CS-523 SecretStroll Report

Baccari Mariem, Borges Trivelli Caetano, Gasmi Donia

I. Introduction

This project explores privacy challenges in the SecretStroll location-based service. We first develop an access control mechanism using attribute-based credentials. We then investigate how query metadata can be exploited to infer sensitive user information through mapping, identification, and interest-based attacks. To mitigate these threats, we implement a geo-indistinguishability-based defence and evaluate its privacy-utility trade-off. Finally, we explore machine learning-based fingerprinting as a broader threat to user anonymity.

II. Attribute-based credential

The SecretStroll system requires storing a username, a set of subscriptions and ideally the user's secret key as attributes in the credential. The Pointcheval-Sanders scheme fixes a size for messages to sign and attributes to store at the moment of key generation. Option 1 of those proposed in the handout was chosen for the following reasons:

- It includes a user-defined secret key as a user-defined attribute, allowing for its selective (non)disclosure.
- It includes all subscriptions as an issuer-defined attribute, which (1) keeps the amount of attributes constant for all users (which wouldn't be the case with valid subscriptions), making implementation through the PS scheme of fixed attribute number possible, and (2) gives the issuer control over the values of these attributes, effectively preventing malicious users from potentially assigning themselves arbitrary active subscriptions.
- It includes the username as a issuer-defined attribute, which can be necessary for database-related purposes, such as ensuring uniqueness of usernames in the platform.

The only attributes that are revealed when the client makes a request to the application are the subscriptions of the types of POIs requested. Thus, the server only learns, for every type in the request, if the user has that subscription (`1`) or not (`0`). This ensures the minimum amount of information is given to the server while preserving application functionality.

A. Fiat-Shamir heuristic

The Fiat-Shamir heuristic was used to implement all necessary zero-knowledge proofs in non-interactive form. It removes one round of communication in proofs of Pedersen-like commitments by avoiding having to request a challenge from the verifier, and computing this challenge deterministically through a hash function based on the previously computed commitment and public information present in the context of the scheme. It is of vital importance that all relevant public information and the proof commitment be given as input to the hash function as to avoid malicious collision of challenges in different executions of the protocol. In our context, all ZKPs used in the ABC implementation were of the form

$$\mathrm{PK}\left\{(t,(a_i)_{i\in S}) : C = b_0^t \prod_{i\in S} Y_i^{a_i}\right\}$$

where b_0 and Y_i are called "basis" elements and are public. Thus, the elements taken as input for the hash to create the Fiat-Shamir challenge are: all basis elements (which in the case of the first commitment in the issuance protocol includes the group generator $g \in G_1$), the Pedersen-like commitment of which knowledge wants to be proven, and the specific proof commitment $T = b_0^{t_0} \prod_{i\in S} Y_i^{t_i}$.

The hash function used is `shake_256`, an extendable-output function from the SHA family that allows for output size specification, which helps avoid statistical bias in computed challenges when reducing modulo the group order. Additionally, a Schnorr's signature-like mechanism was implemented to sign messages by binding them to a credential through the disclosure proof. For this, the message is added to the hash function input, and the verifier includes it in the classic non-interactive proof verification too.

B. Test

Different tests were created and used to make sure specific components functioned correctly. For the general ABC scheme workflow, we highlight the following three tests:

1) `test_correct_credential()`: verifies that the normal execution path of key generation, credential issuance, showing and verifying is correct.
2) `test_incorrect_credential()`: verifies that a credential issued by issuer A does not pass verification with a verifier that uses a public key different from A's.
3) `test_malicious_attribute()`: verifies that the user cannot modify a issuer-defined attribute in the credential, preventing malicious users from e.g. attempting to get a free subscription for one POI type. Note that users also cannot modify user-defined attributes after credential issuance, but since they can control them before issuance and simply make another issuance request this is considered out of scope.

Future work to assess the effectiveness of tests could include

- Evaluate its coverage of branching and edge cases using tools to oversee code execution.
- Evaluate its funcionality against different adversarial inputs coming from all involved parties.
- Introduce mutation testing to ensure tests are correct, precise and effective.

