University of Rouen Normandie & INSA Rouen Normandie

Master Data Science and Engineering
Data Science Track

Master thesis

Robustness for GOssip Private Averaging

Author
Alexandre Huat
alexandre.huat@insa-rouen.fr

Supervisor
Dr Jan Ramon
INRIA Lille Nord Europe
MAGNET team
6 rue Héloïse
Bâtiment B
59650 Villeneuve d'Ascq
jan.ramon@inria.fr

28 August 2018 (v1)
10 November 2018 (v2)

Acknowledgments

I would like to sincerely acknowledge my supervisor Dr Jan Ramon for having given me the opportunity to carry out my master internship at INRIA Lille Nord Europe. In the absence of all the ideas he gave me, I could not have produced so much work. He reoriented my research when necessary and his critical mind was source of progress. I also thank him and Aurélien Bellet for their comments on my thesis that contributed to its quality.

I am also grateful to Cesar Sabater, with whom I worked through both theoretical and technical issues, in particular cryptographic ones. I would also like to thank Nathalie Vauquier who coded almost all our prototype implementation. She was a model of simplicity for me, and her assistance allowed me to focus on theoretical aspects of this internship. Together with, I am thankful to Carlos Zubiaga Pena, not only for his contributions to our prototype and for having raised relevant questions about our work, but also for his friendliness.

I must also thank all the people of the MAGNET team for their agreeableness and the enriching discussions we had. In particular, I thank team leader Prof. Marc Tommasi, for his welcome and his care for my work comfort.

As this thesis marks the end of my master studies, I also want to acknowledge my engineering school and university teachers, particularly Prof. Gilles Gasso and Dr Benoît Gaüzère who gave me passion for science and for work and supported me in difficult times over the last years. Finally, I express my warm thanks to my family, my many friends and Sofia, above all, for their unconditional support over the last six years.

To my mother.

Contents

Chapter 1

Introduction

1.1 Research topic

‘Personal data’ means any information relating to an identified or identifiable natural person (‘data subject’); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person; (Art. 4 GDPR — Definitions)

In recent years, companies, governments and citizens expressed the need of improving the protection and the self-controlling of their personal and/or private data in an increasingly data-driven world. As an illustration, more than forty data breaches affecting hundreds of millions of people and costing hundreds of millions of dollars were reported through the world in 2017; more than any year prior [8]. Also, in a common effort, old European regulations on data protection, Data Protection Acts (DPAs), evolved to the General Data Protection Regulation (GDPR)[1] whose entry into force was on 25 May 2018. As a matter of political, economical and cultural relations, this change already began to impact other continents. Evidently, the new threats to privacy are forcing us to response and adapt.

In this context, another element to consider is technological progress. In particular, we should pay attention to growing ubiquitous technologies (internet of things, smart cars, smart cities, smart homes, *etc.*) that require networking in open spaces potentially exposed to malicious listeners or other crackers. For example, we can imagine a smart road covered with smart cars that permanently exchange data in order to optimize their routes (*e.g.* for safety reasons). Then, suppose that some of the data needed to perform the optimization tasks are private (*e.g.* as data of each company or even of each car passenger). It becomes obvious how important are secure and privacy-preserving networking protocols in this use case.

In the last years, this kind of problem gained the interest of researchers of the MAchine learning in information NETworks (MAGNET) team of the French National Research Institute of Computer Science and Automatic Control (INRIA) of Lille. To answer it, Dellenbach, Ramon, and Bellet [11] recently conceived GOssip Private Averaging (GOPA), a decentralized protocol to deterministically compute the average of a private data shared by users of a peer-to-peer (P2P) network without revealing it. This thesis focuses on this protocol, which was made as a lower component of machine learning algorithms in need of average calculations (*e.g.* decision trees, kernel-based algorithms). In MAGNET, the design of GOPA is part of two projects: Personalized and decentrAlized MachinE Learning under constrAints (PAMELA), funded by

[1]`https://ec.europa.eu/info/law/law-topic/data-protection_en`

the French National Research Agency (ANR), and MyLocalInfo under State-Region Planning Contract (CPER).

