CS-523 SecretStroll Report

Wicky Simon, Nunes Silva Daniel Filipe

***Abstract*—This project consists of three parts, all related to a central topic : Privacy in a location based service. The first part makes use of Attributes based credentials (ABC) and zero-knowledge proof to anonymously query a server for nearby Points of Interests (PoIs), while proving the ownership of a subscription. The second part we evaluate the loss of privacy induced by IP-level metadata. The last part of this project is dedicated to trying to infer location of query author, based solely on metadata.**

I. Introduction

The aim of the first part of this project is to build a privacy-friendly location based service, SecretStroll, which allows users to query a database about nearby PoIs while remaining anonymous. Since this service is not free, every query must provide the proof of an active subscription. Each PoIs is part of a category such as *Restaurant*, *Cinema*, *Train station*, etc. Each of these category are part of a subscription and a user can have many of them, which naturally leads to ABC. A user joining the service first register himself with the category he paid for and then sign each of his query anonymously, revealing only the categories he want to get PoIs from.

II. Attribute-based credential

The base of the ABC scheme for this work is described in Section 6 of [1]. This scheme has been used as follows :

- The messages mentionned in [1] are attributes, representing categories of PoIs. The server creates a public key large enough to encode all of these attributes.
- When a user wants to register, she commits to attributes she paid for, that will remain hidden. She uses a zero-knowledge proof (ZKP) to prove that she did so correctly. The server checks the proof, and sign this commitment that she will then use as credential.
- When a user makes a query, she choose the attributes (i.e the categories of PoIs) that she want. She then makes use of her credential to prove she has a subscription for the queried attributes, using a ZKP. She also uses this credential to sign a message representing her current location using the Fiat-Shamir heuristic described in [2] and detailed later.

In order to make the ZKP non-interactive, we use the Fiat-Shamir heuristic. The challenge that the prover would send is replaced by a cryptographic hash of all known values. During the registration, the challenge is composed of the committed attributes, the commitment used for the ZKP and the server's public. To sign the message for a query, it is simply added to the other values constituting the challenge.

This overall approach leaks only the number of attributes chosen, but it is necessery for the server to correctly compute the ZKP. Otherwise, hidden attributes cannot be inferred and every credential is unique.

	Rounds	Mean	Std. deviation
Key generation	1000	27.30 ms	4.30 ms
Credential issuance	1000	41.86 ms	8.24 ms
Message signing	1000	44.34 ms	10.32 ms
Signature verification	1000	44.31 ms	9.52 ms

TABLE I: Result of the offline benchmark,

A. Test

The system is tested with six different tests :

1) *test_generate_ca()* : The server's certificate generation is tested with a simple structure test, to see if all keys have the right lenghth.
2) *test_credentials_difference()* : Two requests are created with the same input parameters, a registration is perforemd with both of them and two credentials are created. To preserve anonymity, the requests and the built credential should be different.
   Two credentials created from the same request should also be different.
3) *test_tampered_credential()* : A request with invalid credential should be rejected. In this test, a valid credential is tampered to be invalid.
4) *test_correct_credential()* : A request with a correct credential should be accepted.
5) *test_correct_credential_no_attributes()* : A request revealing no attributes should be valid.
6) *test_wrong_revealed_attr()* : A request revealing unobtained attributes should be invalid.

The effectiveness of these tests could be assessed using a standard metric such as branch coverage or coverage. However, these metrics are not complete and will not discover every bug present in the code.

B. Evaluation

The system is benchmarked in two parts, an offline part, which evaluate pure computationnal time and an online part which evaluate the communication time.

1) Offline benchmark: The offline benchmark tests four features :

- Key generation
- Credential issuance
- Message signing
- Signature verification

