CS523 Project 2 Report

Pavliv Valentyna, Lion Clement

***Abstract*—In this project, we want to create a privacy-preserving app that returns Points of Interest (POI) near the user's location. Since users need to have a subscription to access our application, we want to implement an authentication method based using zero-knowledge proofs. In order to do so, we use the Fiat-Shamir heuristic. We then evaluate the risks caused by the fact of sending the users IP address, in particular, how much information can be disclosed about the user from just his queries. Finally, we evaluate the threats one privacy due to the meta-datas that are still sent. We try to recover the position of the user based on a traffic analysis of the network. In order to do this, we use machine learning.**

I. Introduction

Most recent location-based application opt for a free-to-use business model. They let the user use their services without any form of paid subscription. However, these types of application will frequently ask to login or create an account. The application can then build a marketing profile of the user by linking the different locations he goes to. One can recover the job, the house, the centres of interest... These information are valuable and we don't know what the application does with it.

In this project, we will code a location-based application that, based on the user's location, returns nearby points of interest. We want to avoid infringements on privacy like described earlier. In order to do so, the user should never have to login to the application, but he will have to prove that he has a subscription by using a zero-knowledge proof.

We start by designing an anonymous authentication mechanism using attribute-based credentials. We inspire ourselves from the signature scheme described in this paper[1], we consider that the messages to sign correspond to the POIs the user wants to see. However, we want to make our protocol non interactive and therefore, we use the Fiat-Shamir heuristic to sign our messages.

In a second part, we evaluate the privacy risks on the user by trying to uncover information about them by looking at the queries they send. Once we have seen the threat to privacy caused by this type of attack, we try to propose a defense mechanism.

Finally, we conduct a fingerprinting attack on the application to see what information an adversary can recover by just eavesdropping on the communications between the client and the sever. In order to do so, we train a machine learning algorithm to recognize the client's queries.

II. Attribute-based credential

A. Details

In the algorithm, a Server object has access to an issuer object which can generate new parameters (secret and public), but also proceed requests from the Clients. Clients can communicate with an AnonCredential object which takes the user data to create valid signatures using the Fiat-Shamir heuristic. A sample run of the algorithm is described below :

Algorithm 1 Sample run of the algorithm

1: The Server asks an issuer to generate a new public key and a secret key.
2: A Client wants to register. He creates an object AnonCredential who will create a registration request.
3: The AnonCredential creates a zero-knowledge proof based on the Fiat-Shamir heurisitc. The message to sign are the user attributes.
4: The Server receives the registration request. He asks an issuer to verify the correctness of the request and to send back credentials.
5: The Client asks the AnonCredential to proceed the received information.
6: The Client asks the AnonCredential to create a Signature object which can then be sent to the Server.
7: The Signature object is created using the Fiat-Shamir heuristic.
8: The Client can now have access to the application without having to reveal his credentials, but by presenting his Signature object.

We can see that only the objects AnonCredential and Issuer know information about the Client. However neither of them know the full information, and both of them are discarded after use. Therefore, the Server is unable to find the identity of the clients and to use the information to his advantage. We use two zero-knowledge proofs at two points in our system. The first one being in our registration request and the second one in the verification of our signature. In our registration request, we want to prove that he knows the message m he wants to get credential for, and a secret value t which he will use to create a valid signature.

We use the Fiat-Shamir heuristic a second time, to prove that we know a message and that we have valid credentials for this message.

All these measures make sure that a client can prove that he has access to the application, without revealing his identity or personal information.

B. Test

To test our system, we did a series of individual verifications (checking our serialization function, checking the public key and private key and checking the attributes). We also check that for a pairing function e, a public key $(\tilde{g}, \tilde{X}, \tilde{Y}_i) = (\tilde{g}, \tilde{g}^x, \tilde{g}^{y_i})$. With a credential $\sigma = (\sigma_1, \sigma_2)$ and $\sigma\prime = ({\sigma_1}^r, ({\sigma_2} * {\sigma_1}^t)^r)$. We check that :

$$e(\sigma_1\prime, \tilde{X} * \prod \tilde{Y_i}^{m_i}) = e(\sigma_2\prime, \tilde{g})$$

This property is vital for us to implement our zero-knowledge signatures. Finally, we check that a valid signature given to an honest user is accepted by the server.

All these tests serve us to prove that an honest client can get access to the application and will not encounter some strange error.

We also check that dishonest users fail to connect. In order to check this, we test three failure cases :

- If a dishonest user intercepts a registration request from an honest user and wants to use it for himself with another name.
- If a dishonest user wants to use the fake credentials with $\sigma\prime$ = (1, 1).
- If a dishonest user wants to forge fake credentials by taking random elements as $\sigma\prime$.

