CS-523 SecretStroll Report

Siim Markus Marvet, Changling Wang, Peiran Ma, Andrii Isakov

Abstract—**In this project we aim to implement an application to fetch nearby points of interest from a central server based on a clients location. For increased privacy we implement the authentication with Attribute Based Credentials (ABCs) using the Pointcheval-Sanders scheme.**

I. Introduction

In section II we implement, test and evaluate the performance of our ABC scheme. In section III we evaluate the privacy leakage from users' queries. In section IV we attempt to fingerprint the Tor traffic from our client application to disclose location information about users. Section V describes each authors contribution to this project.

II. Attribute-based credential

We chose option 1 as the attribute configuration in our implementation. First, the user's secret key **`_user_sk`** is included as required in the assignment handout, which serves as a secret user-defined identifier of the requested credential, and is generated as a random integer on $[1, p-1]$, where p is as defined in the ABC guide. On the other hand, **`_username`** is included as an issuer-controlled identifier of the issued credential. All offered subscriptions should be included as attributes, as the size of public keys is proportional to the number of attributes in the credentials. If we only include the subscriptions purchased by the user, anyone will be able to infer the number of subscriptions purchased by a user using that public key and potentially distinguish between users with different subscriptions, even if they have only revealed the same set of subscriptions. Furthermore, the order of all attributes in the credentials must be the same across all credentials issued to all users, no matter which subset of services is subscribed by the users. Since the issuer should have control over which subscriptions are granted to which users, the subscriptions are all issuer-defined.

In summary, for a set of n offered subscriptions $S = \{a_1, a_2, .., a_n\}$, the attribute configuration is **`[_user_sk,`** a_i **`for`** a_i **`in`** S**`, _username]`**. And with this configuration, for a user with subscriptions $U \subset S$, the attribute values will be **`[_user_sk,`** a_i **`if`** $a_i \in U$ **`else b'' for`** a_i **`in`** S**`, _username]`**.

In our attribute value scheme, all values are of type **`bytes`**, and they are mapped to the exponents in the ABC scheme by interpreting their **SHA256** hash value as an integer. Since the order of the groups used by **`petrelic`** is approximately 255-bit long, **SHA256** serves as a good candidate for mapping the attribute values to exponents.

When querying POIs from the server, the clients only need to disclose the corresponding subscription(s) to the server. For example, a client subscribing to **restaurant** and **dojo** only needs to reveal their subscription to **restaurant** if they are only querying POIs of type **restaurant**. This guarantees that minimum information is shared in each request, as any further reduction in the information shared would result in the server being unable to check whether the user is qualified for the query.

A. Zero-Knowledge Proofs

The zero-knowledge proof scheme, as well as its Fiat-Shamir transformation, is based on Schnorr's proof of identification (in the credential issuing phase) and Schnorr's signature (in the attribute disclosing phase) as presented in lecture 5. However, since we need a proof for a vector $\vec{x}$ instea of a scalar x, the extension described in [1] is implemented. Using the notations in the lecturte slides, the scheme is:

$$h = \prod_{i=1}^{n} g_i^{x_i}$$

$$\vec{r} = \text{rand}(\mathbb{Z}_p^n), R = \prod_{i=1}^{n} g_i^{r_i}$$

$$c = H(\vec{g}||h||R)$$

$$\vec{s} = \vec{r} + c\vec{x}$$

Similarly, for Schnorr's signature:

$$c = H(\vec{g}||h||R||m)$$

B. Test

The implementation is tested at both the PS-ABC scheme layer and the stroll client/server layer. In addition, the basic implementation of the PS signature scheme is also tested, although it is never directly used in the implementation of the stroll protocol.

At the PS-ABC scheme layer, the failure test cases involve tampering with hidden attributes or messages to sign during the attribute disclosing phase, and tampering with the user-defined or issuer-defined attributes during the credential issuance phase.

At the stroll protocol layer, the failure test cases involve tampering with all or part of the attributes to disclose, as well as tampering with the message.

C. Evaluation

To evaluate our implementation, we measured the computational cost and communication cost of every part of this system using different numbers of server attributes (from 40 to 150), user attributes (from 5 to 40), and disclosed attributes (from 5 to 40, with the two previous parameters taking the maximum value). All data presented below are the **mean $\pm$ 1 standard deviation of measurements from 10 independent runs** of the protocol.