The result are shown in figure 1 and reported in table I

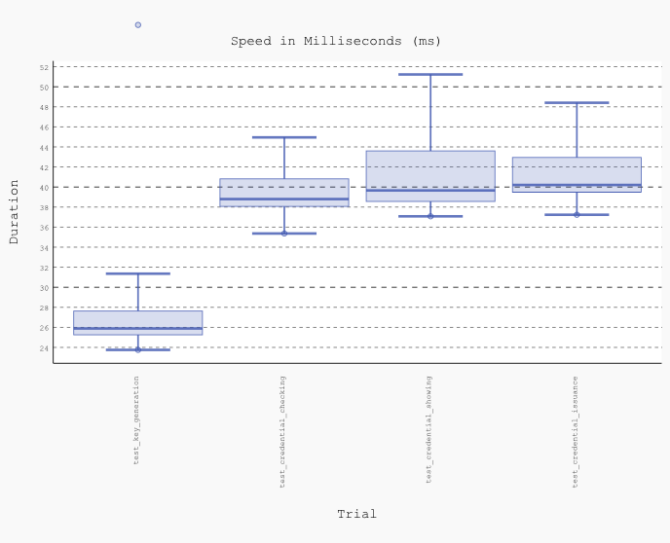

Fig. 1: Histogram of the offline benchmark

	Rounds	Mean	Std. deviation
Registration	1000	53.58 ms	1.93 ms
PoIs request	1000	90.07 ms	1.90 ms

TABLE II: Result of the online benchmark

2) Online benchmark: The online benchmark tests two features :

- Registration
- PoIs request

Note : This benchmark has requirements. Follow the instructions at the beginning of *benchmark_net.py*.

The result are shown in figure 2 and reported in table II

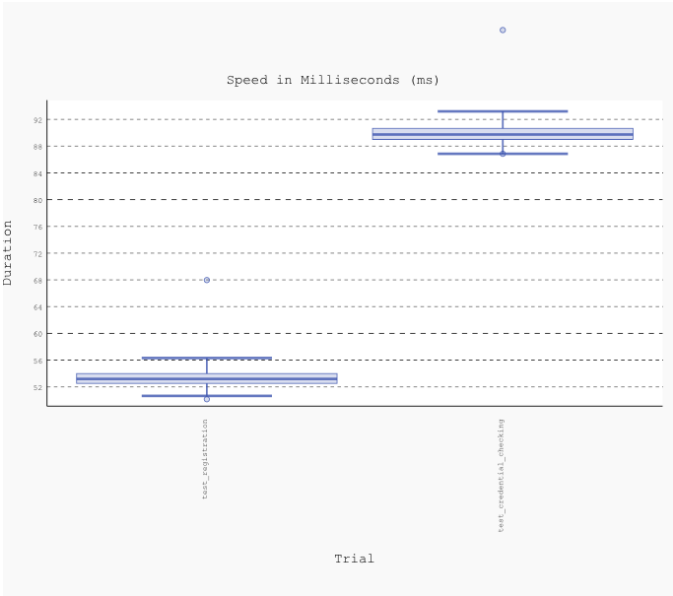

Fig. 2: Histogram of the online benchmark

III. (De)Anonymization of User Trajectories

A. Privacy Evaluation

We evaluate the privacy risks using simulated data of two hundred users who made use of the application hundred times each in average over twenty days. We assume that no mechanism to hide any kind of data is used, i.e. data is sent in cleartext inside standard IP packets. Any malicious adversary could hack the application servers or sniff the network between a user and the server in order to retrieve similar datasets which include IP addresses, locations, query types, timestamps and responses. According to [3], the IP address or a set or IP addresses are relevant attributes because they can be linked to a given user since they are persistent for a certain duration. Moreover, combining IP addresses with location and time data makes sense since users often keep their habits that they may share with very few people [4], i.e. they may work during the day at their office, be back home in the evening and do an activity at some specific location. Therefore, we expect this implementation to leak senstive informations about the users. As a proof of concept for breaching users privacy, we try to infer work and home locations of users as well as habits in their activities.

We build our attacks in a constructive way, starting by learning general tendencies about the users, then, we focus on one specific user in order to learn how far we can go with breaching breach its privacy.

As stated before, home and work are sensitive locations because they allow to identify users. In order to learn about them, we go through all users queries and check if users queries locations correspond to home types locations such as *appartment blocks* or *villas* as well as work types locations such as *offices*, *laboratories* or *companies*. It turns out that each user has one location of each from where it does the majority of its queries which allows us to draw a map linking IP addresses to their corresponding home and work locations. Moreover, 3 confirms our concern about users anonymity once their home and work locations would leak in the given setup. The logarithmic scale clearifies the orders of magnitude between people who share or not these POIs showing that the majority of the users have a tuple home and work location that they share with very few people as presented in [4].