Attribute list size	**4**	**10**	**20**	**40**	**60**	**100**
Key Generation	1.61 ± 0.005	3.21 ± 0.005	5.89 ± 0.011	11.20 ± 0.010	16.60 ± 0.053	27.20 ± 0.027
Issuance	1.03 ± 0.008	1.04 ± 0.002	1.11 ± 0.008	1.18 ± 0.008	1.27 ± 0.005	1.45 ± 0.006
Showing	7.48 ± 0.014	15.29 ± 0.010	28.45 ± 0.034	54.88 ± 0.209	80.80 ± 0.049	133.28 ± 0.055
Verification	7.39 ± 0.015	15.27 ± 0.018	28.57 ± 0.116	54.86 ± 0.089	80.98 ± 0.070	133.42 ± 0.090

TABLE I: Mean **computation time** with standard error of the mean in miliseconds for each subroutine and attribute list size.

Attribute list size	**4**	**10**	**20**	**40**	**60**	**100**
Key Generation	2914 ± 45	5075 ± 49	8695 ± 74	15935 ± 92	23175 ± 106	37658 ± 141
Issuance	1131 ± 12	1209 ± 14	1359 ± 16	1659 ± 20	1959 ± 23	2559 ± 25
Showing & Verification	1110 ± 10	1209 ± 13	1338 ± 15	1638 ± 19	1938 ± 22	2538 ± 27

TABLE II: Mean **communication cost** with standard error of the mean in bytes for each subroutine and attribute list size.

C. *Evaluation*

We evaluate our ABC framework by measuring **computation time** and **communication cost** of specific subroutines like key generation, issuance, signing and verification.

1) Evaluation Setup: Experiments were conducted on the provided docker containers running on a student laptop with an Intel Core i5 CPU (13th Gen, 4.6 GHz, 16 cores, 16GB RAM). Note that communication occurs locally and does not reflect real-world latency or bandwidth limitations.

2) Measurement Repeatability: To ensure statistically reliable results, **each experiment was conducted twenty times**. We present both the mean and standard deviation to capture performance trends. Repeating the experiments mitigates the influence of system noise.

3) Results: All results are presented in Tables 1 and 2, for computation and communication costs respectively. We present more details on each specific subroutine in the following subsections.

a) Key Generation: Note that the key generation algorithm itself is executed solely by the signer, so there are no communication costs associated to it. However, we also consider in this section the key retrieval executed by the client as communication costs.

b) Credential Issuance: This subroutine considers issue request creation, issue request signature, and signature unblinding. It is worth noting that the communication cost in the request creation scales with the amount of subscriptions the users registers with. To illustrate this communication cost, in the experiments conducted the user registers with all subscriptions.

c) Credential Showing: This subroutine considers credential disclosure proof computation. The communication costs considered are those associated to the `loc` POI request, i.e. sending the credential that signs the request message. Note that this operation is related to both credential showing and verifying, so communication costs for these two subroutines were grouped.

d) Credential Verification: This subroutine considers credential disclosure proof verification. Communication costs were grouped with credential showing for the reason stated previously.

e) Extra: Credential Showing with varying number of disclosed attributes: An interesting case is when the number of attributes being disclosed is also varying. For the previous experiments, all requests were made using all the subscriptions the user had. However, as the amount of requested POI types increases, the amount of hidden attributes decreases, affecting the communication costs significantly. Figure 1 illustrates this behavior.