Before more detailed analysis, it is worth mentioning that the dominant component in all the computations involved in the scheme is always the exponentiations of the group members, and since the exponents are either randomly generated (during key generation) or derived from the hash values of the attributes (which can be deemed as pseudo-random numbers), the computation cost is in general linearly related to the number of exponentiations required. Similarly, the communication cost is dominated by the representations of the attributes involved, which generally scales linearly with respect to the number of attributes involved. However, when attributes are transmitted in plain text, the scaling factor is the average size of the attributes (denoted as **`attribute_size`**), whereas in Schnorr's proofs, each attribute is represented as a (seemingly) random exponent $s \in \mathbb{Z}/p\mathbb{Z}$, whose representation is on average of size $\sim \log(p)$.

For simplicity, for the following analysis, we denote the computation time as T, the communication cost as S, the total number of valid attributes as #**`attributes`**, the number of user-defined attributes as #**`user_attributes`**, the number of issuer-defined attributes as #**`issuer_attributes`**, the number of disclosed attributes as #**`disclosed_attributes`** and the number of hidden attributes as #**`hidden_attributes`**.

1) Key Generation: The number of y_i' s (and the resulting Y_i' s and $\tilde{Y}_i$' s) in the generated key pairs is proportional to #**`attributes`**, which shall dominate the size of the key pairs. Similarly, T_{KeyGen} is dominated by the exponentiations to compute Y_i' s and $\tilde{Y}_i$' s. Hence, during key generation, both T_{KeyGen} and S_{KeyGen} (public key size) shall be $O(\#\texttt{attributes})$, which is consistent with our measurements in 1.

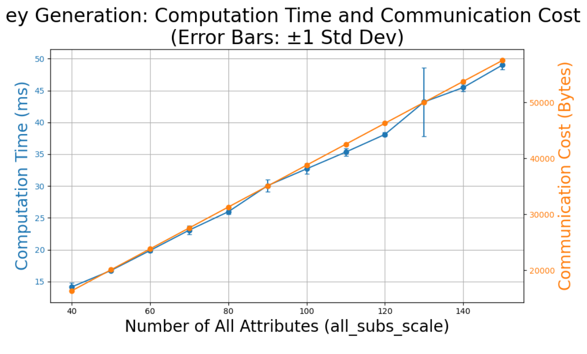

Fig. 1: S_{KeyGen} and T_{KeyGen} vs #**`attributes`**

2) Credential Issuance: The credential issuance process consists of 2 subroutines on the client side and 1 subroutine on the server side, as well as a request-response communication between the 2 parties. $T_{\text{CredIssue}}$ is defined as the sum of execution time of all 3 subroutines, and $S_{\text{CredIssue}}$ is defined as the sum of size (in bytes) of the request and response. The size of the request is dominated by the size of the Schnorr's proof of identification, which is $O(\#\texttt{user\_attributes} \cdot \log(p))$, while the size of the response shall be $O(\#\texttt{issuer\_attributes} \cdot \texttt{attribute\_size})$, due to the issuer attribute mapping in the response. Hence, the total $S_{\text{CredIssue}}$ shall be $O(\#\texttt{attributes})$, which is consistent with our measurements in 2. However, since **`attribute_size`** is smaller than $\log(p)$ in our evaluation data, we also see a linear increase in $S_{\text{CredIssue}}$ with respect to #**`attributes`**.

Similarly, $T_{\text{CredIssue}}$ shall also be $O(\#\texttt{attributes})$, which is consistent with our measurements in 3.

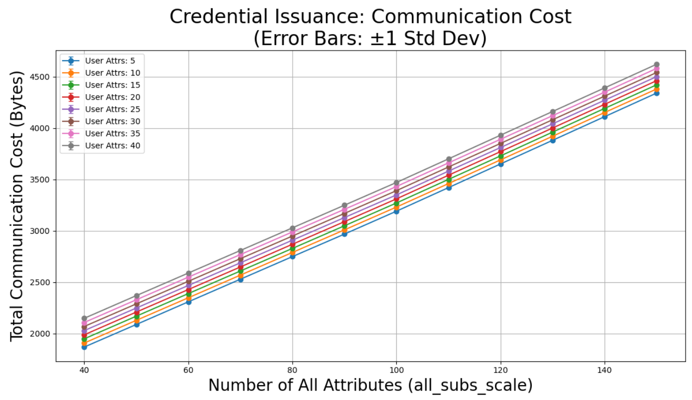

Fig. 2: $S_{\text{CredIssue}}$ vs #**`attributes`**, for different #**`user_attributes`**

