CS-523 SecretStroll Report

Francesco Intoci, Elvric Trombert

***Abstract*—This paper aims to describe our design choice for the ”SecretStroll” LBS system, together with a Privacy Evaluation of the system itself.**

I. Introduction

SecretStroll is a Location Based System (LBS), which aims to provide users with information about Points of Interest (POI) near their location, satisfying user-specified criteria.

LBS are prone to be privacy-sensitive systems. If not implemented following a *Privacy by Design* paradigm, they can leak many user information including but not limited to: user location, user movement patterns and user centre of interests. The SecretStroll system was designed by identifying layers at which the system could potentially leak information and implementing privacy-preserving mechanism to try mitigating such leaks. We also conducted a privacy evaluation of the system.

A. Attribute-based credential

SecretStroll uses an Attribute-based credential protocol to authorize users to fetch POIs of a given type, for which they must have subscribed to. The ABC protocol serves two purposes. First, it allows the server to check whether a user request is valid, w.r.t the payment status of the user for that service. Secondly, it guarantees user anonymity since the server will not be able to link a credential to a user between subsequent requests.

B. (De)Anonymization of User Trajectories

ABC alone are not enough in order to guarantee users privacy. In particular, if the server is able to observe users metadata, such as their IP Address, the ABC scheme clearly looses its *unlinkability* property. We analyzed the consequences of IP address linkage on users' privacy via attacks mounted by an adversary with access to such metadata. We then constructed and evaluated a defence mechanism aimed to mitigate such attack.

C. Fingerprinting attack on Tor

SecretStroll exploits the Tor Network in order not to leak metadata at Network level. Nevertheless, we evaluated the privacy offered by this solution. In particular, supposing that an adversary was able to sniff the traffic between a client and the entry guard of Tor network. We mounted a *web fingerprinting* attack that tries to infer in which cell, the user querying the service is located on.

II. Attribute-based credential

For the implementation of the authorization through *Attribute-based credentials*, we have carefully followed the protocol described by *Pointcheval and Sanders* **PS˙signature**. Nevertheless, during the development of the *ABC* system, some design challenges needed to be solved:

- **Server-Client agreement on the attribute domain**: as described in **PS˙signature**, before starting the protocol, server and client must agree on the public parameters, which includes number of potential attributes L. Moreover, users should choose attributes (subscriptions) which are recognized by the server. To this end, we decided to embed the list of available subscriptions in the server public key. The 'attribute domain' is thus formed by the $L-2$ possible subscriptions, followed by the username, and a client secret key. Clients are expected to choose their subscriptions from the provided set: failing to comply with the protocol on client side will result in the abortion by the server.
- **Attribute encoding**: once we designed the server-client agreement in such a way that each parameter of the public key was mapped to one attribute, we decided to encode attributes in the following way: if the user decides to subscribe to service i, then the exponent of public key parameter i is a fixed prime number in Z_p (where p is the order of the prime groups defined in the paper): `SUBSCRIBED_YES`. Conversely, any service to which the client does not subscribe, is encoded with a different fixed prime: `SUBSCRIBED_NO`. The 'username' attribute is encoded through `int.from_bytes(username.encode(), 'big')`. Client secret key is a random number in Z_p.
- **Fiat-Shamir Heuristic for Issuance Request**: Clients choose their *user-defined* attributes, which in our implementation are: the secret key and username, together with a blinding factor t. Note that the random blinding factor guarantees *issuer unlinkability*. User commitment C will thus be $g^t * Y_L^{client\_sk}$. Client will also produce a *NIZKP* for its commitment. In order to generate the challenge in a non-interactive way, we applied *Fiat-Shamir Heuristic* to the *sigma protocol* defined for the *Pedersen's Commitment PK*. The provided challenge was the *sha256* digest of the public parameters (the commitment C, the randomness R of the proof, and the public key parameters used for exponentiation, g and $Y_L^{client\_sk}$)
- **Fiat-Shamir Heuristic for linking Disclosure Proof to location**: *Fiat Shamir Heuristic* plays also a crucial role when linking the *Showing protocol* of the paper to

a specific message (i.e client's current location). This prevents a malicious user eavesdropping communication to steal one user's credential in order to gain access to the service from its current location, different from the one of the user. In order to apply *Fiat-Shamir Heuristic*, we modelled the *Disclosure Proof* as a regular *NIZKP* on Pedersen's Commitment, where generators belong to the G_T group:

$$PK\{(t, a_i), i \in H : C = e(\sigma'_1, \tilde{g})^t \prod_{i \in H} e(\tilde{Y}_i^{a_i}, \tilde{g})\}$$

