Privacy-Preserving Aggregation of Time-Series Data

Andrea Pergetti & Janneke van Oosterhout

January 2023

1 Introduction

This document contains the implementation details of our project for the Applied cryptography project seminar course. The topic of our project is *Privacy-Preserving Aggregation of Time-Series Data* and our goal is to implement the protocol described in the paper [1].

First, we will discuss the problem discussed in the paper. After that, the details of the implementation are described together with the results and some challenges we faced. We will end this document with possible further improvements and a conclusion of our project.

2 Problem description

A data aggregator is an organization that collects data from one or more sources, to study patterns or statistics over a population. An important challenge in these applications is how to protect the privacy of the participants, the people that upload the data to the data aggregator, especially when the data aggregator is untrusted. An untrusted data aggregator can learn desired statistics over multiple participants' data, without compromising their privacy. [1]

Hence, the main problem discussed in the paper is how to protect the privacy of the participants when they are dealing with an untrusted data aggregator.

2.1 Solution

The paper proposes a construction that allows a group of participants to periodically upload encrypted values to a data aggregator, such that the aggregator is able to compute the sum of all participants' values in every time period, but is unable to learn anything else.

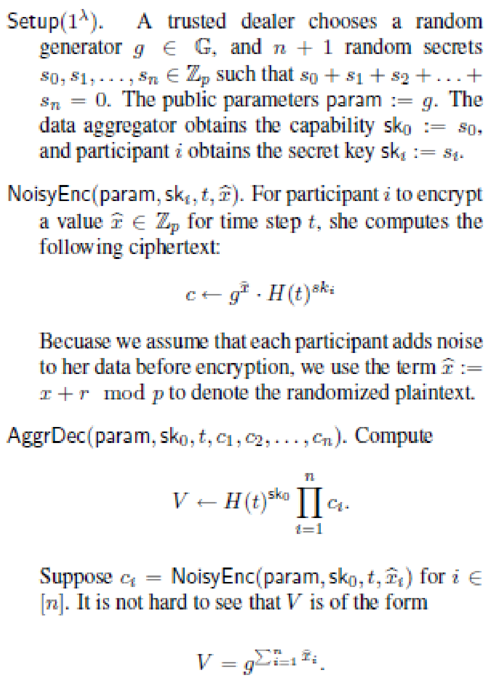
The construction in the paper achieves strong privacy guarantees using two main techniques: [1]

- Aggregator oblivious: aggregator is unable to learn any unintended information other than what it can deduce from its auxiliary knowledge and the desired statistic.
- Distributed differential privacy (for each individual participant): the statistic revealed to the aggregator will not be swayed too much by whether or not a specific individual participates. Therefore, users may safely contribute their encrypted data, as presence in the system will not lead to increased risk of privacy breach. It is called distributed differential privacy because the noise in the released statistic is collected from all participants.

To achieve strong privacy guarantees, each participant generates independent random noise and computes a noisy version of her data.

2.1.1 Construction

The construction discussed in the paper [1] consists of three phases: setup, NoisyEncrypt and AggregatorDecrypt, which can also be seen in Figure 1.



$\mathsf{Setup}(1^\lambda)$. A trusted dealer chooses a random generator $g \in \mathbb{G}$, and $n+1$ random secrets $s_0, s_1, \ldots, s_n \in \mathbb{Z}_p$ such that $s_0 + s_1 + s_2 + \ldots + s_n = 0$. The public parameters $\mathsf{param} := g$. The data aggregator obtains the capability $\mathsf{sk}_0 := s_0$, and participant i obtains the secret key $\mathsf{sk}_i := s_i$.

$\mathsf{NoisyEnc}(\mathsf{param}, \mathsf{sk}_i, t, \widehat{x})$. For participant i to encrypt a value $\widehat{x} \in \mathbb{Z}_p$ for time step t, she computes the following ciphertext:

$$c \leftarrow g^{\widehat{x}} \cdot H(t)^{sk_i}$$

Becuase we assume that each participant adds noise to her data before encryption, we use the term $\widehat{x} := x + r \mod p$ to denote the randomized plaintext.