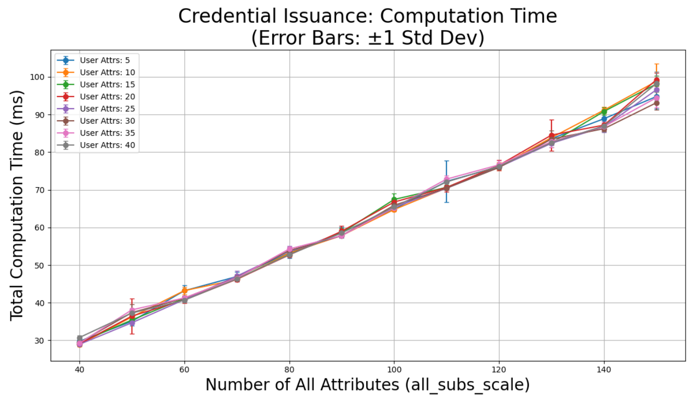

Fig. 3: $T_{\text{CredIssue}}$ vs #**`attributes`**, for different #**`user_attributes`**

3) Credential Showing: For credential showing, S_{CredShow} shall be $O(\#\texttt{hidden\_attributes} \cdot \log(p) + \#\texttt{disclosed\_attributes} \cdot \texttt{attribute\_size})$, and the computation cost shall be $O(\#\texttt{hidden\_attributes})$. Since there is no difference in the treatment of user-defined attributes and that of issuer-defined attributes, changing the ratio between them shall have no impact on either of the metrics.

Hence, for fixed #**`disclosed_attributes`**, S_{CredShow} shall scale linearly with respect to #**`attributes`**, which

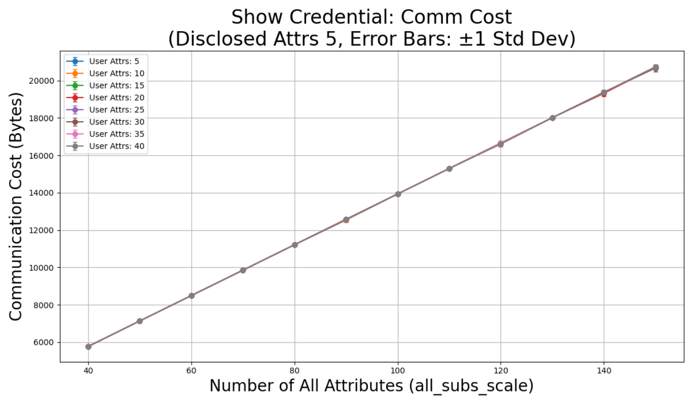

Fig. 4: S_{CredShow} vs #**attributes**, with #**disclosed_ attributes** $= 5$, for different #**user_attributes**

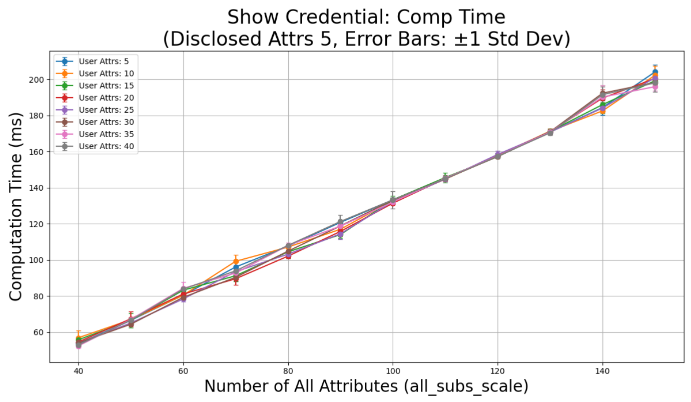

Fig. 5: T_{CredShow} vs #**attributes**, with #**disclosed_ attributes** $= 5$, for different #**user_attributes**

is consistent with our measurements in 4 and 5. On the other hand, with fixed #**attributes**, #**hidden_attributes** decreases as #**disclosed_attributes** increases. And since $\log(p) >$ **attribute_size**, the decrease in the size of Schnorr's signature dominates the increase in the size of attribute mapping, and therefore the overall S_{CredShow} shall also decrease as #**disclosed_attributes** increases. The analysis is consistent with our measurements in 6.

