 3

- User attributes = [secret key; all subscriptions]
- Issuer attributes = [username]

Option 4

- User attributes = [all subscriptions]
- Issuer attributes = [secret key; username]

Option 5

- User attributes = [secret key]
- Issuer attributes = [valid subscriptions; username]

Option 6

- User attributes = [secret key; username]
- Issuer attributes = [valid subscriptions; username]

1.4 Requirements

Your report should answer the following questions:

- How did you map PS scheme to the SecretStroll system. How did you choose the set of attributes to use among the ones we propose? How did you choose which ones to reveal? Does this approach guarantee that minimum information necessary for the operation is revealed in each request?
- How did you use the Fiat-Shamir heuristic to make SecretStroll's zero-knowledge proofs non-interactive?
- How did you test the system? How would you assess the effectiveness of your tests? (Performing the assessment is not necessary.)
- How did you evaluate the performance of the ABCs? You need to report a fine-grained evaluation of communication and computation cost in key generation, issuance, showing the credential, and verifying. Each reported number must include both the mean and standard error of the mean.

The code that you deliver must:

- Work without any changes in the immutable part of the skeleton.
- Follow common coding practice like providing good documentation, having comments, having good variable names, etc.
- Include tests (use pytest format). You need to test at least two failure conditions in addition to a successful run.

1.5 Tips and Tricks

- The skeleton provides a command-line interface to interact with the SecretStroll system (e.g., register, query, etc.). Use `--help` or check the README file for more information.
- The skeleton uses docker containers to enforce isolation between the client and server.
- We strongly recommend using the `petrelic` cryptographic pairing library to implement PS credentials. The library exposes the BLS12-381 curve, which contains bilinear groups of the right type. You can find `petrelic` at `https://github.com/spring-epfl/petrelic/`. We recommend you read the documentation (available at `https://petrelic.readthedocs.io`) before starting. The multiplicative interface offered by `petrelic` will be the easiest to use. The `petrelic` library is bundled in the provided docker container and virtual machines.
- The skeleton handles the communication without knowing the application layer logic. The skeleton treats all communications as a byte array. This means that you have to (de)serialize your objects. We provide `serialization.py` as a 'petrelic' extension of `jsonpickle` to help you with the serialization of cryptographic objects.
- Normally the server should serve requests through HTTPS. The skeleton reduces this requirement to having simple HTTP connection to make the deployment and operation of dockers easier, however, you can assume encrypted communications.
- You should think about the following questions before reporting your performance evaluation. Does your performance evaluation depend on the hardware that you are running on? If so, what are your hardware specifications? What about the way you set up your testing environment? How many measurements do you need to compute the standard error?

2 (De)Anonymization of User Trajectories

Anonymous credentials clearly improve the privacy of your application: The service provider can no longer link location queries to user accounts. Unfortunately, despite all the complicated cryptographic machinery that you have built, there are still other information leaks that result in privacy risks for your users. As you can see in Figure 1, box 2, the service provider learns for every query a user's location and their subscription information. Even though each query is anonymous at the application layer, i.e., there is no user identity attached to it, the service provider still observes another piece of metadata that allows them to link queries of the same user: the source IP address.

You want to evaluate the remaining privacy risks that might result from this linkage before deploying the system to real users. To do so, you run a

privacy evaluation on simulated data. You have simulated the behaviour of two hundred users over twenty days in one geographic area. You now want to use the simulated dataset to test how bad is the privacy leakage due to the IP-level metadata, cleartext location and subscription information, and what you can do about it.

2.1 Data Description

The data for your privacy evaluation can be found in the folder named `privacy_evaluation`. You get two datasets as a result of your simulation: a dataset of queries `queries.csv`, and the POI database `pois.csv`.

Dataset of Queries The first dataset contains details of the location queries issued by simulated users. It is represented as a table where each row corresponds to a query which contains the following fields:

Column	Type	Description
IP address	`str`	Origin of the query
Latitude and longitude	`float`	Location queried by the user
Timestamp	`int`	Time when the query was issued
POI type filter	`str`	POI type the user requests (e.g., restaurants, cafes, museums, clubs, etc.)

For your convenience, the timestamp format is in **hours** from the beginning of the simulated data capture. The data capture begins on a midnight between a Sunday and a Monday.

POI Database The second dataset is your current database of POIs. It is represented as a table with the following fields:

Column	Type	Description
POI ID	`int`	ID of this POI
Cell ID	`int`	Grid cell ID (see below)
POI type	`int`	POI type (e.g., cafe, museum, club, etc.)
Latitude and longitude	`float`	Location of this POI

2.2 Server response

In SecretStroll, with each query a user asks the service for a list of POIs of a specific type (the ones they are subscribed to) in the vicinity of their current geographic location. To find nearby POIs, the service gets the user's location and returns all POIs within a fixed radius around the user's current location.

