CS-523 SecretStroll Report

Alexandre Santangelo, Andrej Milicevic

***Abstract*—This paper aims to provide a detailed description of a Location Based System called SecretStroll. The paper contains our decision regarding the implementation of privacy-preserving mechanisms and the privacy evaluation of the system.**

I. Introduction

SecretStroll is a Location Based System which users can query in order to get the location of the nearest point of interest (POI) to them. The system is designed in multiple layers, each of which can leak private information about its users, such as their top locations (home and workplace) and their daily habits. For each of these layers, a privacy-preserving mechanism was designed, to prevent as much privacy leakage as possible. We also conducted tests, and a privacy evaluation of each part.

II. Attribute-based credential

In order to offer a user authorisation system that relies solely on user subscription possession to allow or refuse requests, we chose to give each possible subscription a unique identifier. The user can possess multiple subscriptions but is only required to disclose the one he needs for a specific request.

Users do need to have at least one subscription, otherwise they are not considered as such.

The attributes consist of a non disclosed identifier which uniquely identifies the user. This attribute is never disclosed for obvious privacy reasons, but was added as suggested. It could be used to uniquely identify malicious users, when requested to show their "secret" (not the secret key).

A. Types

1) **AttributeMap**
   For the attribute representation, we use Big numbers (Bn) from the strongly recommended petrelic library. These allow us to play with them in their respective groups and use the library specialised operations.
   For the AttributeMap, we decided to use a simple dictionary mapping integers to attributes. The first value of the map corresponds to the user uid, the rest correspond to attributes, i.e, subscriptions. If valid, these are big numbers that map to an existing subscription string or to the string "None". The subscription to big number mapping consists of encoding the string representation to bytes, hashing those bytes (SHA-256) and finally using the petrelic function *Bn.from_binary*. This minimizes the collision risk between multiple subscription strings.
2) **Fiat-Shamir**
   In this process, we first define our attributes and credentials in accordance with the Pointcheval-Sanders (PS) scheme. We then proceed to refine our proofs of knowledge. It is important to note that the foundation of our secrets in both proofs is tied to equations defined by the multiplication of certain group elements (notated as Y_i or $e(\sigma', \tilde{Y}_i)$) that are acknowledged by both parties, and then exponentiated by our secrets (designated as t or a_i).
   Consequently, we adapted our proofs of knowledge in accordance with this analogy, as delineated in the solutions to Exercise Set 1.2. We further extended this by employing the Fiat-Schamir heuristic to achieve non-interactive proofs. This eliminates the necessity for the server to transmit a challenge, instead computing it as a hash of the public values, the commit value (notated as 'com' in the exercise and the code), and the prover commitment (notated as 'R' in the exercise and code).
   The prover thus excludes the transmission of 'R' and the responses s_i, instead sending only the generated challenge 'c' and the responses s_i. This modification on the verifier's end necessitates the computation of the commit value 'R' with the given information (as per the interactive scenario) and the utilization of this value to calculate the challenge c'. The verification then involves the assertion that $c = c'$, a computation that should be computationally infeasible for a prover without a valid credential since 'c' is bound to 'R' and 'com', both of which depend on the said credential. We emphasize, in this report and in the code, that we refer to the proof as a "proof of knowledge" instead of a "zero-knowledge proof", as the Fiat-Schamir heuristic breaks the zero-knowledge property by making the proof's trace non-simulatable (a non-holder of the secret cannot feign possession by producing valid 'c' and s_i).
   There are subtle distinctions between both knowledge proofs that we should highlight:
   In the proof of holding user attributes to blindly sign, the challenge 'c' is computed as $c = H(g, Y_1, ..., \tilde{Y_L}, com, R)$. This is similar to the implementation detailed in Exercise Set 1.2.
   For the second proof, we utilized the commit value's inherent properties to circumvent its transmission, thereby reducing communication cost without compromising the cryptographic security of our proof. As shown in the ABC guide (protocol part 2.b), the left-hand side of the equation is computable using only the disclosed attributes, public values, and the presented credential, allowing the server to calculate it without direct receipt. The right-hand side is computable using public values, the presented credential, and hidden attributes, making it calculable by the prover. This process enables the

completion of the equation (both sides are independently computed, and for the proof to be successful, the equation must hold true) and prevents the redundancy of sending a 'com' value. We observed that binding the credential showing knowledge proof to the actual location query could be beneficial. This reduces the hash collision probability by increasing the entropy of the hashed material and also necessitates the adversary to have access to the location query, demanding a stronger adversary. Thus, the final challenge is calculated as $c = H(g, Y_1, ..., \tilde{Y_L}, com, R, message)$, where 'message' contains the location query text.