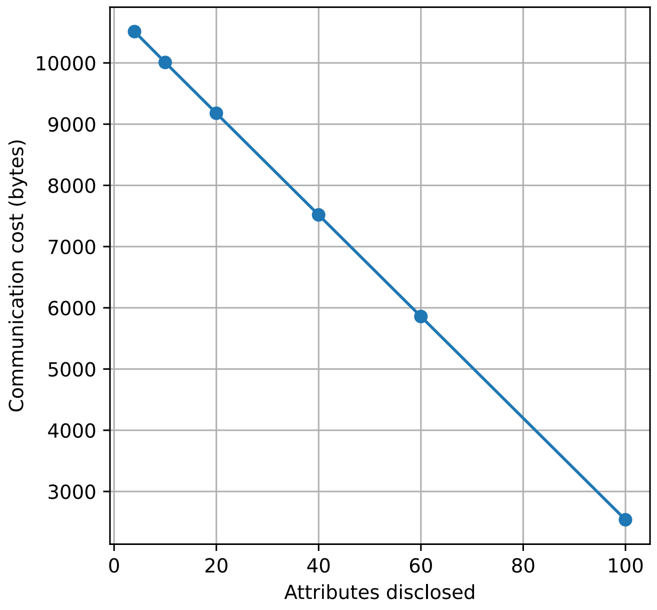

Fig. 1: Effect of amount of attributes disclosed (amount of POIs requested) on communication costs

III. (De)Anonymization of User Trajectories

A. *Privacy Evaluation*

We assume an honest-but-curious adversary with access to query metadata: IP address, timestamp, location, POI type, and the server's responses. Each IP is assumed to correspond to a unique user, as per the project handout. The goal of the attacks is to undermine a user's privacy and find out sensitive locations (home/work) and interests. All these attacks are implemented in the `evaluation.ipynb` and have a linear complexity in the number of queries ($\mathcal{O}(Q)$). More details about the analysis as well as the full code are available there. Some are ommitted here to stay concise.

1) Mapping User Movement: In this attack, the adversary knows the IP address and attempts to discover a user's home and work by clustering queries over time and space. For a selected user (IP = 146.71.112.211), we observe two frequent locations as shown in figure 2: one matching a residential area queried during evenings and weekends, and another near Renens Gare, active during weekday lunchtimes. These temporal-spatial patterns suggest probable home and work locations.

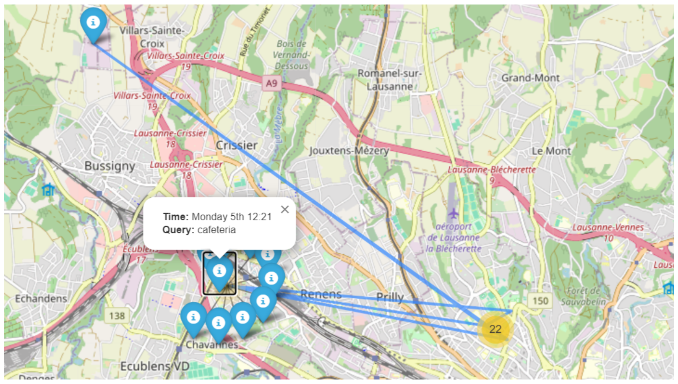

Fig. 2: User query locations (IP = 146.71.112.211) over one week

2) Linking Identities to IPs: This attack reverses the previous one: the adversary starts with known real-world locations (home/work) and tries to identify the corresponding IP in the dataset. Using spatial and temporal filters to look for queries around the home location on the weekend mornings or weekday evenings and for queries around the work location during the week days around lunch time, we retrieve matching IP addresses. With accurate input (e.g., 100m margin) we managed to obtain the correct IP. With less accuracy of the background knowledge or a denser / bigger data set, we might find multiple users corresponding to the criteria, making this attack harder to deploy.

3) Revealing User Interests: In this attack, given an IP (perhaps using previous attacks or as background knowledge), we try to build a behavioral profile based on query patterns and POI types (interests and frequented places). Using the queries metadata, we can see how often the user queries for a specific POI type and when they do so. For example, if they query for "restaurant" a lot, we can assume that they are a foodie. If they query for "bar" or "club" a lot, we can assume that they are interested in nightlife. We can also use the POIs file to see which specific POIs are returned.

We ran our script for the same user as previous attacks and plotted a pie chart to demonstrate the prevalence of each POI type in the queries as shown in figure 3. We can see that this user is interested in nightlife activities (clubbing), practice some martial arts (as the query for dojos) and goes to the gym. We can go one step further and use the servers' responses to find out which exact clubs, dojos or gyms the user is most likely to frequent, assuming that they end up going to the places that are returned by the server. For example, we retrieved the resulting locations for the type "club" and put them on a map as shown in figure 4.