For your convenience, we provide a file called `query.py` (included in the `privacy_evaluation` folder) that implements this procedure. The function

`get_nearby_pois` takes in a user's location and the requested POI type and returns the `poi_ids` of all POIs within the defined distance threshold.

2.3 Privacy Evaluation

Your first task is to evaluate the privacy leakage of the service. Can you breach the privacy of simulated users in the dataset? Can you figure out where some users live, work, or what are their interests? What other information can you infer? You should assume that in this dataset *each IP address corresponds to a unique user.*

2.4 Defence

In the previous task, you have found attacks that can breach user privacy. But you also do not want to give up with building a privacy-preserving location service yet. You start thinking about defences that protect users' privacy against the attacks you found. You choose to implement a client-side defence such as those seen in the CS-523 lectures. In this task, you need to propose and evaluate one such defence.

For the purpose of this task, your defence **cannot** rely on users changing their IP address.

2.5 Requirements

First for the evaluation, the report has to include these points:

- Specification of your assumptions and adversarial models. Explain why you think that these assumptions and adversaries are useful for the privacy analysis.
- Definition of each attack strategy within the stated adversarial models. Describe for each adversary their attack strategy and how it is mounted.
- Demonstration of the attacks, possibly with plots or tables. Discuss their severity and the cost to mount them.

It is convenient to use a `Jupyter` notebook for this analysis. You cannot, however, substitute the report with the notebook; even if you include it in your submitted code, you still need to write the report.

Second, for the defence, the report has to include these points:

- Definition and description of the defence that you are proposing. Provide intuitions for why it should work and what kind of attacks it prevents.
- Experimental evaluation of the defence in terms of privacy. State your definition of privacy, and evaluate how does your defence impact this notion of privacy. Use the provided data for your evaluation. Remember that in a privacy evaluation you have to consider a **strategic adversary**. *Hint:* Your definition of privacy has to be quantifiable.

- Experimental evaluation of the defence in terms of utility of the service. State your definition of utility. What is the utility loss associated with using the defence? Use the provided data for your evaluation.
- Brief discussion of your defence. What are its privacy-utility trade-offs?

You do not need to find the best possible defence in terms of privacy-utility trade-off. What is important is that your write-up includes the points above, demonstrates a good understanding of adversarial modelling and privacy, and your ability to assess the level of privacy a defence can provide.

Non-graded Questions

We encourage you to think about these questions about your defence: What attacks it cannot prevent and do you think that is acceptable? When does it fail? What if your initial assumptions are broken? You don't need to include the answers in your report.

3 Cell Fingerprinting via Network Traffic Analysis

In the previous part, you saw that because users' IP addresses were collected, the service provider could infer some sensitive information about user activity. As a defence, you decide to randomize users' IP addresses so that it is harder to link a single user to an IP address. In order to effectively randomize the IP addresses, you set up the user and the service provider to communicate using Tor. Because the connection is encrypted, someone eavesdropping the network connection between the user and the service provider cannot observe users' location queries. However, after attending CS-523, you are concerned about the possibility that an eavesdropper could use network traffic metadata to infer the content of the queries. You want to evaluate whether such an attack is possible on your system. In this part, your goal is to perform a network-traffic analysis attack to recover sensitive information about users' whereabouts from their network traffic.

Grid For efficiency reasons, you have decided to implement a simpler version of finding nearby POIs given a user's location. Instead of searching for the closest points, you have divided the area from which users can query into a rectangular grid. Each cell of the grid has the same dimensions. Upon querying the service, the user receives all the POIs of the requested type *in the grid cell containing their location.* This process is illustrated in Figure 3.

3.1 Task

Your task is to perform a network-traffic analysis attack to *identify the cell ID that a user is querying for*, solely by analyzing the network traffic metadata.

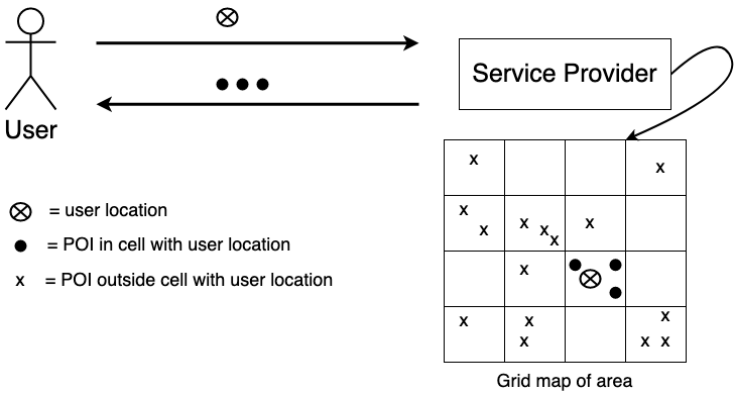

Figure 3: Obtaining POIs from user location. The provider divides the map of the area into a grid of cells, and responds with the POIs in the same cell as the user location.