B. Test

We built two sets of tests, the first one aims at testing the functions in credentials.py and the second tests the functions in stroll.py. We test most of our functions for success and failure scenarios.

1) *Test Key Generation*: This test validates the generation of the secret key (sK) and the public key (pK) based on given attributes. It verifies the elements of the generated keys, confirming that they meet the expected criteria.
2) *Test Empty Key Generation*: This test checks if the key generation function can handle an empty list of attributes. It should raise a ValueError in this case.
3) *Test Negative Key Generation*: This test checks if the key generation function can handle a list of negative attributes. It should raise a TypeError in this case.
4) *Test Sign*: This test validates the sign function. It generates a signature based on given secret key and message, and verifies the signature using the corresponding public key.
5) *Test Sign Fail*: This test checks the scenarios where the sign function should fail. It should raise TypeError or ValueError when given incorrect messages, or return false when verifying with an incorrect public key.
6) *Test Issue Request*: This test validates the creation of an issue request based on a given public key and a set of user attributes.
7) *Test Sign Issue Request*: This test validates the signing of an issue request based on a given secret key, public key, issue request, and issuer attributes.
8) *Test Verify Issue Request*: This test validates the verification of an issue request based on a given public key.
9) *Test Obtain Credential*: This test validates the process of obtaining a credential. It goes through the complete flow of creating an issue request, signing the request, and then obtaining the credential.
10) *Test Disclosure Proof Verification*: This test validates the creation and verification of a disclosure proof. It goes through the complete flow of obtaining a credential, creating a disclosure proof, and then verifying the proof.
11) *Test Certificate Generation*: This test is about the generation of a server-side cryptographic key pair: the secret key (sk) and the public key (pk). It uses the list of subscriptions as a part of the key generation process. The key pair generation process is verified by making sure that keys are of the correct types. Then, it checks that the signing and verification process works with all possible subscriptions and with some subscriptions.
12) *Test Request signing*: This test verifies the client's ability to prepare a registration request and the server's ability to process it. It checks the workflow for a user named "Mike" who wants to subscribe to "cinemas", "museums", and "theaters". The client prepares the registration request, the server processes it, and then the client processes the response from the server. The client then tries to sign a request with the obtained credentials.
13) *Test checking signature*: This test verifies the server's ability to check the request signature created by a client. It first checks the process for a user named "Jhon" who wants to subscribe to "ballet" and "opera". Then it checks the server's ability to verify the client's signature under different scenarios: with only one subscription, all subscriptions, and no subscriptions. Lastly, it verifies that the server properly handles invalid scenarios: no subscriptions and invalid subscriptions, in both cases expecting the server to raise a ValueError.

C. Evaluation

We evaluated our implementation of ABCs in terms of computation cost, presented by the runtime in milliseconds. These values were obtained by running the credentials tests on cs523_x86_64 virtual machine on a poor 2020 MacbookPro.

stat	step	CS523 VM
Mean	Key Generation	27.6606
	Issuance	23.5081
	Signing	15.6503
	Verification	10.2663
Standard Deviation	Key Generation	10.1341
	Issuance	9.7123
	Signing	9.8231
	Verification	10.1645

TABLE I: Performance using Benchmark (ms)

III. (De)Anonymization of User Trajectories

This section presents our privacy evaluation of the SecretStroll service, which includes two possible attacks which can be mounted, and one defence which could be implemented to increase the privacy of the users.

A. Privacy Evaluation

1) Identity Inference Attack
   - *Threat Model*: For this attack we assumed an Honest but Curious (i.e. passive) adversary, the SecretStroll service provider. The provider has access to the queries made by all the users, as well as information about POIs registered in the system. We also assume the provider has the necessary knowledge to deanonymize a user completely once they know where the user lives and works.