4) Verification: To conduct the verification, the verifier needs to first reconstruct the commit from the disclosed attributes, whose computational cost is $O(\#$**disclosed_attributes**$)$, and then it needs to verify the given Schnorr's signature, which comes at a cost of $O(\#$**hidden_attributes**$)$. Together, T_{Verify} for the verification is $O(\#$**attributes**$)$, which is consistent with our measurements in 7. On the other hand, T_{Verify} should remain somewhat constant as long as #**attributes** remains unchanged, which is consistent with 8 (notice that scale of the y-axis is pretty narrow).

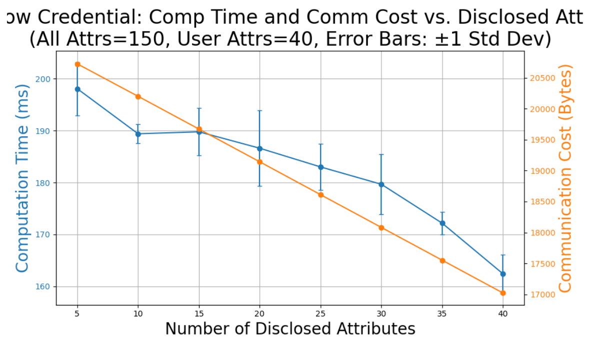

Fig. 6: S_{CredShow} and T_{CredShow} vs #**disclosed_ attributes**, with #**attributes** $= 150$

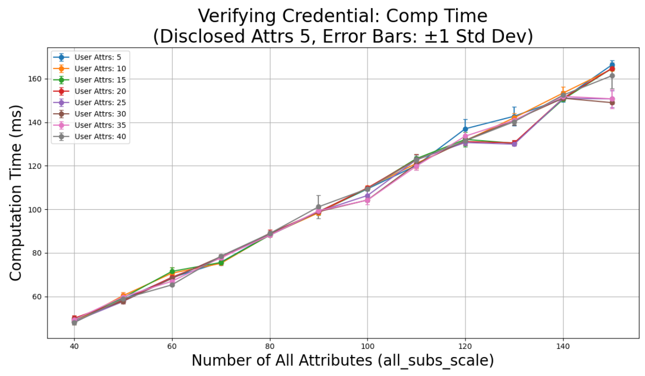

Fig. 7: T_{Verify} vs #**attributes**, with #**disclosed_ attributes** $= 5$, for different #**user_attributes**

III. (De)Anonymization of User Trajectories

A. Privacy Evaluation

The adversary is modeled as an honest-but-curious service provider (server). The server is assumed to honestly provide the service (serving POI's to querying users), but tries to collect and extract as much information about its users as possible for user profiling. Since the analysis is ultimately for building a privacy-preserving location-based service, such an assumption models well the server's capability given its

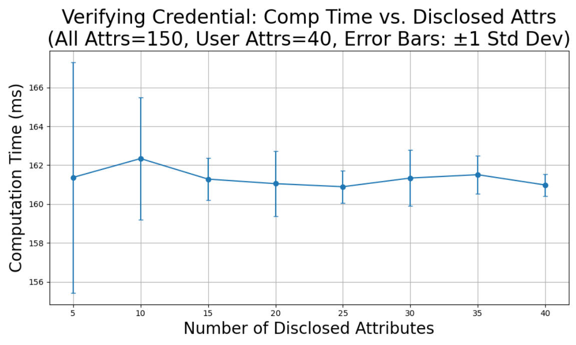

Fig. 8: T_{Verify} vs #**disclosed_attributes**, with #**attributes** $= 150$

intention, that is, to honestly provide service while reducing users' privacy leakage.

Specifically, the adversary is assumed to be able to record all queries it receives and perform data analysis on the records and other information it has. In other words, the adversary has access to *pois.csv* and *queries.csv*.

To perform the attack, the query records are categorized by their **`timestamp`** into 3 groups, which are, respectively, queries made on weekends, queries made during working hours (9:00 - 17:00) of weekdays, and queries made during nonworking hours of weekdays. Then, we plot the locations of queries, and get 9.