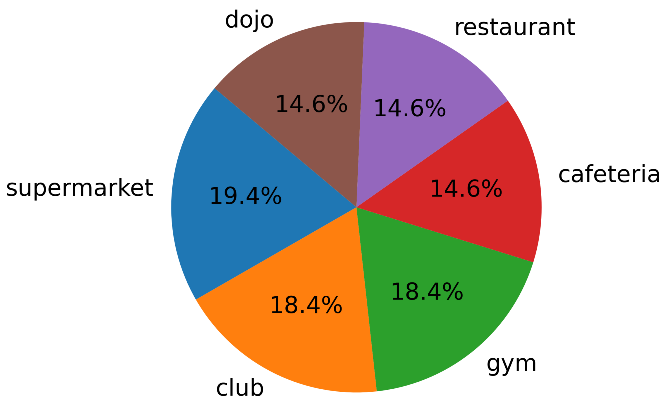

Fig. 3: Pie chart of queries POI types for user IP = 146.71.112.211

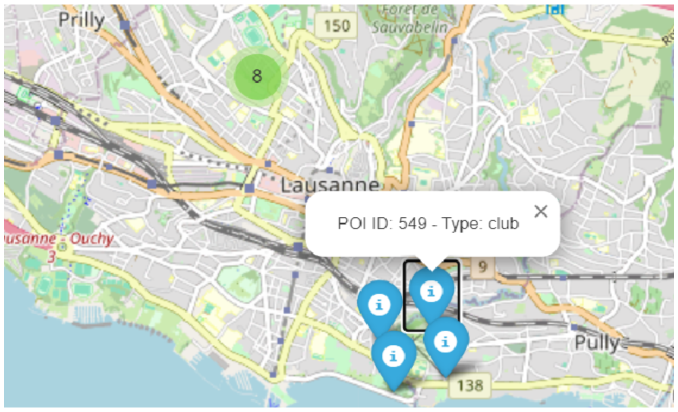

Fig. 4: Map of resulting POIs for type "club" for user IP = 146.71.112.211

4) Discussion: We demonstrated three attacks in which adversaries can undermine user privacy solely relying on query metadata and varying background knowledge. They were all computationally cheap and scalable which makes them realistic and dangerous. In particular, the first attack shows how real-life identification is possible through passive access to such data by anyone who can know your IP address including an ISP, state-level surveillance agencies or even a stranger on the internet who somehow obtained the IP. Similarly, the second attack shows how partial knowledge can de-anonymize a user in a dataset especially in low-density areas. Finally, the last attack helps us infer personal preferences which allows profiling users based on their routines and habits which can serve purposes as trivial as customized advertising and as dangerous as targeted stalking.

B. Defences

The presented attacks rely on access to precise user locations. To mitigate this, we implement spatial obfuscation using the geo-indistinguishability framework [1]. The idea is to add noise to user locations before they are sent to the server, so that nearby locations are reported with similar probabilities. This is achieved using a planar Laplace mechanism, with noise drawn from a 2D Laplace distribution parameterized by ε. Smaller values of ε yield stronger privacy guarantees at the cost of utility, and vice versa.

Since our coordinates are expressed in degrees, we convert between meters and degrees to apply the mechanism correctly.

We empirically tested the impact of this defence on the three attacks. As ε decreases, it becomes harder to infer home and work locations accurately, and identifying specific users becomes less reliable. However, since query types and resulting POIs are unaffected, the attack on user interests remains largely feasible. Mitigating this would require more advanced defences, such as adding dummy queries or generalizing POI types, which are beyond the scope of this report. A more detailed analysis of the defence's impact on these attacks with variable parameters can be found in the Jupyter Notebook.

1) Defence Evaluation: To evaluate the defence, we varied ε and repeated each experiment 5 times before computing the average. For each setting, we estimated user locations and computed:

- **Precision:** Proportion of correctly inferred locations (TP / (TP + FP)).
- **Average Distance Error:** Average distance between true and inferred locations.