The decentralized setting of GOPA is motivated by the robustness it offers. Indeed, in the centralized setting, a corrupted central entity could deceive all users. For instance, suppose a network of users who connect through a server. Suppose that users send their private data only to the server and that they delegate the average calculation to it. Then, if the server were dishonest, it could just send a fake average to users but they could not detect the fraud. But also, the server could share the users' private data with other parts. In contrast, GOPA is made such that users neither delegate the average calculation nor share their private data to anyone. Instead, they compute the average by their own, through successive pairwise exchanges of values that guarantee privacy. Due to the way it operates, GOPA can be classified as a secure gossip protocol.

Apart from the privacy challenges theoretically solved within GOPA [10], we must take into account practical issues in the objective of implementing the protocol. First of all, a major practical concern is raised by the imperfection of real applications, *i.e.* some users could drop out or fail, or some messages may be lost while the algorithm is running. Secondly, GOPA features a mechanism to detect malicious users who may try to jeopardize the correctness of the average calculations. But, this mechanism handles only one threat among many. Here comes my master internship, whose purpose is to improve the robustness of GOPA to dropping out users and malicious adversaries. By robustness, we mean the strength of privacy guarantees and the exactness of the average calculation. And by malicious adversaries, we mean any part whose purpose is to corrupt the protocol, either by skewing its outcomes or by taking it down. Also, in the next chapters we will generalize the problem of dropouts to the problem of *churns*, *i.e.* users' disconnections and arrivals.

1.2 Work context

Historically born in 1967, INRIA is at the cutting edge of research in France. It employs 2,400 members from 102 different countries and 1,400 PhD students who promote scientific excellence for technology transfer and society. INRIA counts 8 centres in Paris, Rennes, Sophia Antipolis, Grenoble, Nancy, Bordeaux, Lille and Saclay, which are organized in 184 'project-teams' totalling up to 4,400 scientific publications per year. Furthermore, with a total budget of 231 M€, the institute is engaged in many industrial partnerships that lead to innovations in such diverse fields as healthcare, transport, energy, communications, security and privacy, smart cities and factories of the future.

During this internship, I joined the MAGNET team which was born in 2013. It is a joint team between the University of Lille 3[2] and INRIA Lille Nord Europe. My work place was only INRIA, at half an hour from the Lille center. The MAGNET team counts 6 senior researchers, 1 Emeritus Professor, 3 PhD students, 2 postdoctoral researchers, 5 engineers, 1 administrative assistant, and we were 3 master students on an internship period of one semester (5 March to 31 August 2018). MAGNET is interested in data over large information networks and it covers subjects such as natural language processing (NLP), graph learning and privacy-preserving decentralized machine learning.

1.3 Report organization

This thesis follows up on the latest version of GOPA [10], attached in Appendix B. Chapter 2 presents this version and formulates the problematic of our study. Chapter 3 makes a state-of-

[2]within the Data Intelligence Group (DataInG) of the Research Center on Computer Science, Signal Processing and Automatic Control of Lille (CRIStAL) (National Center for Scientific Research (CNRS), Mixed Research Unit (UMR) 9189)

the-art review. Chapter 4 presents our contributions to the improvement of GOPA's robustness to churns and malicious adversaries. These theoretical results are then discussed in chapter 5. Also, chapter 6 presents a prototype implementation of the latter version of GOPA. And finally, chapter 7 concludes this thesis. Readers are also referred to the Lexicon chapter, p. 51, regarding the meaning of the acronyms and the mathematical symbols used in this document.