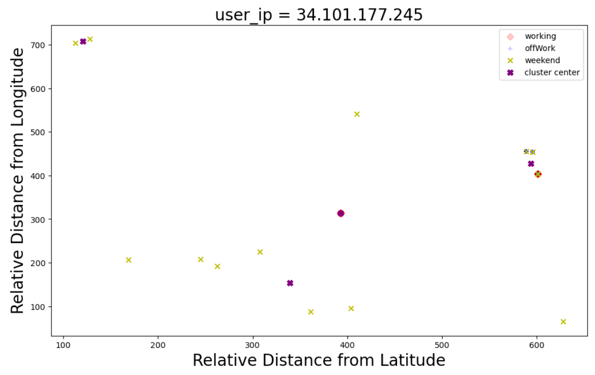

Fig. 9: Location of Queries from a User

It is pretty obvious from 9 that there is a single location where the user only makes queries during working hours of weekdays. And there is exactly one match of coordinates of this point in *pois.csv*, of **`poi_type`** = **`company`**. It turns out that a similar point (of **`poi_type`** $\in$ {**`company`**, **`laboratory`**, **`office`**}) exists for all users in the given dataset, enabling us to identify the workplace of all users in the dataset.

It is also notable that some queries are made at the same location, both on weekends and during nonworking hours of weekdays (and sometimes even during working hours). For this specific user, if we look up in *pois.csv* the coordinates of the intersection point of all 3 groups of locations, we get a record of **`poi_type`** = **`villa`**. It turns out that for all users in the given dataset, by exact coordinates matching, we can find a unique POI of **`poi_type`** $\in$ {**`appartment_block`**, **`villa`**}, allowing us to identify the residential address of the users. Similarly to 9, all simulated users tend to make a lot of queries in the vicinity of their home.

These observations imply that the adversary can use clustering algorithms (K-means) to get an approximation of the location of a user's home and workplace, as marked out by the purple crosses in 9. It is notable that this attack is relatively robust against naive defenses by adding noise to the coordinates in queries, as it essentially exploits the statistical characteristics of the data.

As shown above, it is quite easy to extract the location of the simulated users' home and workplace.

Rounding	Metric	Mean	Std
2-digits	Utility	0.775	0.104
	Attack success-rate	0.079	-
3-digits	Utility	0.978	0.022
	Attack success-rate	0.916	-

TABLE I: Evaluation of rounding-based defence - per-query

With this information, the adversary could further infer the social relationships between users, such as which users are colleagues, based on their workplace (see `part2/defence2_and_attacks.ipynb`). Furthermore, it is even possible to infer the users' financial status with some background information and assumptions. For example, with location of home and workplace and the data from *pois.csv*, it is possible to infer the type of housing and the occupation of a user, which are usually good indicators of one's financial status.

B. Defences

We model our defences towards the secretstroll service provider, who is a passive internal adversary, that can read all the queries made by the users in plaintext and knows the location of all POIs.

Following the intuition from the lecture, we define privacy as every time, when the attacker does not succeed in recovering user's private information. Specifically, for the attack success-rate S, we define privacy P as follows: $P = 1 - S$.

We implement a user-side defence (see notebook `part2/rounding_defence.ipynb`), where the client application rounds the coordinates to two decimal places before sending them to the service provider. The adversary then tries to recover the exact location where the query was made from by searching for the closest POI to the coordinates. We define utility as the proportion of "original" POIs, that are returned by the server when querying from the obfuscated coordinates. For the results please see Table I. For comparison we also tested rounding to 3 decimal places and observed that while the utility improved from 77.5% to 97.8%, the attack success-rate rose significantly more from 7.9% to 91.6%, highlighting the utility-privacy tradeoff in designing this application.

As discussed above, apart from per-query based attacks, an adversary could also try to infer the location of a user's home or workplace using clustering algorithms, so we also evaluated our defence against such attacks. The attack is simulated by running K-Means clustering (with **`n_clusters`**=3) on the dataset with the defence applied, and the algorithm will output 3 cluster centers. A real adversary will need to plot things out as is done in 9 and guess which cluster center corresponds to the approximation of the location of home (and workplace), and the guess may be incorrect. But for evaluation and analysis, we can assume the worst case, that is, the guess is always correct, and the adversary always picks the closest cluster center to the ground truth location. Then, the adversary would predict the closest POI of type **`villa`** or

Rounding	Home	Workplace
2-digits	65	96
3-digits	139	184

TABLE II: Evaluation of rounding-based defence - statistical

`apartment_block` as the user's home, and similarly for the workplace. II presents the number of correct predictions the adversary can make with this attack strategy, and it is obvious that such statistical attacks are much harder to defend against.

