CS-523 SecretStroll Report

Florian Delavy , Endrit Vorfaj

Abstract—**Handling location data, given its high dimensionality and sensitivity, necessitates careful privacy-preserving designs for Points of Interest (POI) recommendations. In this project, we tackle the challenge of processing sensitive user data securely. Our focus lies on three crucial aspects: the authentication system, data storage and type, and client-server communication. Our goal is to establish a resilient system that can address potential privacy concerns related to location-based services and ensure confidentiality under a specific threat model.**

I. Introduction

Location data, due to its personal nature and complexity, is highly sensitive and requires meticulous handling, especially in location-based recommendation systems. Our project strives to provide privacy-preserving mechanisms within these applications. Our specific focus revolves around enhancing user privacy during the authentication, data storage, and client-server communication processes. This includes deploying an attribute credential scheme for optimal subscription security and user anonymity. We'll also scrutinize potential weak spots in the client-server communication, even when using Tor for heightened security. Subsequently, we'll explore strategies to safeguard against exploit attempts by attackers.

II. Attribute-based credential

To maintain compact attribute size and boost security, the AttributeMap was structured as a dictionary which contains attributes encoded using a hashed, positive Bn value, ensuring simplicity and compatibility with the petrelic library. Each attribute in the AttributeMap is also provided an index to streamline attribute retrieval during subsequent searches. By adopting this method, only crucial data is transmitted and hashed, lowering the possibility of an adversary discerning exact subscription details. The decision to omit the client's username was made to increase anonymity further. To obstruct attackers from figuring out a client's subscription count, one potential approach would be to insert a random quantity of dummy attributes at the time of registration, creating noise. However, this specific implementation was not undertaken due to time limitations.

A. Non-interaction

In accordance with the ABC_guide, the Pointcheval-Sanders scheme was implemented, and it was recommended to apply the Fiat-Shamir heuristic to the corresponding Sigma protocol in step 4. This application serves to prove to the server that the computation of C has been performed correctly. Drawing from the exercises introduced during the week of Anonymous authentication, we can conclude that the Pedersen protocol, combined with the aforementioned heuristic, is suitable for our code.

Referring to the exercise solutions, we calculate the challenge c by hashing all the values of the public key, as well as the prover's commitment R and the commit value C. This allows the server to verify C without requiring additional challenges to be sent. In the second proof, the same principle is applied, but with the inclusion of the client's message (i.e., the latitude and longitude coordinates). This additional layer of security becomes necessary as the attacker would require this information to compromise the system.

B. Test

To thoroughly test the developed code, three files were created: `test_credentials`, `test_stroll` and `test_performance`. The `test_credentials` file focuses on verifying the functionality of the functions implemented in `credential.py`. It includes individual tests for small functions such as `generate_key()`,`sign()`, and `verify()`. Additionally, a comprehensive function tests the entire process of the PS protocol, considering various scenarios involving incorrect AttributeMap, public and secret keys, and invalid signatures. Random attribute generation is utilized in these tests, and each test is executed one hundred times to ensure reliable results and to mitigate any chance successes.

On the other hand, `test_stroll` employs a less random approach to assess the behavior of `stroll.py` with more realistic attributes and potential errors. The tests in this file also iterate one hundred times to detect any anomalies effectively.

Assessing the performance of our tests can involve checking coverage. However, since the scheme primarily involves integer multiplications, relying solely on simple statement coverage might overlook edge cases related to overflow and underflow. To address this, a more elaborate and resource-intensive approach could include verifying the values generated by the tests in `stroll.py` and `credential.py`. This approach, in addition to coverage analysis, ensures that all possible edge effects have been considered, thereby confirming the robustness of our code.

C. Evaluation

To gather performance statistics on our code, we have developed a file named `test_performance`. This file conducts key generation, issuance, signing, and verification operations to assess their efficiency. It focuses solely on the code's performance without involving real communication, as network performance could introduce additional variables. The data collected for the tests was obtained from executing them on a x86_64 architecture with 4 cores.

1) Key Generation: The `test_key_gen()` evaluates the performance of the `generate_ca()` function in the server. Since the server only needs to send the public key to the client, we focus on measuring the size of the public key as an indicator of the communication cost. The function is called with different numbers of initial attributes.