Thus, the generators in the Proof of Knowledge are $e(\sigma'_1, \tilde{g})$ and $e(\tilde{Y}_i, \tilde{g})$. In order to link the location to the proof, we used a *Schnorr's Signature*, producing the challenge as:

$$c = sha256(R|C|public\_parameters|location)$$

The server validations follows two steps: first, it checks it can recompute client's commitment via the *bilinear* property of the pairing:

$$C = e(\sigma'_2, \tilde{g}) \prod_{i \in D} e(\sigma'_1, \tilde{Y}_i)^{-a_i} e(\sigma'_1, \tilde{X})^{-1}$$

Secondly, it checks the validity of the proof on client's commitment.

A. *Test*

In order to test the correctness of our *ABC protocol*, we implemented a simple test suite using `pytest` called `test_abc.py`, involving a correct run of the protocol and 4 runs where clients deviate from the protocol in different ways:

- `test_registering_invalid_attributes`: client issues a request with an attribute that does not belong to the agreed domain.
- `test_requesting_service_not_subscribed`: client tries to request a service to which it did not subscribe. Implicitly, this will lead to a *Disclosure Proof* which is not valid for the given credentials.
- `test_invalid_signature_on_message`: it simulates an active adversary who is able to eavesdrop the communication between a client and the server. The adversary will try to ask for the same service requested by the user, from its current location (which is different than that of the client), by replaying the *Disclosure Proof* it captured.
- `test_invalid_user_commitment`: when creating the issue request, client presents a proof which is not valid for his commitment.

B. *Evaluation*

For the evaluation of the *ABC protocol*, we analyzed how the communication cost in terms of exchanged bytes, and the computational cost in terms of runtime, variate in respect to the number of available subscriptions. Each measurement was repeated 20 times. The plots show also the standard error as error bars (where they are not visible, the error was too low):

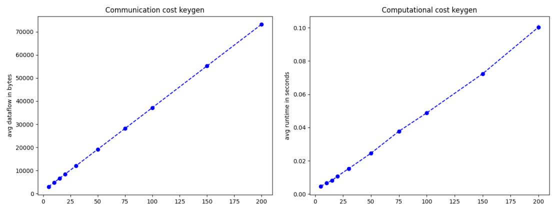

Fig. 1: Avg. dataflow and runtime for the key generation step

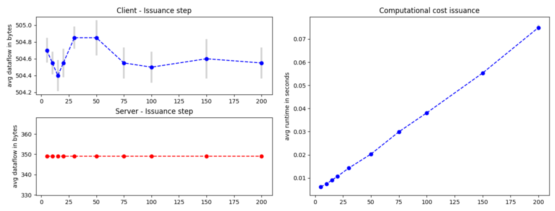

Fig. 2: Avg. dataflow and runtime for the credential issuing

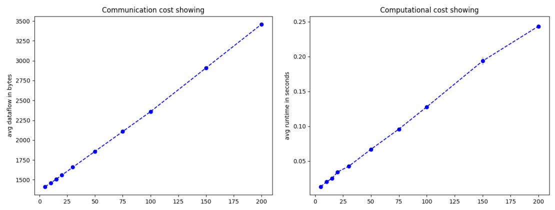

Fig. 3: Avg. dataflow and runtime for client request

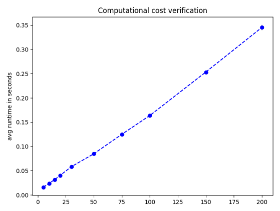

Fig. 4: Avg. runtime for the credential verifications

III. (De)Anonymization of User Trajectories

A. *Privacy Evaluation*

In this section, we performed a privacy evaluation on the network level of SecretStroll.

1) *Identity Inference Attack:*

a) Threat Model: For this attack, we considered SecretStroll service provider as a **passive** adversary (i.e *Honest but Curious*). The adversary is able to read all the queries made from users in SecretStroll grid: the adversarial view is represented by `queries.csv` dataset. We assumed the following background knowledge for the adversary: the adversary knows a mapping between users' POIs and top locations (e.g home, work) and their identity.
We assumed the adversary computational power to be bounded by Polynomial time.

b) Adversarial Goal: The goal of the adversary is to infer users' **top-3 locations**, namely their home location, their workplace location, and any particular place where they spend their leisure time. By knowing such sensible information and combining with her background knowledge, the adversary can link users' IP addresses with their identity.

c) Implementation Details:

- **Feature Extraction**: the first step for performing the attack was a simple process of feature extraction on *queries.csv* dataset. We extracted 4 new features from *timestamp* column: *cell_id* (cell id. of user location in the query), *hour* (hour of the query), *day* (day of the query, from day 1 to 20) and *daytime* (it represents when the query was made during the day. For example, a query launched at 10 a.m was labelled as ”Morning”).
- **Trajectory Analysis**: the second step was to analyze the ”quality” of the provided data: generally, when performing inference on users home or workplace location, the common assumption is that users follow a habitual pattern. Before mounting the attack, we decided to test this assumption. We collected, for each IP address, their daily trajectory (as a time-ordered sequence of latitude and longitude coordinates). We then computed the distance matrix between each pair of daily trajectories using *Frechet Distance*. The results showed that, the trajectories of users taken from day to day tended to be very similar. Furthermore we could assess that *Frechet Distance* represented a meaningful metric, since for each user we could witness a larger distance between trajectories collected during week days and trajectories collected during weekends (as expected by the different mobility patterns of average people during work days compared to their pattern during days off).

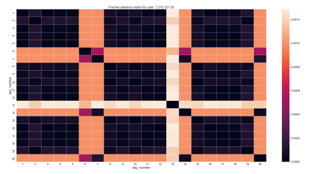

Fig. 5: Distance Matrix of user 7.210.121.52. Each row/column represents the trajectory on that day(from day 1 to 20). The darker the cell *i,j*, the more similar trajectories on day i and j.

- **Location clustering**: After validating the data, we proceeded to mount the attack. We first grouped the daytimes we defined in three groups (”home”, ”work”, ”leisure”) accordingly to where a user was likely to be in that moment. For example, the group *daytime home*, was comprehensive of daytimes ”Early” and ”Night”, since in those moments of the day users are likely to be at their homes. Then, for each user, for each daytime group, we extracted latitudes and longitudes of locations. Then, we fed the coordinates to the **DBSCAN** algorithm. After the clusters were formed, we filtered out the noise and computed the centroid of the biggest cluster. Results are provided in the `top_locations.csv` file. It is worth noticing that, for a few users, there were not enough queries during the ”Evening” to infer something about their ”leisure time” top location.

2) De-anonimization of trajectories trough Co-Location:

a) Reference: The attack was inspired by **8228621**.

b) Threat Model: For this attack, similarly to the previous one, we considered the adversary to be again the **service provider** (again **passive**). As far as her background knowledge is concerned, we assumed that the adversary knows the identity of a subset of users, i.e knows a mapping between users' IP and their identity (we believe that this is a reasonable assumption, either because the adversary run the previous attack identifying a subset of users or she colluded with the ISP gaining such information). Furthermore, we assumed that the adversary has access to a side channel of information about users, more concretely to users' public profiles on social networks. Our additional assumption is that all targeted users have a social network account, and that all co-location information (e.g tagging in a post) corresponds to people being physically together (i.e no fictitious tagging).

c) Adversarial Goal: The goal of the adversary is to derive a pseudo-anonymized social graph of users, where nodes are users' IP addresses and the edges represent a similarity between users' trajectory on a specific day. Furthermore, if the adversary overlays the social graph she derived with the side information she can access from OSN, she can effectively de-anonymize the social graph (or at least part of it), exploiting co-location information. For example, if the adversary learns that user with IP ”A” (Alice, who the adversary knows) and user with IP ”B” had a similar trajectory on day 6, and on the same day Alice posted pictures and posts tagging her friend Bob, the adversary can identify user ”B” as Bob with good probability.

d) Implementation Details: We adopted an approach very similar to step 2 of the previous attack. For each day, we collected each user trajectory. For users whose trajectories were made up of fewer points than the longest trajectory of the day, the last known position was used as padding. We then computed the distance matrix between trajectories, using again *Frechet Distance*. Finally, we used **DSBSCAN** algorithm in order to find clusters within the trajectories. We reported, for each day where a cluster of size greater than one was found, the users whose trajectories belonged to the same cluster, and the day. The results are provided in the `users_trajectories_similarities.csv` file, reporting the two users whose trajectories were similar, and the day of the trajectories.