$\mathsf{AggrDec}(\mathsf{param}, \mathsf{sk}_0, t, c_1, c_2, \ldots, c_n)$. Compute

$$V \leftarrow H(t)^{\mathsf{sk}_0} \prod_{i=1}^{n} c_i.$$

Suppose $c_i = \mathsf{NoisyEnc}(\mathsf{param}, \mathsf{sk}_0, t, \widehat{x}_i)$ for $i \in [n]$. It is not hard to see that V is of the form

$$V = g^{\sum_{i=1}^{n} \widehat{x}_i}.$$

Figure 1: Construction discussed in the paper to solve the problem [1]

In the setup phase, a trusted dealer chooses a random generator and generates random secrets, one for each participant and also one for the date aggregator. After the setup phase between all participants and the data aggregator, no further interaction is required except for uploading a noisy encryption to the data aggregator in each time period. In NoisyEncrypt a participant encrypts their noisy value (noise is already added to the real value) for time step t. The output is the ciphertext c. In AggregatorDecrypt, the aggregator computes V. To decrypt the sum, you have to compute the discrete log of V base g.

2.1.2 Theoretical challenges

Some theoretical challenges addressed in the paper are: [1]

- Final statistic has to have sufficient randomness to guarantee each individual's privacy. If there are compromised participants that collude with the data aggregator and reveal their data or randomness, you still have to ensure that the remaining uncompromised participants' randomness is sufficient to protect their privacy.
- Algebraic constraints: you have to work within the algebraic constraints induced by the cryptographic construction.
- Small plaintext space: the cryptographic construction imposes the plaintext space to be small. But this ensures that the probability of an overflow is small, so that the aggregator can successfully decrypt the noisy statistics with high probability.
- Dynamic joins and leaves: if a participants joins or leaves the system, the setup phase needs to be performed again. So you want to make sure that the set of participants is not changing too often.
- Node failure: if a participant fails to upload their encrypted values in a time period, the data aggregator cannot compute the statistic successfully.

3 Implementation

3.1 Roadmap

- Research libraries & background (Both): November 15 - November 25
  - Library research based on implementation division
  - Background research based on 'difficult' parts of the paper
- Discuss implementation details and what input should look like (Both): until December 2
- Implementation: First week of December - January 15
  - Input generator (Janneke)
  - Protocol - Encryption, hash, decryption, big step giant step (Andrea)
  - Debugging (Both)

- Finish project (Both): January 15 - January 22
  - Presentation (Janneke: problem, challenges. Andrea: approach, results.)
  - Testing
  - Documentation

3.2 Approach used

In every steps of the protocol we choose specific implementation aspect that are explained below:

- In the setup function the first task was to use a cyclic group for which the Decisional Diffie-Hellman problem is hard. To do it we first generate a random prime number p and from it we compute q as 2*p+1 which it's also prime (called Sophie Germain prime). [2] From this prime we can have a cyclic group, called G, of prime order p with the form $G = \{x^2 \ mod \ q | 0 < x <= p\}$.
  Then we have to select a random generator from G so we pick a random number in the set G shown before. Then we create the secret keys for the aggregator and the users as random numbers in the additive group modulo p such that the sum of all the secret key modulo p is 0 (since the order of the additive group is p). This will be very important for be able to obtain the correct result after the decryption.

- For the input generator we first select a random values in the multiplicative group modulo p, which are the noise. Then we select random values in the multiplicative group modulo p, which are the user inputs, such that the sum of all these values is less or equal p, this is done to be able to obtain an element of the group in the decryption phase. After we sum the input of each user with a noise value previously generated.

- We also have to implement a function H that should take an integer as input and output an element in the group G modelled as a random oracle. So we first use a one-way hash function (SHA-256) on the input obtaining the hash in an hexadecimal encoding, we convert it to an integer modulo p and then we map it to a element of group G. [3]

- For the encryption we just use the formula $c = g^{\hat{x}} \cdot H(t)^{sk_i} \ mod \ q$ for each user randomized input.

- For the decryption we use the formula $V = H(t)^{sk_0} \prod_{i=1}^{n} c_i$. From this formula we obtain $V = g^{\sum xi}$. Since we are interested in the sum of the input, that is the exponent of the previous equations we should compute the discrete log of V base g. To do it we use an algorithm called Baby step Giant step. [4]

3.3 Implementation challenges

We also faced some challenges while implementing the protocol.