- *Goal of the adversary*: Based on the queries received, find out the main locations of each user: the location of their home and their workplace. Once the adversary has that information, they can find the identity of the user.
- *Implementation*: At first, we analysed the data provided in the `csv` files and did some feature extraction. Based on the data, we were able to extract some additional information, such as the exact time of day and hour of each query, and with that whether or not it was a weekday or the weekend. Continuing our analysis, we found that all the queries issued in the `queries.csv` file came from some POI already known to the service provider. We then checked all POI types which appear in the dataset, and separate the home (e.g. `apartment_block`, etc.) and workplace (e.g. `office`, etc.) types. Thus, we chose to filter the data based on where the query came from, i.e. from which type of POI was it issued. This gave us a view of the data from which we were able to pinpoint the exact `poi_id` of each user's home location and workplace location. With their background knowledge and this mapping between the user's IP address, home, and workplace ids, the service provider can now identify our users uniquely.

2) Inferring user interests and habits

- *Threat Model*: Similarly to the previous attack, we assume the adversary is an Honest but Curious service provider. We also assume the provider knows each user's home and workplace location, which they could learn by doing the previous attack.
- *Goal of the adversary*: Find out the interests and habits of the SecretStroll service users.
- *Implementation*: The way we implemented this attack was by looking at the queries the users made from location which were different to their home and workplace. This resulted in finding out where each user was and how many times they went there, in the 20-day span of collecting data. For example, in table II, we can see data about a few users. In particular, the user with IP 113.244.164.228 used the SecretStroll system numerous times from a club or a dojo, whereas the user with IP 101.193.212.180 evidently eats out very often, at a cafeteria or a restaurant. Now, in order to gain some insight into daily routines of the users, we also looked into our user's requests at a specific time of day during the week. For this we used all queries issued, even those from home and work. By doing this, we were able to learn what is the most frequent POI searched for by each user, for each time of day. As seen in figure 1, a specific user with IP 113.244.164.228 is most likely to look for a restaurant in the morning and a club in the evening.

IP Address	bar	cafeteria	club	dojo	gym	restaurant	s.market
101.193.212.180	0	19	3	0	1	14	0
113.244.164.228	0	6	15	9	0	0	7
13.103.179.102	6	0	0	11	10	0	12

TABLE II: Some of the user's locations during data collection

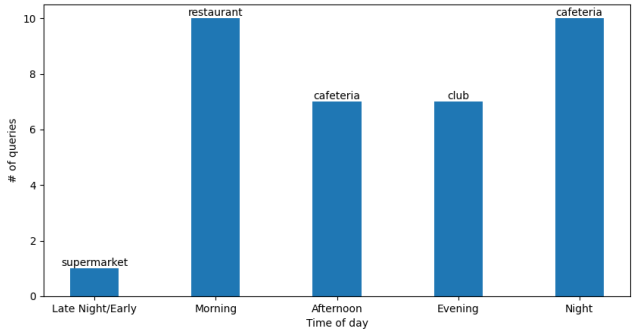

Fig. 1: Most frequent queries for user with IP address 113.244.164.228, during a weekday

B. Defences

In order to defend against the first attack described in this section, we decided to implement a client-side defence based on location obfuscation. Following are the details of this implementation.

1) *Adversary model and assumptions*: The adversary model here is the one described in the first attack in this section, since this defence concerns that particular attack. We used the results from the first attack as true information about the users. We also assumed a strategic adversary, one who knows the details of this defence mechanism.
2) *Implementation details*: The user can choose a value which represents the probability p of the system obfuscating their location when using the service. The values of p are from the interval $[0, 1]$, with increments of 0.1. The user would thus appear to be located at a different location than they are with probability p, and their true location would be revealed with probability $1-p$. Since, in the dataset, we have 100 different cell ids, we assumed locations are divided in a 10x10 grid. The obfuscation considered the neighbourhood of the current cell the user was in, namely the 8 closest cells surrounding it (if the true location is not in some corner, then there are fewer candidates). Instead of the true location, the system chose a random location within the current cell, or in any of the other cells applicable. To simulate this, we made modifications to the original dataset and conducted the evaluation from there.
3) *Privacy evaluation*: We tested our defence mechanism with each possible value of p. The adversary in this case was looking for the most frequently seen POI of the correct type (home type/workplace type) to try to infer the true home and workplace locations of each user. We then compared the true values against the obfuscated

ones. The way we compared the datasets was by using haversine distance to calculate the distance between two locations given by their latitude and longitude. We took the average distance (in kilometers) of distances between true and obfuscated home locations, and the same for workplace locations. The results we got can be seen in Figure 2 and Figure 3. As can be seen, the obfuscation is not very effective until users choose a high value for p, when the privacy gain rapidly increases, and it is the same effect for both locations, only with larger distance between them.