3) User Interest inference via POI:

a) Thread Model: For this attack, we considered once again SecretStroll service provider as a passive adversary (i.e Honest but Curious).In addition, we also assumed that the adversary has knowledge of users' home and work locations (gained after running attack 1).

b) Adversarial Goal: Here the goal of the adversary is to find the interest of its users based on how many times a specific type of POI was requested by a user over the weekdays and the week-end, based on the time of the day (Early, Morning, Afternoon, Evening and Nights). Such information could be used by the adversary to mount a *Location Inference Attack*,i,e infer users' most likely location in terms of POI locations at a given time of the day by cross-referencing the POI type information with the user top locations (e.g if a users mostly queries "restarant" during the morning, it will probably be at a restaurant near his work place).

day by cross-referencing the POI type information with the user top locations (e.g if a users mostly queries "restaraunt" during the morning, it will probably be at a restaurant near his work place).

c) Implementation Details: In order to do so, entries were split into weekdays and weekends, then grouped by IP addresses, then the number of each POI request by daytime was counted and the most frequent POI per daytime extracted. The information about each POI request count per user can be found in the *users_poi_counts_weekdays.csv* and *users_poi_counts_weekends.csv* files.

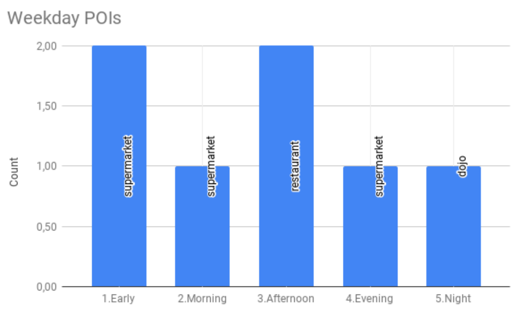

Fig. 6: Most common POIs queried for user 10.229.150.53 during weekdays

The figure above shows that during a week day in the afternoon for example user 10.229.150.53 is most likely at a restaurant.

B. Defences

In order to protect users privacy against the attacks we described in section *A*, we designed the following *Location Privacy Preserving Mechanism*. When accessing the application at client side, users will select a parameter p, to which we refer as *obfuscation probability*. When performing a query to the service, the application will retain the current user location with probability $1-p$. With probability p, the client will choose a random location in the current cell in which the user is located or one of the neighbouring cell (depending if the cell is at one corner of the grid, the number of available neighbours will vary between 2 and 4).

1) Adversarial Models: The LPPM we designed is aimed to protect users against attacks 1 and 2 in section *A*, by introducing noise in the localization mechanism used by the system. Thus, the adversarial models we took into account are the ones we described in the first two aforementioned attacks. Clearly, the mechanism is ineffective against the adversarial model described in attack 3.

2) Assumptions: When evaluating our mechanism, the following assumptions were made:

- **Ground Truth**: Since we were not provided with the ground truth about users in the dataset (i.e the real location of their house and workplace), we assumed the results obtained by the adversary of attack 1 without the mechanism in place as ground truth.
- **Strategic Adversary**: In our evaluation, we considered a strategic adversary, i.e we assumed that the adversary knew the details about the mechanism and that she could consequently mount the best attack possible. Nevertheless, we assumed that the adversary did not know the value of p chosen each time by the client.
  Consequently, the adversary we used for our evaluation was able to dynamically adapt is algorithm. More details about this in the following section about the evaluation.

3) Privacy Gain Evaluation: For the evaluation of the Privacy Gain introduced by our mechanism, we simulated the same scenario of attack 1. The following changes were made:

- **Client**: we modified the queries dataset applying the LPPM. In each test, a different value of p was used.
- **Adversary**: the adversary used the same algorithm described in attack 1, as we believed that the algorithm could perform well for the adversary even in presence of noise (DBSCAN is capable of labelling points as noise). Nevertheless, taking into account that the adversary is strategic, we modified the algorithm to adapt to the amount of noise detected by DBSCAN. If the clustering algorithm returned at least two points labelled as "not noisy", then the algorithm proceeded as described. If no "not noisy" points were found, the adversary used the centroid of the "noise" cluster as prediction (since the noise added by the mechanism is uniform along all directions. There is a probability that by averaging out the noise the adversary can find a good enough prediction).

As Privacy metric, we used a metric applied in the literature **Shokri**, the **Expected Adversarial Error**, i.e the distortion introduced by the mechanism in the outcome of the adversary inference in respect to her inference without the mechanism:

$$Privacy = \frac{\sum_{i=1}^{i=n_{users}} haversine(x, \tilde{x})}{n_{users}}$$

where x is the original prediction and $\tilde{x}$ is the distorted one (i.e the one with the mechanism in place). The evaluation took into account both predictions for home and workplace locations. As distance metric, we used the *haversine* distance between the predicted points (as latitude-longitude pairs), in kilometers.

4) Utility Loss Evaluation: For the evaluation of the Utility Loss which the mechanism eventually adds, we reported the percentage of queries for which the mechanism produced a change in the cell identifier in which the user appeared located, against the parameter p.