From these tests, we considered that the application filed its role, as it would accept honest users with zero-knowledge and reject malicious users.

C. Evaluation

We notice that most functions have a very low execution time (to generate new private and public keys with the method generate_ca, we only need on average 128 milliseconds with a variance of 5 * 10^{-9}). We notice that the communication time (time between the moment the client sends a request and the server answers), is bigger with an average of 390 milliseconds and a variance of 8 * 10^{-9}. This result was predictable, since all our functions are just doing mathematical operations, which has a low complexity. We can expect the communication time to grow if we have a real network infrastructure and not just two dockers communicating. We find that the whole issuance of credential process (steps 2 to 5 in the Algorithm 1) takes 10 milliseconds with variance of 2 * 10^{-7}. The signing of a message takes 8 milliseconds with a variance of 3 * 10^{-9}. The verification of a signature takes 8 milliseconds and has a variance of 3 * 10^{-8}.

In order to get these numbers, we would take the time for 20 iterations, then keep adding new points until the mean would stop changing from one iteration to another.

III. (De)Anonymization of User Trajectories

A. Privacy Evaluation

We assume that the application lets you query for POIs in your immediate neighbourhood. The position sent to the server is therefore the position of the client. Our application is privacy-preserving, we thus assumed that the adversary in our case a maleficent eavesdropper who can observe queries. Another possible adversary could be the system that deals with queries, i.e. the server. We got some test data of the queries some potential users could make: the server receives each query with the following information: the IP address (one per user), location of the user, time when the query is made and a POI. It is clear that in this case the IP address is equivalent to a pseudonym, so we can sort queries per user, which is very easy and fast.

We now assume that every user is at work from Monday to Friday at 9-12AM and 2-5PM and users are at home during most of the time outside of this period. We could refine the model by saying that users sleep at home between 10PM and 6AM, or by suppressing queries made from the workplace during free time (we noticed that there is a lot of queries from exactly the same place many days along, so we could use that to define more precisely the home/work location, but we supposed that 20 days are not sufficient to use it). Based on this, we can recover home and work location of each user. We can see on Fig. 1 the locations off all queries of 10 random users (one color per user, the same color on each figure), their work location on Fig. 2 and their home location of Fig. 3.

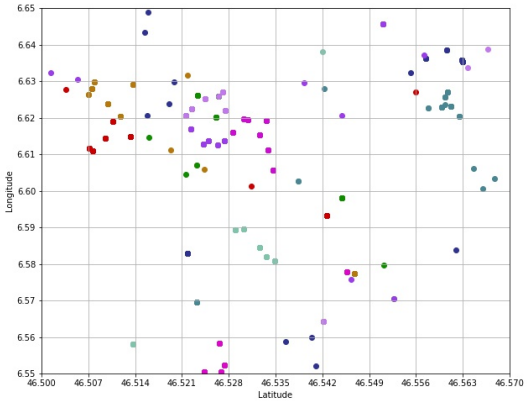

Fig. 1: Total locations of 10 users

When we look at all the queries, they seem very confused. If we can assign to each person a ”zone”, we don’t know what this zone represents. Our data point need to be cleaned in order for us to recover precise information.

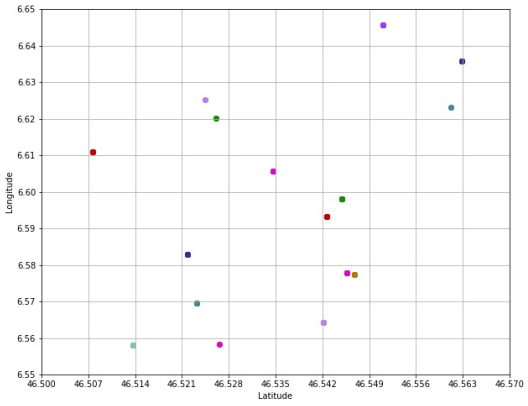

Fig. 2: Work locations of 10 users

We can see that, though the data are a bit noisy, we can get an idea of the workplace of each individual. From that, we can guess what his job is (with eventually a bit of extra information like ”At what time precisely does he go to work ?”, ”When does he leave his workplace ?” or ”Does he live in a rich neighbourhood ?”)

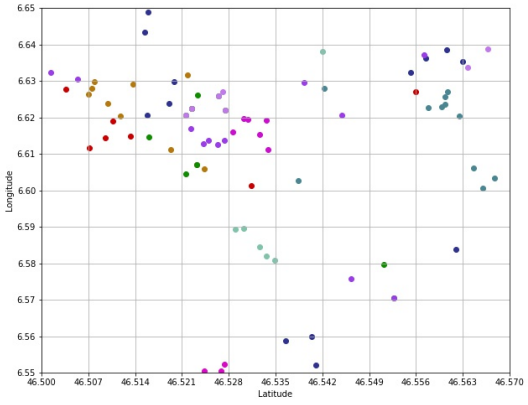

Fig. 3: Home locations of 10 users

It is clear that there are more queries made during the free time, and thus location of ”home” becomes more noisy, because users during free time can go outside, i.e. don’t only stay home.

