CS-523 SecretStroll Report

Lucas Rollet, Lucie Hoffmann

Abstract—**In this report, we present our privacy-friendly Location Service Provider (LSP) based on the Attribute-based Credential (ABC) scheme seen in class and discuss potential user de-anonymization and localization.**

I. Introduction

We designed a system where a user can subscribe to some Point Of Interest (POI) types on a server and query any POI type contained in their subscriptions without the server being able to link the queries between them as well as to the user. We first implemented this system based on ABC, then we analyzed privacy breaches in the face of an honest but curious adversary: the LSP, finally we evaluated how well an eavesdropper in the Tor network could infer the source localization of a query packet, without having direct access to the request content.

II. Attribute-based credential

The user gets their credentials from the issuer LSP when subscribing for a set of POI types. Their secret (hidden attribute) is their username and the attributes they disclose correspond to their subscriptions. The user commits to their username and obtains a proof from the server that they can access the service for these specific subscriptions they paid for: a blind signature on these subscriptions. This corresponds to the issuance step.

The user can then query the service by signing their request with their credentials, which they randomize each time (as described in the showing protocol) to avoid issuer and verifier linkability.

For the user to prove their commitment of username and thus validate its registration, we implemented a zero knowledge proof (ZKP) based on Pedersen's commitment using a Fiat-Shamir heuristic to make it non-interactive. Once this commitment is verified and an issue request is signed, the server can prepare the registration of the user by verifying that the commitment is valid. Once this step is done, the user can use the server's blind signature on its issuance request to obtain the final credential that will be used for later requests.

When the user proves their authorized access to the service for a specific set of subscriptions (disclosed attributes), we use the equality in slide 44 of the ABC lecture to verify the signature corresponds to the user attributes and we also multiply the corresponding user commitment with a hash of public values (queried types, commitment, message) to match the request signature with the specific query on specific types (should belong to the subscriptions set) at the specific location. This way, the disclosed set of subscriptions as well as the message cannot be tampered with once a request has been signed. This also constitute a non-interactive ZKP since the server verify the commitment based on the disclosed attributes and the public values they know.

We implemented the first ZKP as follows:

To create a commitment, we generate as much random r_i as there are secrets (= hidden attributes). Then we elevate public generators (those matching the hidden attributes in the server's public key) at the power of these random generated values. The product of all of these constitute an integer R, for which we create a challenge. The challenge is made out of R, the list of public generators, as well as the user commitment (and the optional message). As shown in the course and exercises, we then create a list of exponent $s_i = (r_i - \text{challenge} * \text{secret}_i) \bmod p$, where p is the order of the group we do the commitment in, for i corresponding to all secrets indices (in our case there is only one secret, i.e. the username).

To verify such a commitment, we are given the challenge computed before, the list of s_i as well as the public generators. We verify that the challenge re-computed from the given parameters corresponds to the sent challenge.

Note that in our implementation, the hidden attributes (here the username) always appear before the disclosed attributes (i.e. subscriptions) in the lists of attributes.

A. Test

We test 3 successful paths: the correct generation of keys by the server, the whole stroll process (credentials generation and client signed request) and the case where 2 credentials for the same attributes (hidden and disclosed) should not be linked. For the keys generation, we check that the number of elements in the secret and public keys are consistent with the number of attributes (=subscriptions + username) and we check these elements' types (group elements and big numbers). For the stroll process, we verify the length and types of all elements in the process stays consistent across the different stroll operations. We also check that the signature is valid when a client makes a signed request to the server with their credentials.

We test 5 failure cases: we verify that credentials are not valid and system fails when the user or the server modifies the username or subscriptions, or when the queried types don't match the subscriptions, or when the signed message is tampered. For the case where the server modifies some attribute, i.e. a user attribute is tampered between the blind signature of the server on these attributes and the derivation of credentials from this signature, then the system fails the client does not get any credentials. For the case where the user modifies some attribute after getting the credentials for their attributes, the server does not accept the user credentials associated to the wrong set of attributes.