Then, we analyze when users make use of the app from their home and work locations. On the one hand, we observe that they use it regularly on a daily basis from home. On the other hand, we can see that they query the service from their work location only during weekdays, which is compatible with not being on their work location on the weekend. Moreover, we consider when the app is used during the day on weekdays in 4. This shows that users likely spend their day at work while being at home at the end of the day. This short analysis gives a great understanding of users habits even though the presented results seem trivial.

Finally, we focus on breaching privacy of a chosen user to assess if we can learn more about a given user than general tendencies. Our results are condensed in the graph presented

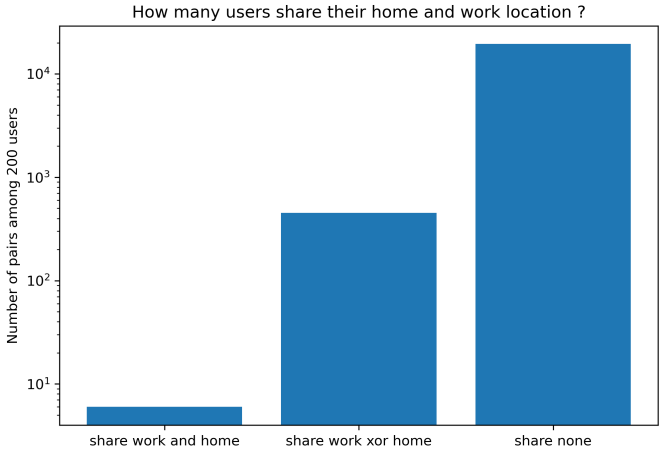

Fig. 3: Comparison of how people share their main POIs

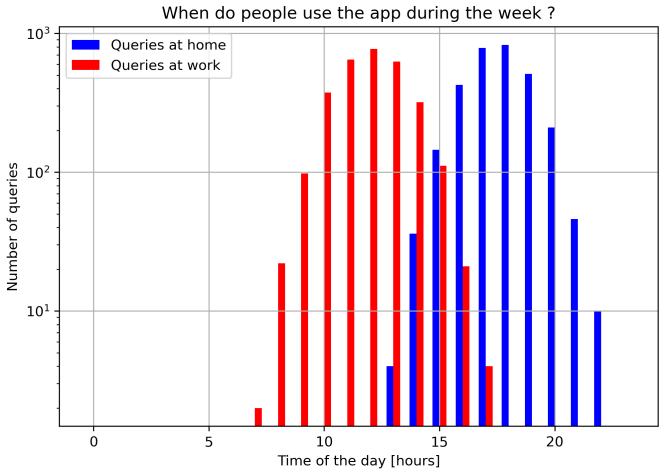

Fig. 4: User habits

in 5 that shows how user moves as a function of time. We are able to determine where the user lives and works making use of our previous findings. From that, we can see how this user's life is centered around its home and work locations. In fact, it goes back and forth its home and work during the week while its trajectory take completely different paths on the weekends. Applying similar techniques for other POIs types allows us to learn about this user's favorite restaurant and when it used to eat there. Then, grouping *bars* and *clubs* as entertaining activities opposed to *gym* and *dojo* which can be considered quiet ones, we deduce that this particular user is more into the former ones which are located close to its home. He basically goes to the gym on the weekends as we can see these spiky outliers in 5.

To sum up, we assessed how we could breach users privacy by analyzing either general habits about the users or more personal ones by focusing on specific users. This leads us to conclusive results regarding our initial references and expectations leveraging simple algorithms such as matching users queries locations with meaningful POIs. Cost would depend