We also tried cropping the coordinates to three digits after the decimal point and randomizing the last digit (table III). The resulting cloak area is roughly 1.1km x 1.1km.

State	Longitude	Latitude
original	46.54673993140806	6.57737743004016
cropped	46.546	6.577
randomized last digit	46.541	6.575

TABLE III: Defence strategy with randomness

The measured attacker success rate is 21/200 and 50/200 for home and work respectively. And the utility was roughly 45%. The privacy is higher, but the utility is lower, so based on the ratio between privacy and utility of each defence strategy, we decided to go with the first one.

IV. Cell Fingerprinting via Network Traffic Analysis

A. Implementation details

We started by building the docker containers and running the initial setup for both the `server.py` and the `client.py` programs. After that we wrote a script `part3/make_queries.sh` that sequentially queries each of the 100 cells while also recording the start and end timestamp. Those timestamps along with the cell_id would be used later to label packets that were sent/received during the query. We also added functionality to run the script multiple times to collect more data.

To perform the fingerprinting attack we needed to capture network traffic coming from and going to the client docker container. To that end we used tshark[2] - a command line interface for the network traffic capturing tool Wireshark. We used the tool to capture traffic on the (bridge) network interface that is created for the docker containers and filtered the traffic to only those packets that concerned the client. After capturing and saving the network dump into a pcap file, we used tshark a second time to parse out only the packets' timestamps, source IPs, destination IPs and size. The tshark commands used are outlined in the README.md file for part3 and the code for feature extraction can be found in `part3/part3_analysis.ipynb`.

It took us roughly 8 hours to collect 2500 samples (100 cells * 25 iterations) for our classifier. To extract features for ML we took the output of tshark and grouped the network packets into flows corresponding to each individual cell query. We then calculated 6 features for each flow: *nr of packets sent*, *nr or packets received*, *total nr of packets*, *nr of bytes sent*, *nr of bytes received* and *total nr of bytes*. We used the provided skeleton code in `fingerprinting.py` without significant changes to train our model on those 6 features.

Metric	Mean	std
Accuracy	0.818	0.024
Precision	0.832	0.028
Recall	0.818	0.024
F1 Score	0.804	0.027

TABLE IV: Evaluation results over 10-fold cross validation

B. Evaluation

We ran a 10-fold cross validation of our model and averaged the accuracy, precision, recall and F1-score from those 10 runs. Overall our model achieved an accuracy of 81.8% in predicting which of the 100 map cells was being queried based on the network flows. For additional results, please refer to Table IV.

C. Discussion and Countermeasures

The model's performance was particularly noteworthy, considering it was tasked with differentiating between 100 separate classes. Our classifier was able to reach this accuracy thanks to the different network patterns generated by the queries, however it fell short of perfect due to multiple reasons. Firstly Tor uses cover traffic in the form of connection-level padding and circuit-level padding to mask real flows and provide plausible deniability [3]. This leads to a constant background noise that is impossible for an external adversary to distinguish from real traffic and therefore leads to slightly differing traffic profiles for what would otherwise be identical queries. Secondly we noticed occasional network stability issues when using Tor, which caused repeat packages to be sent and therefore also altering the profile. Those issues were particularly problematic with grid IDs, where even minor fluctuations in network traffic made it difficult to differentiate between two separate queries.

A countermeasure to fingerprinting could be to change the way our client and server applications communicate - we could add padding to queries and responses to make them look identical in size and count, leading to indistinguishability between different queries.

V. Author contribution statement

Changling Wang implemented the ABC scheme and tested it, while Peiran Ma ran the performance evaluation (Section II). All group members worked on the (de)anonymization of user trajectories (Section III). Siim Markus Marvet implemented cell fingerprinting via network traffic analysis (Section IV).

References

[1] Pedersen commitments. [Online]. Available: https://www.zkdocs.com/docs/zkdocs/commitments/pedersen/#proof-of-knowledge-of-secret

[2] tshark. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[3] Tor specification. [Online]. Available: https://spec.torproject.org/padding-spec/overview.html