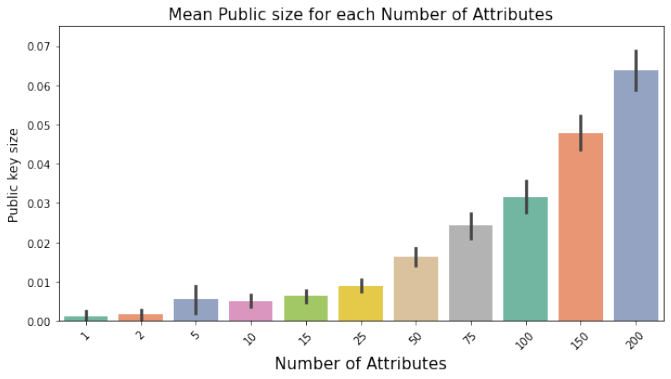


Fig. 1: Mean of the time of `generate_ca()` with it's standard deviation, in seconds

Nb Attributes	1	2	5	10	15	25
Nb Bytes	1088	1530	2856	5066	7276	11696

Nb Attributes	50	75	100	150	200
Nb Bytes	22746	33796	44846	66946	89046

TABLE I: Size of the public key generated by `generate_ca()`

As seen in the previous plot and table, we can see that the time of this function has a linear growth, as the size of the public key. The size was also constant, thus has an std of 0.

2) Issuance Request: The `test_issuance_req()` evaluates the performance of the `prepare_registration()` and `process_registration_response()` function of the client as well as the `process_registration()` function of the server. Since we are primarily interested in the data exchanged between the client and server, we capture the size of the blind signature and issuance request in bytes. This allows us to assess the communication cost of these functions.

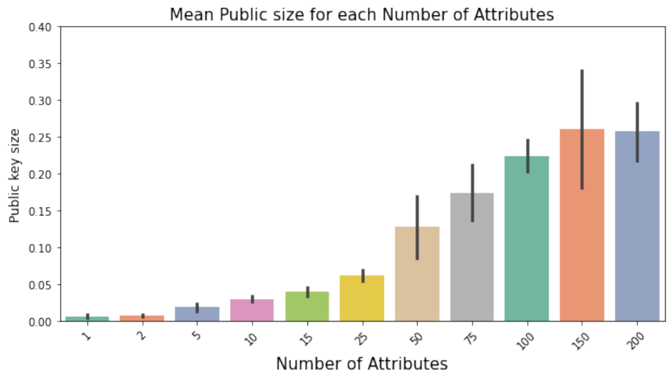


Fig. 2: Mean of the time of the issuance with it's standard deviation, in seconds

Nb Attributes	1	2	5	10	15	25
Issue Request	553	657	969	1489	2014	3084
Blind Sign	412	412	412	412	412	412

Nb Attributes	50	75	100	150	200
Issue Request	5689	8314	10939	16239	21539
Blind Sign	412	412	412	412	412

TABLE II: Size of the Issue Request and the Blind Signature generated by issuance

The test `test_issuance_req()` demonstrates that the execution time exhibits linear growth with a plateau at around 270 milliseconds. Interestingly, the size of the blind signature remains constant across different numbers of attributes, with a standard deviation of 0. This behavior can be attributed to the design of the test, where the client registers and makes requests for all attributes. As a result, the issuer AttributeMap is always null, minimizing the size of the blind signature to its maximum extent.

3) Showing: The test `test_showing()` measures the performance of the the `sign_request()` function of the client, capturing the size of the disclosure proof, in bytes.

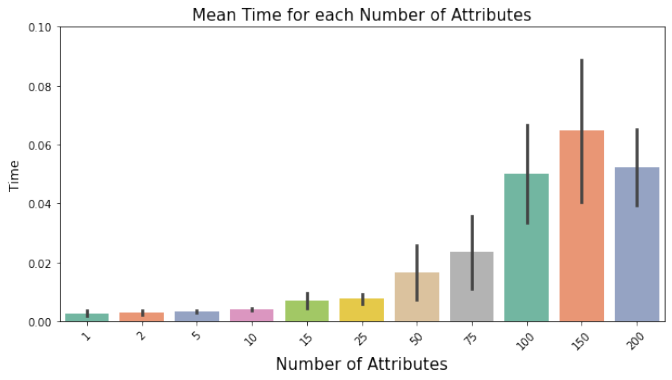


Fig. 3: Mean of the time of `sign_request()` with it's standard deviation, in seconds

Here we can see a curious peak at a 100 attributes. Furthermore, the size of the disclosure proof for any number of attributes is equal to 659 bytes, with the same explanation of the size of the blind signature, as discussed previously.