If we weaken our assumption and consider that a client can query any location from anywhere, we can still use the attack described above. We can suppose that most clients search for POIs around places where they spend a lot of time (i.e. their workplace, their house or places where they like to go). For example a worker wants to know if there are any cafeterias around for him to have lunch. Moreover, the second assumption can explain the noise in our data. Indeed, sometimes a client will look at the POIs around his vacation place or do plans for his weekend during the work time or when he is at home.

The home and work location are not the only information we can recover from this data: we can also recover all the POIs the users signed for. This can be done by joining all the queries by IP addresses. An example is given in Figure 4. We can also do some data processing on the number of POIs and get more precision on the preferences of the user, or sort them by work time/free time, as it was done for the locations. (”gym” and ”dojo” are mainly looked for during free time. This is understandable, a client looks for these kinds of location only when he can do sports, and therefore not while working).

ip address	POIs
7.154.93.97	{'bar', 'club', 'cafeteria', 'gym', 'restaurant'}
49.205.49.101	{'dojo', 'bar', 'cafeteria', 'gym', 'restaurant'}
166.3.233.225	{'dojo', 'bar', 'cafeteria', 'gym', 'restaurant'}
114.193.164.29	{'dojo', 'restaurant', 'bar', 'cafeteria'}
222.215.127.43	{'supermarket', 'bar', 'cafeteria', 'gym', 'restaurant'}
250.163.7.81	{'supermarket', 'club', 'cafeteria', 'gym', 'restaurant'}
138.53.90.242	{'dojo', 'supermarket', 'bar', 'cafeteria', 'restaurant'}
21.97.153.208	{'cafeteria', 'restaurant', 'supermarket', 'club'}
227.101.137.102	{'supermarket', 'bar', 'club', 'cafeteria', 'restaurant'}
52.1.34.89	{'supermarket', 'bar', 'cafeteria', 'gym', 'restaurant'}

Fig. 4: POIs of 10 users

These weaknesses are very problematic because we want to hide a user’s subscriptions, locations, and other private information that can be inferred (their home or work location for example). In the current state, they can be found far too easily.

B. Defences

We have some propositions of solutions to counter these attacks.

First, if the queries that the server receives contain no IP address, an adversary can’t find much information. At most, he would find the most popular workplaces for the customers of the app. To hide the IP addresses, we can install a Tor-like system. To be fully privacy preserving, this system needs a lot of people using the app, otherwise we could classify them by zones even without the IP address (If the users are spaced out, if we suppose they don’t move too quickly, two queries made in the same 5km radius in a 30 minutes time frame, are likely to belong to the same user). We suppose that it is the case and that our app is popular (because it is awesome). We have seen in the course that there is no delay when communicating between Tor nodes, so we believe that our app will still have good utility. Tor resists to State-wise enemies, so we think it would be suitable as a defence. But let’s say we cannot install Tor-like system. What are other possibilities?

One of the easiest implementations to provide some defenses is to delete the time and map each exact location to the corresponding cell ID. This would prevent the work / free time hobby spotting, also it would blur the location. But it is

of course insufficient to hide completely the places where the user go, as well as his subscribed POIs: so we need to find some amelioration to this.

Another thing we propose is to add noise to each query: knowing there is only 12 possible POIs, the user could send for instance 4 queries with only one real request and 3 requests for random POIs. It is not important that the user is subscribed to the POI in the fake request, because the answers will be discarded by the user anyway. Moreover, if the POIs of the fake requests do not match the user's subscription, we can hide the subscription to an adversary.To hide the location, we could the same way send fake requests with random locations and one request with the real one. Of course, the more we send fake requests to hide the real POI subscription, the better it is: the adversary would need a lot more data to determine the real POIs. However, we can do statistical analysis to recover the POI which are the more likely to be sent by the user and the location of his house/workplace (we average the location of the queries during work time and during the afternoon. We have seen on the figures that users usually send queries from the same place). Furthermore, let's suppose we send 11 fake requests for one real request, the server would be saturated, which may cause it to break. This solution appears to offer neither privacy nor utility.

If now we assume that a client can query any location from anywhere, we can use the same defences to protect our client's privacy. Indeed, the defences we discuss don't require anything particular from the client.

IV. Cell Fingerprinting via Network Traffic Analysis

A. Implementation details

1) Data Processing

For the third part of the project, we plan to use machine learning to identify the queries made by the user. In order to do so, we use Wireshark to capture traffic for all the different types of queries a user can make. For that, we asked the client to do one thousand queries to the server (ten for each zones) and between each queries, he would "ping" the local host. These "pings" would help us separate each client query and create a total of a thousand csv files.