As shown in Figure 5, precision improves and distance error decreases logarithmically as ε increases, indicating reduced privacy.

For utility, we compare the POIs returned for obfuscated queries against the original set using the Jaccard Index (intersection over union). Higher values indicate more similarity. Figure 6 shows that utility linearly increases with ε, reflecting the privacy-utility trade-off.

2) Discussion: We showed how geo-indistinguishability provides tunable privacy guarantees by adjusting ε. With low ε, attackers struggle to infer locations, but users receive less relevant POIs. High ε preserves service quality but weakens privacy. A reasonable deployment could let users select their privacy level, or set defaults based on application context. Our results quantify these trade-offs to give a basis for fine-tuning and demonstrate the practical viability of this defence.

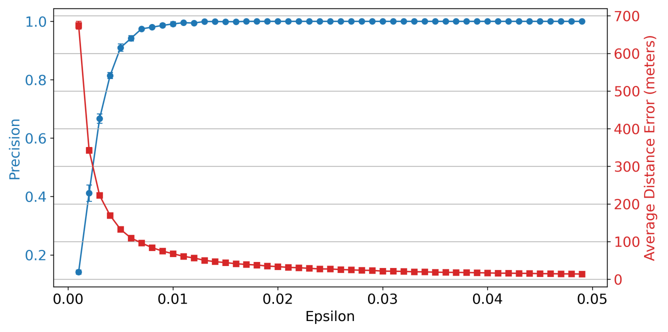

Fig. 5: Precision and average distance error as a function of ε

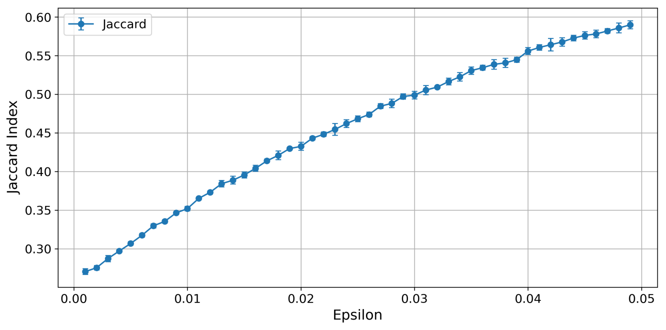

Fig. 6: Jaccard index as a function of ε

IV. Cell Fingerprinting via Network Traffic Analysis

A. Implementation Details

1) Trace Collection: We collected network traces using the SecretStroll infrastructure, simulating an adversary who can eavesdrop on encrypted Tor traffic between the client and the Tor entry guard. The attacker cannot access decrypted content or server-side responses.

An automated Bash script (`collect_traffic_traces.sh`) ran inside the client container, issuing 100 `restaurant` queries per grid cell (IDs 001–100) using `client.py grid <cell>`. We captured traffic with `tcpdump` on the `eth0` interface, excluding Tor control traffic (`not port 9050`).

Each trace was saved as a separate `.pcap` file under `/trace_dataset/cell_XXX/trace_YYY.pcap`. To ensure circuit diversity, the Tor service was restarted after each cell batch. This process yielded **10,000 packet captures** for traffic-only analysis.

2) Feature Extraction: Using a Python script and `pyshark`, we parsed each `.pcap` and extracted the following features:

- Total packets and bytes
- Min, max, and average packet size
- Duration
- Mean and standard deviation of inter-arrival times

Traces were processed in batches of 20 per cell to reduce memory use, and valid feature vectors were saved to CSV along with their cell ID. The final dataset contains 10k labeled samples, each with 8 features. Corrupted or empty traces were excluded.

3) Classifier Training: We trained a Random Forest classifier on the extracted features using 10-fold stratified cross-validation. This model was selected for its robustness, low overfitting, and strong performance in prior traffic analysis tasks. The training pipeline was implemented with `scikit-learn`.

B. Evaluation

1) Overall Performance: We evaluated our model using standard multi-class classification metrics: accuracy, precision, recall, and F1 score. As each cell ID is equally represented in the dataset, weighted averages provide a balanced view of performance.