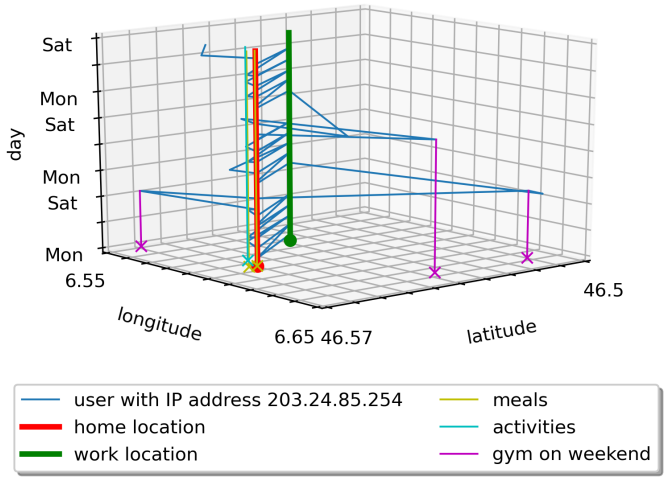

Fig. 5: Monitoring of a randomly chosen user using collecting data

how close and severely someone wants to attack a specific user. As shown, either we can keep it very general to learn about tendencies or focus on a given user. For future attack, we may be interested to evaluate how the current POIs locations may be related to the next queries in order to understand better users behaviour. For example, how likely is it that a user wants to go for a beer after work compared to after gym ?

B. *Defences*

Now that assessed that we assessed the privacy issues of this app, we suggest one simple and lightweight client-side defence that would help improving it.

We remember that the purpose of this app is to suggest POIs of a given type in the grid-based vicinity of the user. According to the specifications and [5], an actual grid cell is of a similar scale of a city whose side measures a few kilometers. However, the collected location infomartions has artificially ridiculous precision considering that eight decimal places is precise up to a millimeter. For this, we can suggest generalization by reducing the precision of the queries locations to two decimal degrees. In fact, it is still sufficiently precise to determine in which grid cell the service should do its recommendation in this setup. A cleaner way of implementing this would be by allowing users to download the grid cells locations and dimensions so that when querying the server they would transmit the minimum of information required to keep its utility, i.e. a cell ID.

This defense aims to protect the privacy related to the exact location of each query. In our previous attack, we take advantage of being able to map queries locations to the exact corresponding POIs. In this case, we are restricted to the POIs in the actual grid cell whose sizes are crucial and difficult to optimize. For example, if there are only bars in a given grid cell then if a user who would query the service from

there would be likely doing it from a bar. Cells should then have k-anonymity, l-diversity and t-closeness to optimize this aspect so that if a user is querying the service from a grid cell, we should not be able to say from which POI nor which kind of POI the user is doing it from. However, this only provides only a probabilistic fine-grained location protection, assuming grid cells have the correct properties, that still may be defeated. A strategic adversary could learn about general habits of users but it would be harder to infer as much detail as we did in our attack, i.e. exact home and work locations as well as wenkend activities or favorite restaurant. For example, it could still observe home and work rides based on grids cells and timestamp but should not be able to say which exact POI inside the given cell it is own.

Our defence machnism does not affect utility from the results perspective. User will be able to receive the same suggestions as without this mechanism. In fact, we limit the user's outgoing information to what is strictly necessary to achieve the same utility in this setup. However, having to process this information locally to deduce the cell ID would require the user to have these grid properties stored on its phone. When travelling to a new region, the user may not have the coordinates available to derive the right cell ID which could limit the utility by requiring additional bandwith.

Making use of this mechanism allowed us to turn the previous anonymity set of peole sharing at least of home or work location from 457 pairs of users to 1186 pairs of user. This poor improvement can be justified mostly by the simplicity of the mechanism as well as by the non uniform distribution of POIs in the specified grids. The used mechanism of strongly utility oriented since it does not improve the privacy reasonably but preserves a similar utility with negligeable overhead. For this mechanism to be more efficient, strong assumptions about how grid are drawn, i.e. increase grid cells in order to include more POIs in each and provide better anonymity. This would also imply a utility decrease as the notion of vicinity my become too broad for the user to find POIs nearby. An alternative that would be interesting to assess in a future work would be replacing actual location informations by dummy ones. POIs are already established, we can make use of their semantic to compensate the usual lack of plausibility often experienced by this mechanism, i.e. a user could replace its home location by an other one located nearby in order to prevent the leak of it.

IV. Cell Fingerprinting via Network Traffic Analysis

A. Implementation details

1) Data collection: The data was collected using the `capture.sh` script. It loops through all the cells, queries the server for it, and dump the traffic between the client and the proxy. The traffic is captured there to avoid parasitic traffic. Each cell was queried 10 times, for a total of 1000 queries.