Chapter 2

Problem formulation

2.1 Introduction to GOPA

Let U be a set of users of a P2P network whose private values that they want to average are $(x_u)_{u \in U}$. Let $G = (U, E)$ be an undirected graph representing this network, where edge $\{u, v\}$ means that u and v can communicate with one another. GOPA aims to compute the average private value of users $\bar{x} \triangleq \frac{1}{|U|} \sum_{u \in U} x_u$ without any u revealing x_u. To do so, the protocol procedes in two phases detailed in the next subsections:

1. *randomization*, which fulfills the privacy requirements as well as giving an input to the averaging process;
2. *averaging*, in which users do the average calculation in a decentralized manner, thus emancipating from a potentially cheating service provider.

2.1.1 Randomization phase for privacy

At first sight, computing $\bar{x} \triangleq \frac{1}{|U|} \sum_{u \in U} x_u$ requires the knowledge of $(x_u)_{u \in U}$. However, if we take a closer look at the equation, it only requires that of $\sum_{u \in U} x_u$ and $|U|$.[1] Then, the principle of randomization is to compute values that differ from $(x_u)_{u \in U}$ but still sum to $\sum_{u \in U} x_u$. To achieve this, GOPA relies on the following algorithm.

Algorithm 1 (Randomization)**.**

1. For some pairs of peers $\{u, v\} \in E$, u and v jointly choose random *P2P noises* $\delta_{u,v}$ resp. $\delta_{v,u}$ such that
$$\delta_{u,v} = -\delta_{v,u}. \tag{2.1}$$
We say they *exchange* noises, and we call the graph resulting from these exchanges the *randomization graph*, $G_R = (U, E_R \subset E, u \mapsto x_u, (u, v) \mapsto \delta_{u,v})$. It is a directed graph whose node-labels represent users' private values and edge-labels represent noises. See Figure 2.1 as an illustration. In the sequel, we will denote $\mathcal{N}_u = \{v : (u, v) \in E_R\}$ the neighborhood of user u, $d_u = |\mathcal{N}_u|$ the *degree* of u, $d_{min} = \min_{u \in U} d_u$ and $d_{max} = \max_{u \in U} d_u$.
2. Each user $u \in U$ computes his *total noise*
$$\Delta_u \triangleq \sum_{v \in \mathcal{N}_u} \delta_{u,v} \tag{2.2}$$
and gets his *noisy value*
$$\tilde{x}_u \triangleq x_u + \Delta_u. \tag{2.3}$$

[1]Subsection 2.1.3 will show that we can even omit the knowledge of $|U|$.

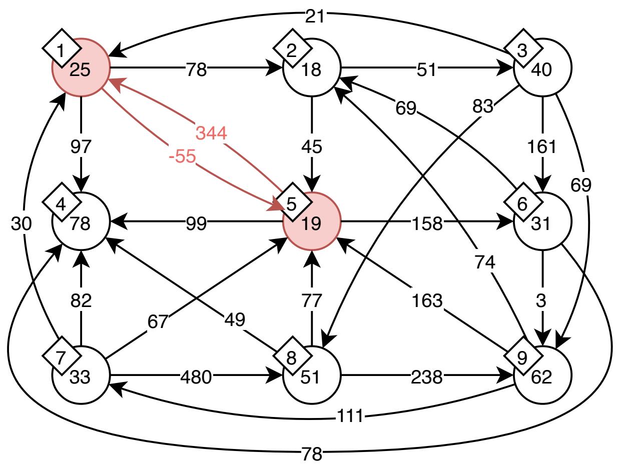


Figure 2.1. Example of a randomization graph. Unless $\delta_{u,v} \neq -\delta_{v,u}$, which is the case for edges $(1, 5)$ and $(5, 1)$, both $\delta_{u,v}$ and $\delta_{v,u}$ are represented with only one edge (u, v) to avoid redundancy.

The end product of randomization is $(\tilde{x}_u)_{u \in U}$. These are the values that users will reveal during the averaging process, thus reaching privacy. Remember that we allow this because of Lemma 1.