4) Verification: The test `test_verification()` measures the performance of the the `check_request_signature` function of the server. We can see in the figure below a linear. time

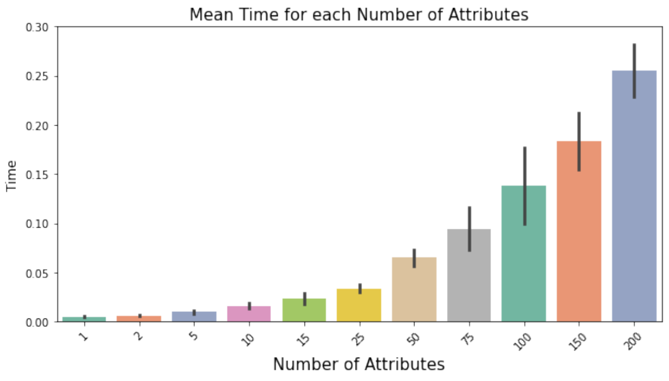


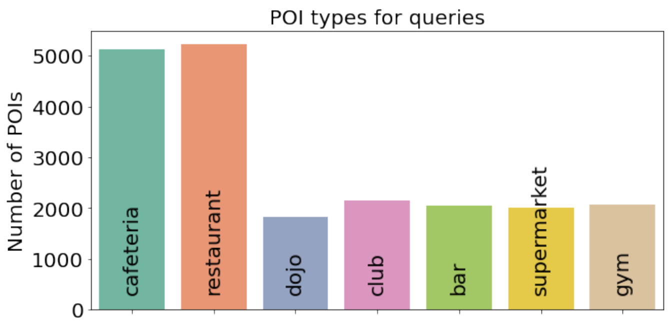
Fig. 4: Mean of the time of `check_request_signature()` with it's standard deviation

III. (De)Anonymization of User Trajectories

A. Privacy Evaluation

For this project, we were supplied with two datasets: queries.csv, which includes user queries along with precise location data, and pois.csv, which provides the grid cell ID for each query, among other information. The distribution of Points of Interest (POI) types present in these datasets is depicted in Figure 5.

The objective is to utilize this data to investigate potential breaches of user's privacy by identifying patterns that might disclose sensitive information, such as users' home addresses, workplaces, or personal interests. We make the assumption that an adversary, such as an Internet Service Provider (ISP), could have access to the network traffic on the server-side, thereby obtaining the same data provided to us.



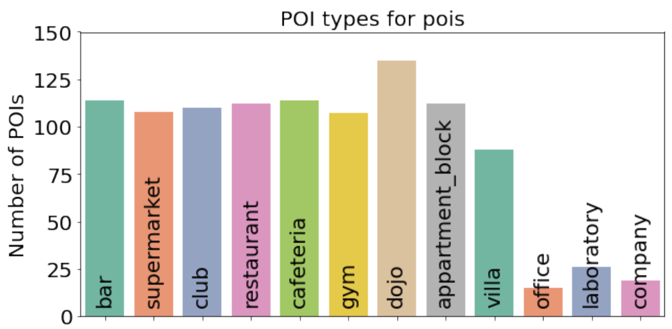


Fig. 5: Data-set description

This gives us the basis for a de-anonymization attack based on the notion that knowing a person's home and work address could with high probability lead to an identification alias. Given the uniqueness of each IP address to a user, we could exploit the latitude and longitude information to pinpoint user locations. Subsequently, temporal patterns in user activity, derived from the timestamp data, may reveal insights into their movements.

We propose using the time of the day (specifically, night-time) and the day of the week (weekday or weekend) as proxies for determining whether a particular location is likely a user's home or workplace. A commonly observed working time interval, such as 8 AM to 5 PM, provides a useful heuristic for this distinction.

The initial step in our strategy involves cleaning and transforming the data. We utilize the timestamp to extract the hour of the day (using timestamp % 24) and to calculate the number of hours elapsed in the week (via timestamp % 168). This allows us to assign a 'day_type' to each query (either 'weekday' or 'weekend').

The second phase of our strategy seeks to identify a user's frequent locations. Here, we take into consideration the standard working hours (8 AM to 5 PM) and weekdays as criteria for identifying likely workplaces. Similarly, we use night hours and weekends to deduce probable home locations.