5) Utility-Privacy Trade-off: As showed in the previous plots, the mechanism parameter p is directly proportional (linearly) to the Privacy Gain of the users as well to the Utility Loss introduced in the application.

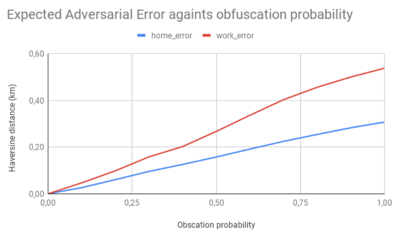

Fig. 7: Privacy gain evaluation

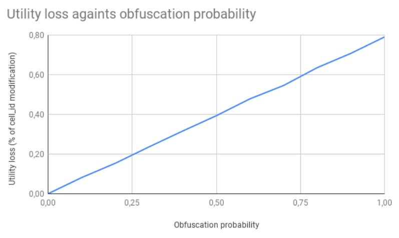

Fig. 8: Utility Loss evaluation

IV. Cell Fingerprinting via Network Traffic Analysis

A. Implementation details

1) Trace collection: The trace collection was performed directly in the client docker container using Tshark. This was done to avoid external noise coming from other sources other than from the communication between the client and the Tor guard nodes. This was automated via a bash script called `generate_traces.sh`. During the capture, we noticed that the communication between the Tor exit Node and the server container was affected by random failures, resulting in incomplete `.pcap` files. In order to fix this issue, we created a bash script called `repair_failed_runs.sh`. The script scans each capture file, for each cell folder, and if the size of a capture file is lower than a certain threshold ,it triggers a new capture for that specific trace and cell. Overall two runs of traces were collected, one with 30 traces per cell, and another one with 148 entries per traces. The traces simulated a user with 5 subscriptions, and every query was made using a combination of the subscriptions, i.e $\binom{5}{k}$ for k in $[1, 5]$.

2) Filtering: Before proceeding with feature extraction, we decided to apply two filters to the captured traces:

- **Less than 609 B**: we analyzed the traffic between client and server, using Tor, when the client did not perform any request. We found out that packets were sent. Nevertheless, these packets were all below the size of $609B$. In particular, we have seen that at TLS level the frame size was $543B$, that appears to be standard for TLS application data sent inside single TOR cells. For this reasons, we filtered out packets below this threshold when evesdropping client requests.
- **Removed TCP ACKs**: Following an approach found in literature **web˙fingerprinting**, we decided to remove TCP ACKs sent by the server since they did not carry any payload.

3) Feature Extraction: In order to select a suitable set of features for the classification task, we tried to follow the features presented in the lecture notes, as well as in literature **web˙fingerprinting**:

- 1 - Total number of packets.
- 2 - Number of incoming packets from server.
- 3 - Ratio between number of incoming packets and total number of packets.
- 4 - Ratio between number of packets sent by client and total.
- 5 - Total number of bytes exchanged in KB.
- 6 - Number of incoming bytes in KB.
- 7 - Ratio between incoming bytes and total bytes.
- 8 - Ratio between bytes sent by client and total bytes.
- 9 - Average size of incoming packets in KB.
- 10 - Markers: how many time the direction of flow changes in the capture.
- 11 - 13 Top 3 Bursts size in KB.
- 14 - 16 Top 3 Bursts timestamp in seconds.
- 17 - 19 Top 3 Bursts sequence number in capture.
- 20 - 21 Percentage fluctuation in packet sizes, i.e ratio between first packet size and median packet size, and between median and last packet sizes.
- 22 Average time interval between incoming packets in nanoseconds.

B. Evaluation

In order to perform the multi-label classification, we trained a *Random Forest Classifier* provided by the `sklearn` library. In order to evaluate the performance of the classifier, we used the Exact Match Ratio metric. As validation method, we used a 10-fold stratified cross validation, and repeated the evaluation 5 times, each time using different randomness for the splits.

C. Discussion and Countermeasures

The overall result of the model was very good, scoring an accuracy of 0.926 with a standard deviation of 0.006. The attack achieved good results even with less traces, scoring around 0.82 accuracy with 0.02 standard deviation with just 30 samples per cell. This very effective attack represents one of the limitation of Tor system. Nevertheless, it is possible to adopt some countermeasures in order to lower the success rate (i.e accuracy, in this case), of the attack. Since the classifier we built mainly relies on packets size and timing, *SecretStroll* server could try to uniform data from all the cells, for example adding decoy POIs, in order to pad the response for every grid queried by the user such that they all have the same length. Furthermore, it could add some random delay between sending packets (at cost of latency), so to hide the peculiar timing for each cell response, or even make sure to send, for each cell, all packets with fixed time intervals.