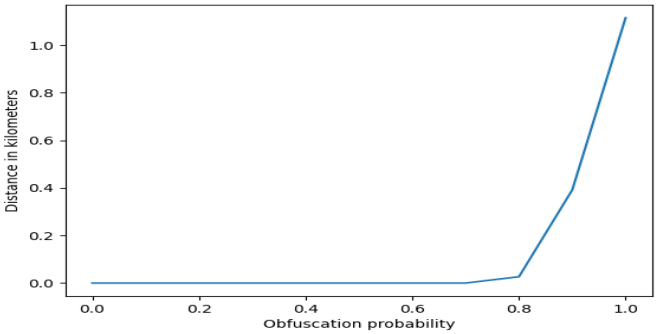

Fig. 2: Effect of obfuscation on inferring home location

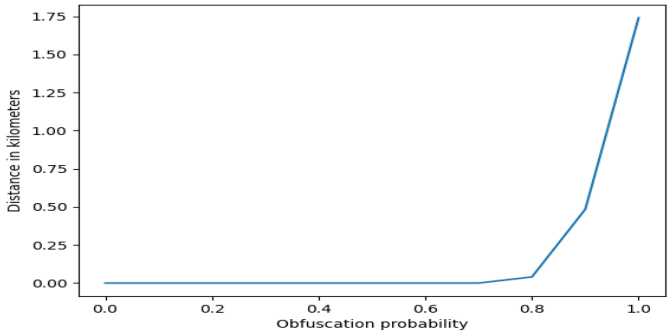

Fig. 3: Effect of obfuscation on inferring workplace location

4) *Privacy-Utility Trade-offs*: To measure utility loss, we decided to check what percentage of the queries were obfuscated during usage. In Figure 4, we can see that the utility loss follows a linear trend with regards to the value of p.

IV. Cell Fingerprinting via Network Traffic Analysis

A. Implementation details

Unfortunately, we did not manage to fully complete the last part of this analysis of the SecretStroll system. Nevertheless, here we present the feature extraction we had in mind for the data we would have collected. We followed the lectures we had in order to find a fitting set of features to be used in the classifier. Those features are the following:

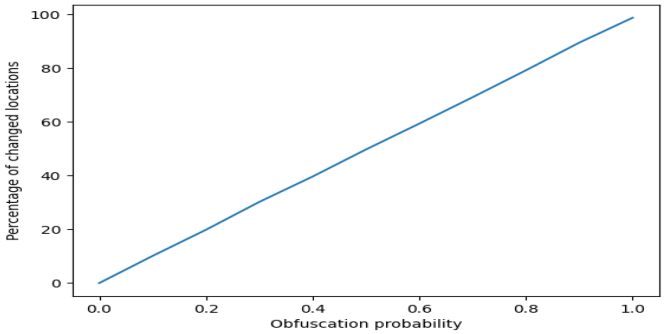

Fig. 4: Utility loss evaluation

1) Total number of packets.
2) Number of incoming packets.
3) Number of outgoing packets.
4) Ratio between the incoming packets and the total number of packets.
5) Ratio between the outgoing packets and the total number of packets.
6) Total number of bytes exchanged.
7) Number of incoming bytes.
8) Number of outgoing bytes.
9) Ratio between the incoming bytes and the total number of bytes.
10) Ratio between the outgoing bytes and the total number of bytes.
11) Average packet rate, total.
12) Average packet rate of incoming packets.
13) Average packet rate of outgoing packets.
14) Average size of incoming packets, in bytes.
15) Average size of outgoing packets, in bytes.

B. Evaluation

Since we do not have a finished classifier, we unfortunately do not have an evaluation.

C. Discussion and Countermeasures

Unfortunately, since we did not finish the implementation of the classifier, we cannot discuss its performance or determine the best countermeasures in our case.

V. Contributions

Alexandre Santangelo carried out the implementation of Attribute-Based Credentials in section II of this report. Alexandre Santangelo also tested and evaluated the implementation. Andrej Milicevic carried out the attacks which were done in section III, as well as designed the defence mechanism. Andrej Milicevic also performed the privacy gain and utility loss evaluations in said part. Andrej Milicevic devised the features that would have been used in the cell fingerprinting section.