B. Evaluation

We compare the execution time and number of packets exchanged between the 5 following steps: the key generation, the credential issuance (user registration and server's response signature), request signature (credential disclosure), server request verification (disclosure verification), and the full process (including all 3 previous steps).

The slowest step in our design is the credential disclosure from the user to the server when querying for some POI types. This is also the step with the greatest size of packets exchanged. Indeed, while the credential issuance only consists of the user sending their registration on one side and the server sending back their blind signature on the other side, in the credential disclosure step, the amount of packets sent by the server is proportional to the amount of matching POIs in the user grid cell. This is the bottleneck of the full process.

The number of packets and the execution time is clearly higher than what we would get with a simple non-privacy-friendly request, with username, POI types and location only encoded as plain-text strings. This is due to the exchanges of heavy credentials in the issuance step and in the showing protocol every time a request is made. A higher number of packets to exchange also leads to higher execution times. Nevertheless, the issuance and showing process for one request stays rather fast. The computation overhead of signing and verifying would be more important with more request, especially for the server who serves more than one user. Still, for a lot of applications the induced overhead of using Attribute Based Credentials would be too much to handle and induce a too high utility cost. For processes with lots of small requests, signing every request with a credential makes requests bigger and slower which is definitely a parameter to take into account when considering larger and more complex ecosystems.

The Figures 1 and 2 show our performance evaluation results, as well as the standard mean error for each evaluation.

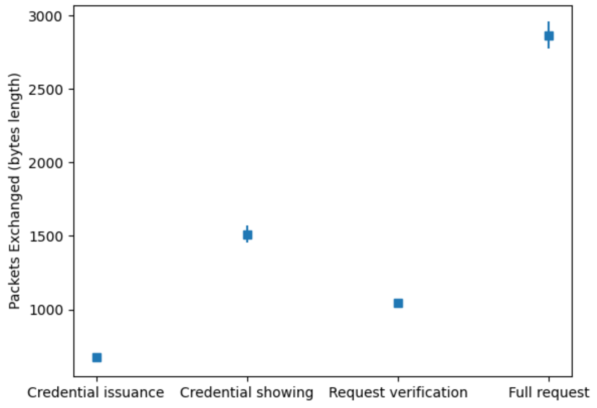


Fig. 1: ABCs execution time cost evaluation

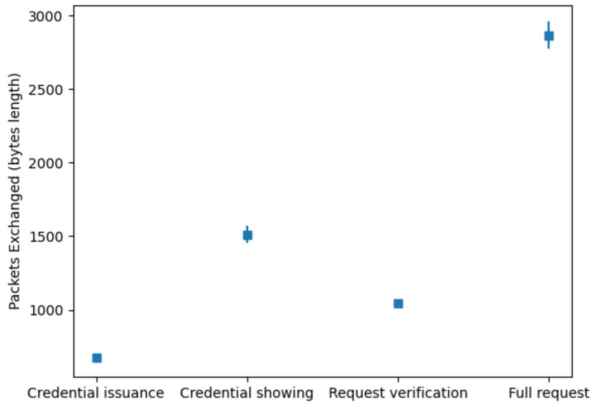


Fig. 2: ABCs communication cost evaluation

III. (De)Anonymization of User Trajectories

A. Privacy Evaluation

The Location Service Provider (LSP) users' pseudonymized (with IP address) queries, containing their location and some of their POI types subscriptions. The LSP, in this threat model, is an honest but curious adversary: they can use the information provided in queries to infer information about their users, such as their lifestyle, eating habits, working place, home, etc. For this, they just need to link a user to their corresponding pseudonym IP address. To do so, an LSP employee can, for instance, be around the user at the moment of the query, scan the network and infer their IP. Another way to deanonymize the IP address-queries is to use prior knowledge (like the home and working place, or some of the user's hobbies/habits) on the user: the more prior knowledge, the more likely they can uniquely identify a user. Especially if this user has uncommon characteristics with respect to the whole queries data set.