- How to generate the random noises. This is not really discussed in detail in the paper, so for now we tried our own way. A further improvement can be to fix this and generate the noise correctly.
- Generate the secrets. Generating the secrets itself was not really the problem but the secrets should sum up to 0 mod p to be able to get the correct output in the end. We managed to get it sum to 0 but the product here should be 1, which was not the case. We couldn't identify the problem so asked one of the teachers and we had to make sure the secrets sum to 0 mod $p-1$. Eventually, we fixed the group creation such that the order of the group is p and the sum of the secret is indeed 0 mod p (as discussed in the paper).
- Decryption. We struggled a bit with getting the correct output in the decryption. One of the reasons was the secrets but also the discrete log problem made it difficult to get the correct output.

3.4 Results

This are the results that we obtain from a run of the protocol:
general parameter $\rightarrow$ bits of prime number p = 16 bit, number of participants = 3, number of timestamp = 1
Prime numbers p = 50513 and q = 101027
Generator = 57063
Array of secret keys (the first is the aggregator's key and the others are of the 3 participants) = [13943, 20851, 42566, 23666]
Initial input = [26538, 3247, 2615]
Randomized input = [2523, 40749, 39641]
Encrypted input = [76048, 88883, 9566]
Final result (decrypted sum) = 32400

In the table below we show the execution time (in seconds) of each function compute as the average time of 5 run of the algorithm for different number of bits for the generated prime number p.

	setup	input gen	encryption	decryption
16 bits	2.325	0.658	0.206	0.026
20 bits	14.32	28.107	20.414	2.13

3.5 Problems

The main problem of our implementation now is the computational complexity. The setup function takes 16 bits for the prime number p, and then the protocol runs in a reasonable amount of time, as can be seen in the result above. When we increase the number of bits, so if we use more than 16 bits, the input generation function and the encryption function take too much time. In the input generation function it will be because it has to generate the random values and we do this in a loop. If a condition is not met, we generate again so this time can increase easily. In the encryption function the heavy part will probably be the computation itself.

3.6 Testing

To test our code, we added several tests:

- `test_sk`: checks if the sum of the secret keys is 0 modulo p.
- `test_product`: checks if the product of the hashes raised to the secret keys is 1.
- `test_result`: checks if the result obtained is equal to the expected result.
- `test_dlog`: checks the discrete log, so if big step giant step algorithm worked correctly.

4 Further improvements

We also want to mention some possible improvements that can be done to this protocol:

- The first is to use Elliptic Curve points that can make become the protocol faster and more secure.
- The second is to try other discrete log solver algorithm different from the Baby step Giant step, like the Pollard's lambda function, and evaluate which is the most efficient one for this protocol.
- The computational complexity can also be improved compared to our implementation now.
- The last one is to fix the randomization function because now we just adding a random noise but it may not be the best way to do it. Indeed in some scenario where there are some compromised users it can be possible for a malicious agent to find out the data of the users that are not compromised yet. Using the addition of noise in a different way can help avoid it.

5 Conclusion

We managed to implement the proposed construction discussed in the paper. However, some parts of our code could still be optimized in the future, for example we could fix the complexity or improve the randomization. Overall, we sticked to our roadmap that we defined before the start of this project. The implementation was as difficult as we expected but we faced some unforeseen challenges that caused a lot of time (for example the one with the secrets that should sum up to 0). We are happy with the collaboration on this project and also with the feedback/answers to our questions that we received from the teacher.

References

[1] Elaine Shi and T-H Hubert Chan and Eleanor Rieffel Fxpal and Richard Chow and Dawn Song, *Privacy-Preserving Aggregation of Time-Series Data*, `http://elaineshi.com/docs/ndss2011.pdf`

[2] Wikipedia, *Safe and Sophie Germain primes*, `https://en.wikipedia.org/wiki/Safe_and_Sophie_Germain_primes`

[3] *hashlib – Secure hashes and message digests*, `https://docs.python.org/3/library/hashlib.html#hash-algorithms`

[4] Wikipedia, *Baby-step giant-step*, `https://en.wikipedia.org/wiki/Baby-step_giant-step`