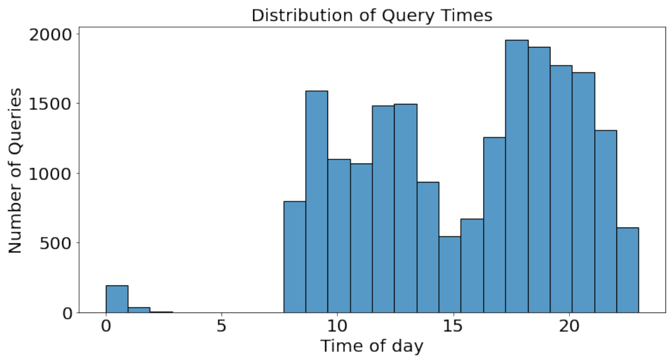


Fig. 6: Frequency of queries at different times of a day

By counting the number of queries made during these periods, we are able to form an idea regarding whether a particular location is likely to be a home or work address.

Figure 6 presents the frequency of queries at different times of the day, enabling us to distinguish between work and home times. Through this approach, we aim to tackle on the potential privacy implications of our data and in the subsequent section we propose countermeasures to safeguard user information or at least mitigate the risks.

B. *Defences*

In this part we aim to introduce a defense mechanism that modifies data to provide anonymity and reduce linkability. Our approach is founded on the premise that the existing location and timestamp data can be exploited to recover user identities, as seen in the previous section. This mechanism is expected to mitigate attacks that leverage precise location and timestamp to compromise user privacy.

The defense we propose involves the modification of how location data is managed. Instead of storing the exact location (latitude and longitude) of each user's Point of Interest (POI) query, the server could save this data using a grid cell ID system. The geographical space is partitioned into grid cells, with each cell being uniquely identified by an ID. In technical terms, the longitude and latitude are given to us with extreme accuracy (down to the millimeter). If we use grid IDs, we split the area into a 10 by 10 grid space and the total size of this area is 0.10 longitude and 0.077 for latitude which is equivalent to 1km and 770m respectively in real map. Consequently, the server would simply need to match the user's location with a corresponding grid cell ID when responding to a POI query. This approach retains service utility while significantly enhancing anonymity.

We define privacy as the degree of location indistinguishability and unlinkability as well as user activity anonymity. By replacing precise location data with grid cells, we increase the indistinguishability, making it more difficult for an adversary to accurately pinpoint a user based on their queries. We when we used grid cells instead of exact coordinates, the size of the anonymity set increased. Specifically, we found that many home/work pairs had more than one associated IP address, thereby increasing anonymity and privacy for users.

	Home	Work
Grid-Cell avg:	3.63	5.63
Lat & Long avg:	1.25	3.57
Original Timestamp:	1.092	1.092
Reduced Timestamp:	3.44	3.44

TABLE III: Average anonymity-set size when using: grid cells vs lat & long and reduced vs original timestamp

In Table III we see that we have greater anonymity set size when we use grid cells for both home and work address compared to using exact location. Also we see that the anonymity is greater we only consider the day type instead of exact time.

Under our proposed defense, the service's utility should remain unaffected for most applications, as there's no significant additional benefit in storing precise location data over storing grid cell IDs. However, in specific use cases, like military applications where exact locations might be needed, using grid cells could lead to a decrease in utility. However, there's an inherent privacy-utility trade-off involved. While the use of grid cells and the omission of timestamps can increase privacy, these modifications might adversely impact the service's utility in certain scenarios. For instance, removing timestamps might limit an application's ability to offer time-dependent services or insights.

IV. Cell Fingerprinting via Network Traffic Analysis

A. *Implementation details*

The *first* step of implementation is to collect the data that we'll be using to train our model. We used `tcpdump` in a shell script to capture the packets over all grid cell IDs and we create a `.pcap` file for each captured trace. We noticed that the captured traces contain many packets (ACK packets, ARP, DNS, etc) that are practically useless for our model as they do not provide information about the exchanged data. It is for this reason that we only keep the TCP packets from every capture, precisely we keep the packets that have a TCP payload and the header (64 bytes). Once we have this data, we do some data cleaning and transformation so that we end up with essential features that could increase the accuracy of the model. To achieve that, we use z-score (which tells us how many standard deviations a data point is from the mean in a distribution) to detect any outliers (z-score ¿ 3).