Metric	Score (%)
Accuracy	89.8
Precision (weighted)	89.9
Recall (weighted)	89.8
F1 Score (weighted)	89.8

TABLE III: Overall classification performance of the Random Forest model.

2) Per-Cell Performance: To better understand class-wise behavior, we computed accuracy individually for each of the 100 cell IDs. The results, shown in Figure 7, reveal a strong skew toward high performance: over half of the cells were classified with greater than 95% accuracy, and several reached perfect accuracy.

However, a small subset of cells remained challenging, with notably lower performance. These cases may stem from overlapping traffic patterns, possibly due to spatial proximity or functional similarity that make it difficult for the model to distinguish between certain locations.

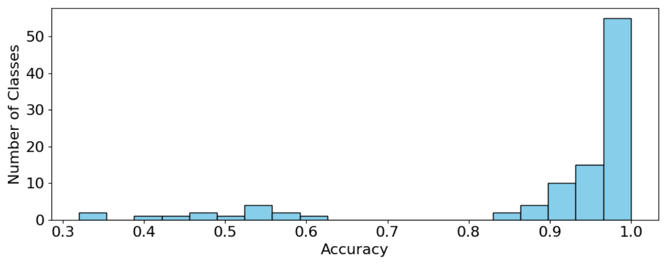

Fig. 7: Distribution of per-class accuracies across all cell IDs.

C. Discussion and Countermeasures

1) Proposed Countermeasures: To mitigate fingerprinting attacks based on traffic metadata, several strategies can be considered:

- **Traffic Obfuscation:** Adding padding, fragmentation, or dummy packets can mask discriminative features like packet size and byte count. Our own results show that even light noise reduces classifier performance significantly.
- **Query Batching and Timing Jitter:** Sending queries in batches or introducing random delays disrupts timing features such as inter-arrival time and trace duration.
- **Uniform Response Sizes:** Standardizing server response sizes removes variability in byte-level metadata, reducing susceptibility to volumetric analysis.

These techniques come with performance and latency trade-offs, and must be tuned carefully. Still, they provide promising first steps toward making traffic less distinguishable.

2) Evaluating Obfuscation via Feature Perturbation: We used Random Forest's built-in feature importance analysis to identify which features contributed most to classification accuracy. As shown in Figure 8, packet size and byte-related features were the most influential—making them the most privacy-sensitive.

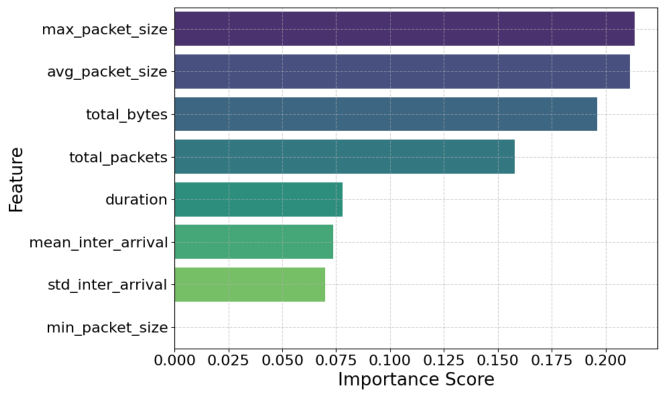

Fig. 8: Feature importance scores from Random Forest classifier.

To test the effect of masking these features, we added Gaussian noise to `max_packet_size`, `avg_packet_size`, `total_bytes`, and `total_packets`, simulating padding, dummy data, and heartbeat packets. We then retrained the model using the same 10-fold protocol.

The result: the weighted F1 score dropped from **0.89** to **0.47**, demonstrating that even modest, plausible perturbation of critical metadata can significantly reduce fingerprinting accuracy.

These findings validate targeted obfuscation as a concrete, effective mitigation—especially when informed by feature attribution techniques.

V. Individual Contributions

Caetano worked on part 1, Mariem on part 2 and Donia on part 3.

References

[1] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: differential privacy for location-based systems," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 901–914. [Online]. Available: https://doi.org/10.1145/2508859.2516735