In any case, it should not be hard to link users to their IP address. In the following, we see how to infer the living and working place for a given IP. Then we infer the habits and interests associated to this IP. Finally we show that we can infer other users (associated IPs) they spend time with: we look into the more specific cases of who they likely spend lunchtime with and who they likely share their home with (we could do this for any time of the day).

Suppose the LSP knows Bob IP address is 197.134.251.204. Then, based on our results, the LSP can additionally learn that Bob leaves somewhere around the location pointed on Figure 3 and works somewhere around the location pointed by 4.

We can also, for instance, observe Bob's habits during the week (Figure 5) versus during the weekend (Figure 6). A Monday timetable of Bob is shown on Figure 7. We can deduce that, on Monday, Bob was at his company looking for a place to eat at 12:48 am, then at 5:10 pm he was in his apartment looking for a club and a gym, and finally after going to the gym he got hungry at about 10pm.

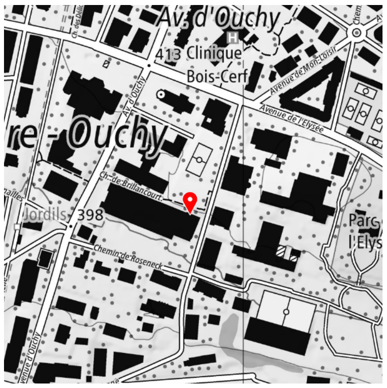


Fig. 3: Bob's home inferred from the location at which he was querying during the night.

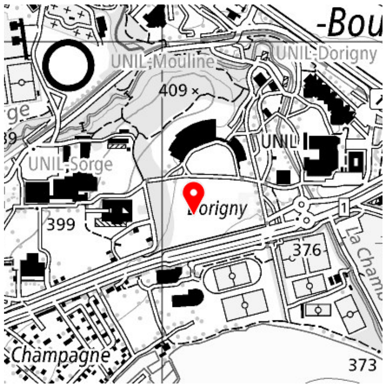


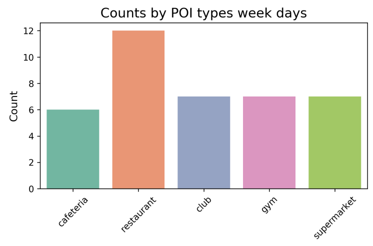
Fig. 4: Bob's work place inferred from the location at which he was querying during weekday working times.

A last element we can get more knowledge on is Bob's social network. By looking at the other users being close to Bob's location at about the same moment during lunch time, we find no match: Bob possibly usually eats alone (or people with him don't use the app in this moment). Similarly, looking at people around Bob during night times, we can deduce users he might be close to (his partner, family or friends). In this case we found he spent the night close to several IP addresses (148.172.128.122, 202.39.106.20, 13.191.142.105, 27.89.29.117, 3.36.170.195, 124.223.29.175, 91.60.233.179) not all at the same moment. By also de-anonymizing these, we can grow our knowledge on Bob's network.

B. Defences

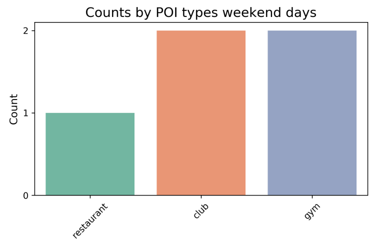
A solution for Bob to protect his privacy would be to add dummy queries with different POI types he is subscribed to. More specifically, for each real query, Bob can add queries for all POI types he is subscribed to and don't correspond to the one he queried for. This should prevent the LSP to get more knowledge on Bob's interests and habits.

In our case, we define privacy by with probability to learn additional information (w.r.t. already known subscriptions) on Bob's interests and habits given a query at a precise time. The higher this probability, the higher the privacy breach.



POI type	Count
cafeteria	6
restaurant	12
club	7
gym	7
supermarket	7

Fig. 5: Bob's queried POI types during week days.



POI type	Count
restaurant	1
club	2
gym	2

Fig. 6: Bob's queried POI types during weekends.