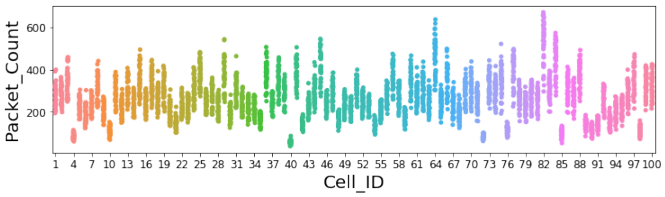


Fig. 7: Cleaned data with z-score

In Fig 7, we can see the clean collected data which contains no outliers. Based on the count of the initial traces, we went from having 3604 traces to 3571, so we had 33 outliers which were removed.

The *second* step in the implementation is to extract important features from the cleaned data. The way we do this is by first getting the very first packet's timestamp for each trace and then we save a tuple of (packet_size, time) in a list which represents once trace. The way we distinguish between a packet being sent and one being received is that we multiply by 1 the packet size if the IP address is private (meaning it is coming from a client) otherwise we multiply by -1. Since the communication between client and server is not "stop and wait", we can have cases where client sends multiple packets of same payload before receiving from the server and we end up with way too much data to store in a raw format. Since we can distinguish between packets being sent and received, we decided to group them into rounds where each round represents and entire payload being sent. This was achieved by grouping sequential packets of same size and adding them to form one payload. To have a ”square” size dataframe, we needed to know the maximum number of rounds we can have and for that we have a `get_max_rounds()` function which calculates that. In the end we end up with a data-frame of the form:

Cell_ID
Packet_Count
Time
Number_Rounds
Round_Size[253]
Round_Time[253]

TABLE IV: Final dataset that we use for the model

In Table IV we have all the rounds (253) and we store the payload sent and received for each round as well as the time when the exchange happened. This will serve as the features for the training model.

B. Evaluation

The model we use for our data is `RandomForestClassifier` which comes from the *sklearn* python library. To evaluate the model we have 3 performane metrics:

- Global Accuracy: this measures the average of predicted labels that correspond to the correct test labels.
- Top 2 accuracy: this metric allows for more prediction margin as it is not a one to one prediction-test but it considers a prediction to be correct if one of the top 2 predictions for a label are correct.
- Top 10 accuracy: Similar to Top 2 with the sole difference being that it considers the top 10 most probable predictions.

We initially trained the model in all the features and we had the following results:

Measurement	Score
Accuracy	0.66
Top 2	0.73
Top 10	0.87

TABLE V: All features metrics

In our second try we removed the round time all together and we had slightly better results:

Measurement	Score
Accuracy	0.73
Top 2	0.79
Top 10	0.90

TABLE VI: No round time metrics

Finally, we drop every other feature and we only keep the round size and the cell id. We get the following results:

Measurement	Score
Accuracy	0.747
Top 2	0.80
Top 10	0.90

TABLE VII: Only Round size feature

C. Discussion and Countermeasures

As we can see in Table V, not all features are needed to increase the accuracy of our model. That is because same as we did with traces where we only keep the TCP packets and we drop packets that don’t aid to our application, in a similar way we also have redundant features here that could worsen our model’s accuracy. One of them being the round time since as we mentioned before, payloads are sent in multiple packets and they might not be of the same size, so the time is not a reliable features in this case. Indeed we notice in Table VI that the accuracy of our model increases when we drop the round time feature. At the end, since the data exchanged is encrypted, it makes sense that the most important feature is the round size as it is the only feature that gives us insight on the exchanges between client and server. We can see this in Table VII where we notice that the metrics have increased compared to the others.

Considering countermeasures against this model requires revisiting our approach to feature extraction. The high dimensionality of the data is attributable to the sequence of round sizes, which reveals the number and types of Points of Interest (POIs) and their respective ratings. To mitigate this, we could shuffle the list of returned PoIs from the server, thereby ensuring that the client does not query them in the same order each time. This modification would not compromise utility. An alternative approach might be to limit the amount of rating information sent for each POI. Providing key statistics rather than comprehensive ratings would both reduce and normalize the size of each round. Lastly, the ”round” nature of the process reveals information. We could prompt the client to query multiple POIs simultaneously, rather than in successive rounds, to make it more challenging to discern the signal of a round query.

V. Contribution

F.D has implemented the code of `credential.py`, the three tests of part 1 and has done the capture for the part 3. E.V have made the analysis of part 2 and designed the code of `stroll.py`. The two of us has worked on the extraction of the features and the training of the model of part 3 as well as working of the report.

References

[1] Advanced Topics on Privacy-Enhancing Technologies CS-523 Anonymous Authentication Exercises - Solutions, https://moodle.epfl.ch/pluginfile.php/3056481/mod_resource/content/3/anon_exe_sol.pdf