The file `pcap.py` contains various function, to check the correctness of the capture file, aggregate features and save them in a csv to store them and prepare the data for our classifier. Note that since we used these functions for data collections only, they are not designed to be robust and might crash if the format requirements are not met.

The data fed to the classifier consists of the bytes of the packets header converted to an integer between 0 and 255. Each of these values is normalized to be in the $[0, 1]$ range. To allows the classifier to extract some structure, we kept only the headers who were 66 bytes long, which was the most frequent. We also kept only the first 400 packets, to have a reasonable training time. All capture smaller than 400 packets are simply padded with 0 bytes. These numbers are totally arbitrary and could be tuned if the available hardware was more powerful.

2) Classification: Inspired by [6] and [7], we implemented our classifier as a standard convolutionnal neural network. Its architecture is shown in figure 6 and is constituted of the following layers:

1) Convolution
   - Input channels: 1
   - Output channels: 32
   - Kernel size: 66
   - Stride: 66

   The parameters of this layer allows the classifier to extract per packet features.
2) Max-Pool:
   - Kernel size: 2
3) ReLu
4) Convolution
   - Input channels: 32
   - Output channels: 64
   - Kernel size: 3
5) ReLu
6) Fully connected
   - Input size: 12672
   - Output size: 500
7) ReLu
8) Fully connected
   - Input size: 200
   - Output size: 500

The training is done with 50 epochs, the cross-entropy is used as loss function and the *Adam optimizer* as optimizer. As we used 10-fold cross-validation, the training dataset is constituted of 900 samples, the test dataset contains 100 samples.

B. Evaluation

After a 10 fold cross validation, the results are represented in table III.

The final average is 73.3% error rate, with a standard deviation of 9.18%.

C. Discussion and Countermeasures

At first sight, it seems that our classifier is quite bad. However, considering the number of possible classes, if we were guessing at random, we would have an error rate of

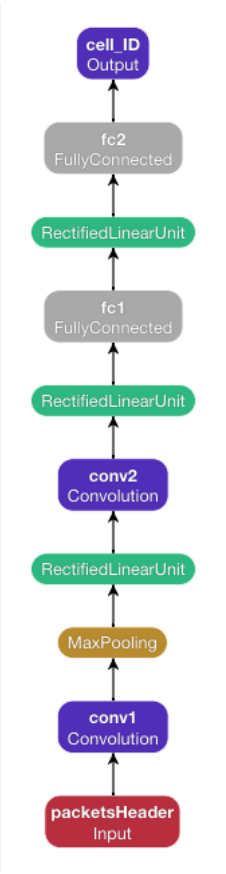

Fig. 6: Architecture of the CNN classifier [1]

[1] The software is available on Github under a MIT license (https://github.com/mlajtos/moniel).

Iteration	Error rate
1	66%
2	81%
3	68%
4	54%
5	74%
6	68%
7	86%
8	73%
9	83%
10	80%

TABLE III: Test error rate for each iteration

99%. This classifier allows us to improve our confidence approximately 26 times! Even by using Tor, our anonymity is not that protected. It is also important to note that this classifier was trained using a relatively small dataset, and low computationnal power. With more time and more powerful processing power, this result can surely be improved a lot.

Finding countermeasures to such a tool is extremely difficult. Machine Learning tries to extract structure in the data, and even when there is no structure at all, as seen in class, we can train an algorithm to detect this lack of structure.

References

[1] D. Pointcheval and O. Sanders, "Short randomizable signatures," Cryptology ePrint Archive, Report 2015/525, 2015, https://eprint.iacr.org/2015/525.

[2] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," vol. 263, 03 1999.

[3] V. Mishra, P. Laperdrix, A. Vastel, W. Rudametkin, R. Rouvoy, and M. Lopatka, "Don't count me out: On the relevance of ip addresses in the tracking ecosystem," *HAL*, 2020.

[4] P. Golle and K. Partridge, *On the Anonymity of Home/Work Location Pairs*, 2009.

[5] W. contributors, "Decimal degrees — Wikipedia, the free encyclopedia," 2020, online; accessed 08-May-2020. [Online]. Available: https://en.wikipedia.org/wiki/Decimal_degrees

[6] M. Kim and A. Anpalagan, "Tor traffic classification from raw packet header using convolutional neural network," 07 2018, pp. 187–190.

[7] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 76–81, 2019.