Now that we have our samples, we need to build a feature vector from these samples. We extract the number of messages sent, the total time it took to deal with a client query (we have seen in part 1 that communication time between the server and a client was high, so time can tell us a lot about our communication). We also classify each message depending on the sender and the receiver and use this as a feature (if there are more POIs in a zone, the server should send more messages). We look at the type of communications that were used (TCP or TLS) and use it as a feature.

Once we extract a vector from our data, we give it to our algorithm to train from it. Our algorithm uses it to learn patterns on each client query and how to make apart one sample from the other.

2) Algorithm Architecture

When it came to creating our algorithm, we wanted to create something simple that could be trained quickly.

To create our machine learning algorithm, we used Tensorflow and in particular the Keras API. In our model, we use a total of 20 dense layers with a relu activation function. We use this activation function since it is easy to compute and we want our algorithm to be quick. We also decide to use a big number of epochs - 2000 epochs. We decided to not use batch size, because we have a very little data size (only 1000 samples). We use 10 - fold validation in order to test the effectiveness of our algorithm.

In order to train our model, we created some functions that extract statistical data from one network traffic observation. We trained our model with all features we found, to have more chances to find which one leak information about the cell ID. In the end, a feature vector contains 14 features: there is the total traffic length and the duration of the communication (these ones could be used in our case because we have laboratory created data - there is almost no noise here; but in the real world data we think that it couldn't be used-at least not without cleaning and subtracting the average additional noise). We found out the IP address of the participants which communicate with TLS/TCP. The communications mostly take place between three IP addresses. So, the other features concerns these packets in particular: which part of the communication is TCP, what is average window length of these TCP communications etc. After processing all data files, we obtain 1000 line of training data, each one with length 14.

B. Evaluation

Our system performed badly with a precision of 11.7%. While this result is better than trying to guess randomly the zone the user queried, it is disappointing. It is sure that if we significantly increase the number of epochs and the number of layers in our model, we would get better results, but we don't think that it would a good result anyway (we tried a model with 10 layers and 20 epochs which give us an accuracy of 5% for a 2 minutes computing. On the other hand, we tried the same model with 20 layers and 2000 epochs and got our 11.7% accuracy for 1 hour of computing). On the other hand, these values could be used anyway: there is 1 chance of 10 to be correct, which is still significant on really big datasets.

C. Discussion and Countermeasures

1) Comments

It is very complicated to find meaningful features. When we looked at a communication intercepted by wireshark, most of

it is hardly understandable by humans. As a result, we didn't know what our algorithm should focus on, and what it could use as feature. In order to do a better classifier, we would need more knowledge about TCP and TLS communications.

Having more training data would have certainly helped us getting better results (only ten samples per category is a low number. Ideally, we would have needed ten times as much). Trying a more complex model (with more layers for example) could have also improved our results. We unfortunately didn't have time to experiment more with these aspects.

All our tests were in laboratory conditions. If we have to face a real time situation, where the server may be crowded, or the network connection is bad, some features like "total time taken to make a query" become complicated to interpret. Furthermore, some information may be different just from running devices that are more or less powerful. In order to do a better classifier, we would have needed training samples from different kinds of devices, in different types of situations.

2) Defences

In order to avoid these types of attacks, we need to look at each features individually and see how we can blur them. For all the queries to have the same number of packets, we could send empty packets just to have a higher number of messages. The server could wait before delivering an answer, so that each query takes the same time to get a response... The problem with this approach is that we don't know all the features that can be used, and that this method is not general enough.

Indeed, we just need our packets to be less recognizable. In order to do so, when a client makes a query for a zone, he adds in nine queries for some zones chosen at random. The server would send him all of his answers, the client's device choosing locally what he really wants. The problem with this approach is that though it might need more training, a well-trained algorithm could still find some information from the user's ten queries. Moreover, sending ten times more packets also means more work from the server.

V. Conclusion and learning outcome

This project was the occasion for us to put in practice the notions of our course. We discovered that if ideas like zero-knowledge proofs are easy enough to understand, they can be complicated to put in place.

This project also raised awareness on how metadatas can be used to retrieve valuable information, and how complicated it can be to create a zero-knowledge application.

This project was also the opportunity to make a link between the topic of Machine learning and our Security and Privacy courses. We had to think on how to collect our data, how to use them and finally which machine learning solution was the more appropriate. We realized however that we lacked the insight in machine learning and packets analyzing to answer correctly to part 3 of the problem.

References

[1] D. Pointcheval and O. Sanders, "Short Randomizable Signatures," vol. 9610, pp. 111–126, 2016, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-29485-8_7