We assume the honest but curious adversary, the LSP, has no prior knowledge on the user's habits and interests.

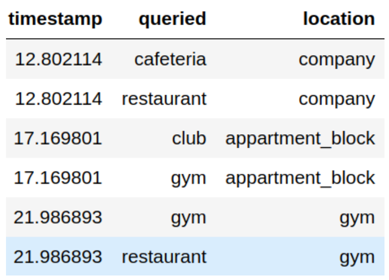
Without our defense, given a Bob's query at a precise time, there are no other queries at this precise time, so we can deduce what Bob was looking for at this precise time and get additional knowledge on him with probability 1.

With our defense, this probability goes down to $\frac{1}{\#\text{subscriptions}}$ (i.e. $\frac{1}{5}$ for Bob, we assume his subscriptions correspond to all query types associated to him in the queries), since every query type the user is subscribed to appears equally for every query and precise time. With additional dummies, Figures 5 and 6 transform into 8 and 10 respectively.

The LSP would still have access to the user's location, being able to deduce their home, work place and other whereabouts, but they would not be aware of their intentions while querying. Unfortunately, depending on their location and time of querying, they may be more a less likely to query certain types of POIs. A user might be less likely to look for a cafeteria than a club in the middle of the night for example. This can increase the defined probability. Also note that the smaller the number of subscriptions, the higher the probability. This means that the user should pay for more subscriptions to ensure more privacy.

As for utility, we define it as the ratio between the amount of useful information the user gets and the total amount of information received for one real query. For each query they make, the user would receive POIs corresponding to all their subscriptions. This ratio is then $\frac{\#(\text{POIs for 1 type})}{\#(\text{POIs for (\#subscriptions) types})}$. The closer this ratio is to 1, the better the utility, so the lower the number of subscriptions, the better the utility. Indeed, the more POIs unrelated to his query type, the less easier the user will see what he is truly interested in.

In this defense case, we see how utility and privacy are opposite of each other and there is a trade-off between increasing



timestamp	queried	location
12.802114	cafeteria	company
12.802114	restaurant	company
17.169801	club	appartment_block
17.169801	gym	appartment_block
21.986893	gym	gym
21.986893	restaurant	gym

Fig. 7: Bob's first Monday timetable based on queries and actual location.

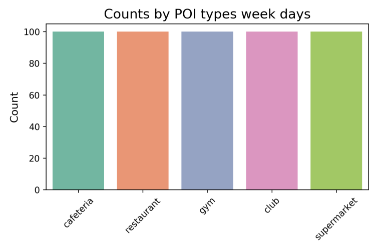


Fig. 8: Bob's queried POI types with additional dummies during week days.

the number of subscriptions to consider in the dummy queries to get more privacy and decreasing it to get more utility.

IV. Cell Fingerprinting via Network Traffic Analysis

A. Implementation details

To perform our data collection we made a custom script (fingerprint_collect.sh) using tcpdump to track packets exchanged between the client and the server on port 9050. We configured the server and the client once in the beginning, generated the key pairs, fetched them on the client, and registered the client. After that initial setup, we looped on every grid point from 1 to 100, generating a new .pcap file containing all the packets exchanged for each new grid point request. We then repeated this step 10 times, so we could have 10 different requests through Tor for every grid point. That would allow us to train our classifier with a complete dataset. It is important to note that for every grid request, the client asks for a random subset of the point of interests it is subscribed to. This allows the request signature to not be biased based on an identifying set of disclosed attributes. One run of the entire fingerprinting process takes approximately 6 hours to complete, and generates 1000 .pcap files.