The skeleton provided to you has a `client.py` script. You can supply a cell ID value as input to this script. The skeleton README contains instructions on how to run the script. The script simulates the behaviour of the app. To simplify this part, we choose to simulate that the application queries for a specific cell in the grid instead of a location. The grid in this case consists of 100 cells in total, i.e., the cell ID value is in the range 1-100. Note that, in reality, the number of cells would be much higher. The provider returns the list of POIs in the specified cell, and data about each POI in the list. The system is shown in Figure 4. There is Tor communication between the user and the service provider. You eavesdrop the communication between the user and the entry guard in the Tor network.

You need to develop an attack method to identify the cell ID queried by the application from the network traffic that occurs when the query is run. We expect you to use machine learning methods and build a classifier. You should collect network traces, extract features from them, and use those to train a classifier. Note that multiple cell ID queries for the same cell might return slightly different responses every time, due to the dynamic nature of the POI information.

While you are free to study the responses returned by the server, you cannot use the response content to identify the cell, only use network traffic.

3.2 Data collection

You are expected to collect network traces in order to train the classifier. Note that since data collection can be a time-consuming process, we recommend that you start early, and run it in parallel with Part 2.

Part 3 uses the finished system of Part 1 as a black-box. **Do not run**

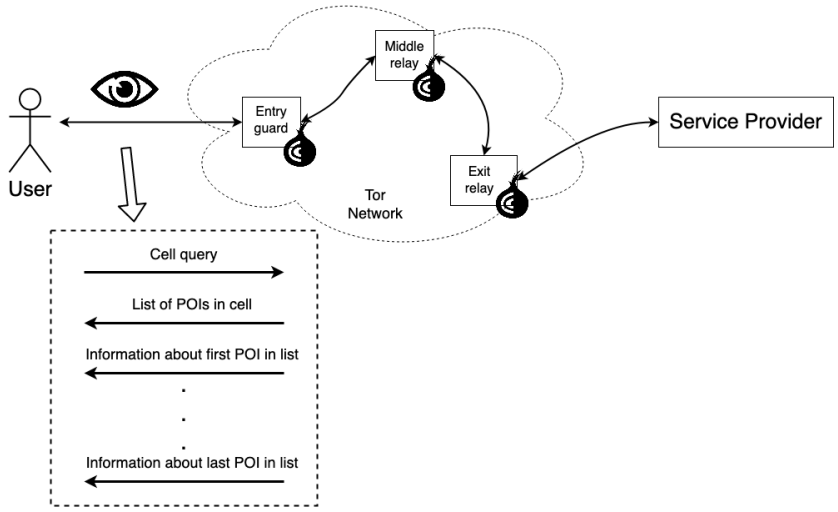

Figure 4: Scenario for Cell Fingerprinting. The app and the service provider communicate via Tor. The app sends a cell query. The provider responds with the list of POIs corresponding to the query, and data for each POI in the list.

data collection while you are implementing Part 1. If you make changes to Part 1 while running data collection, it could lead to changes in the trace behavior. If you have not completed Part 1 by the recommended period, and want to run the data collection independently of your implementation, please modify Part 1's skeleton to use an empty signature.

3.3 Use the skeleton

To help you build the classifier, we have provided a code skeleton, `fingerprinting.py`. The skeleton already contains code to implement a classifier. This skeleton is a guide and you can change/discard it as you see fit. For example, if you prefer to use another classifier, you are free to do so.

You are expected to write code to extract features from your collected traces to feed into the classifier. You are also expected to evaluate the performance of the classifier. The skeleton contains further instructions, and links to a few resources to help you get started. You are allowed to follow any procedure you like to perform feature extraction and evaluation.

3.4 Requirements

Your report should include:

- Description of the process you followed to train your classifier. You need to describe your trace collection and feature extraction.

- An evaluation of your classifier's performance (evaluation should be done using 10-fold cross validation), using standard performance metrics. A discussion on how well your classifier performed and what factors could influence the performance. **Note that the description of the process you followed is more important than the actual performance values of the classifier.**
- A short discussion on possible countermeasures to prevent this fingerprinting.

References

[1] CS 523. Lecture Notes: Pointcheval Sanders scheme. Available on Moodle.

[2] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *CT-RSA 2016*, pages 111–126, 2016.