After analyzing our output files and their structure, we decided to extract the following features: the total number of packets exchanged, the total time to process the request (time of the last packet - time of the first packet), the number of HTTP OK (200) received, as well as the length of all the HTTP payloads. Each HTTP packet payload's length is a feature by itself. To extract those features we used the library Scapy, which allowed us to parse our pcap files and analyze

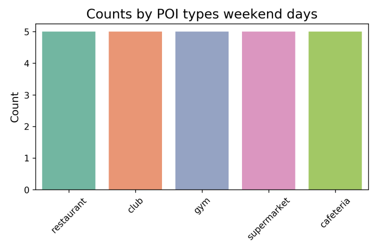


Fig. 9: Bob's queried POI types with additional dummies during weekends.

the packets. Since each class (grid point) must have the same number of features, we padded the features array with zeros when they were too short. For instance, when a trace has less HTTP packets than the others, it has to be padded. We chose these features because we noticed that each grid point seemed to have a different amount of packets exchanged, packets sizes, and request time. On our machines, a request for grid point 5 took around 10 seconds, with 400 packets exchanged in total (12 HTTP). On the other hand a request for grid point 36 took around 20 seconds, with 613 packets exchanged (16 HTTP).

Once the features were extracted, we ran the random forest classifier with 10 folds, feeding it our extracted features and labels, as arrays of numbers. For each grid point, we fed 10 different items to the classifier. We did not need to tweak the parameters explicitly.

B. Evaluation

To evaluate the performance of our classifier, we computed its mean accuracy, as well as its per-class mean accuracy. With all the features quoted above, our classifier achieved a 89% accuracy. It means that the classifier was able to correctly guess the grid point queried from a test request 89% of the time. To perform this evaluation we compared the classifier's predictions with the actual test labels for each run of the classifier, and averaged them in the end of the training.

We also computed the per-class accuracy, to see if the classifier was able to guess correctly for every grid points, or if some points were left behind. The result can be seen in the following figure. It appears that most grid points are guessed correctly, only a few have a lesser accuracy, always between 60% to 100%.

C. Discussion and Countermeasures

As seen in the previous subsection, our classifier performs quite well. This is probably due to the fact that cells are easily distinguishable from each other in this case (each have a differing number of POIs). The more POIs in the cell, the more the number of packets and so the longer the processing time, as well as the bigger the payload (more POIs to communicate). The selected features for our classifier are thus interdependent and complete each other in the identification of the grid cell.

A possible countermeasure would be to pad the HTTP payloads for all packets to have the same length, independently of the grid cell. This padding should be made on the initial

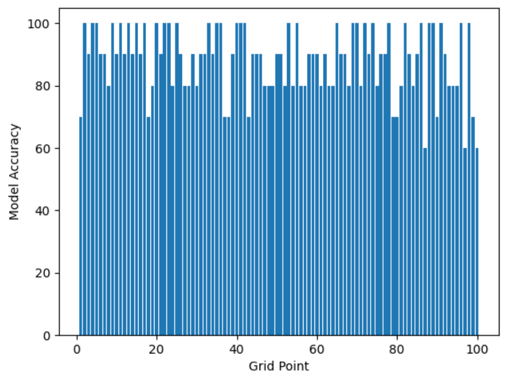


Fig. 10: Classifier's per-class accuracy.

client's request, with the required subscriptions, as well as the server's responses containing the actual POIs. But this countermeasure by itself is not sufficient. Some grid points could take more time to fetch server-side than others. In this situation, the system would still be vulnerable to timing-attacks. One of the biggest problems of the Tor Network is that it is particularly vulnerable to these kind of attacks, to preserve efficiency. To preserve privacy, it would be beneficial to make all the requests execute in the same time, which could be the maximum time reported for any POI request execution. A last countermeasure that would have to be implemented is to make every request exchange the exact same amount of packets. Some grid points contain more data than others and it results with different number of packets in requests. That is also a leak of information that could lead to easier fingerprinting.

Those countermeasures obviously have a big impact on utility. The system's performance would be greatly impacted since every request would be the bigger and longer possible, preventing any kind of optimization. If any grid point is slow to fetch, every grid point will be impacted. In a general setting, this compromise is not acceptable since modern web servers are designed to handle plenty of requests simultaneously for a lot of different clients. Furthermore, the bandwidth waste would make such a service less scalable, and less usable for bad internet connections. As nearly always with privacy enhancing technologies, a compromise has to be made between utility and privacy.