Lemma 1 (Correctness of averaging)**.** *If all users are honest, Algorithm 1 ensures that*

$$\bar{x} = \frac{1}{|U|} \sum_{u \in U} \tilde{x}_u. \tag{2.4}$$

Proof. Remark that noises skew the average calculation of

$$\begin{aligned} \bar{x} - \frac{1}{|U|} \sum_{u \in U} \tilde{x}_u &= \frac{1}{|U|} \sum_{u \in U} x_u - \frac{1}{|U|} \sum_{u \in U} (x_u + \Delta_u) \\ &= -\frac{1}{|U|} \sum_{u \in U} \Delta_u = -\frac{1}{|U|} \sum_{\{u,v\} \in E} \underbrace{(\delta_{u,v} + \delta_{v,u})}_{=0} = 0. \end{aligned}$$

□

Nevertheless, if some malicious users, *cheaters*, decide to exchange non-zero-sum noises, Lemma 1 would not hold anymore. That is why GOPA features a procedure to verify that property: the Zero-Knowledge Randomization Proof (ZKRdP).

2.1.2 Zero-Knowledge Randomization Proof

This procedure verifies equation (2.1), (2.2) and (2.3). However, we want such verifications to be zero-knowledge proofs (ZKPs), *i.e.* we want to evaluate these equations without revealing more than whether they hold or not. Indeed, GOPA cannot ask users to reveal their private values and all their noises because it would infringe privacy. That is when cryptography comes into play. After having described two cryptography primitives, homomorphic encryption (HE) and commitment schemes, you will be able to understand well the verification algorithm in itself.

Paillier homomorphic encryption

HE is a form of encryption which allows operations on ciphertexts whose decrypted result matches the result of the operations as if they had been performed on plaintext. Several homomorphic cryptosystems exist, such as RSA, ElGamal, Goldwasser-Micali, Benaloh and Paillier, each one providing their own homomorphisms. Paillier cryptosystem [41] has an interesting property to verify (2.2) and (2.3) and that is why it was chosen for GOPA among others.

Basically, in Paillier cryptosystem, each entity has:

- Private key $K^{priv} = \lambda = \text{lcm}(p-1, q-1)$ with $p \neq q$ two large primes.
- Public key $K^{pub} = (n, g)$ where $n = pq$ and g is an element of order $\alpha n, \forall \alpha \in \{1, \ldots, n\}$.
- Encryption operation of message $m \in \mathbb{Z}_n = \{0, \ldots, n-1\}$ with random nonce[2] $r \in \mathbb{Z}_n^*$:

$$\mathcal{E}(m; r) = g^m r^n \mod n^2. \tag{2.5}$$

  For purposes of simplicity, we will shorten the notation to $\mathcal{E}(m)$ when r is not to be a specific value or when its value is obvious given the context.
- Decryption operation of ciphertext $c \in \mathbb{Z}_{n^2}^*$:

$$\mathcal{E}^{-1}(c) = \text{L}(c^\lambda \bmod n^2) \div \text{L}(g^\lambda \bmod n^2) \bmod n \tag{2.6}$$

  where $\text{L}(y) = (y-1) \div n$.

Therefore, for two messages $\{m_1, m_2\}$ such that $m_1+m_2 < n$, we have the additive homomorphic property

$$\underbrace{\mathcal{E}(m_1 + m_2; r_1 r_2)}_{g^{m_1+m_2}(r_1 r_2)^n} = \underbrace{\mathcal{E}(m_1; r_1)}_{(g^{m_1} r_1^n)} \underbrace{\mathcal{E}(m_2; r_2)}_{(g^{m_2} r_2^n)} \mod n^2. \tag{2.7}$$

This property is particularly useful for ZKPs of (2.2) and (2.3) as you might feel, but it is not sufficient as we will see later on. Another needed component of the ZKRdP is commitment schemes.

Commitment schemes

A *commitment scheme* is a cryptographic primitive that allows an entity to prove that he committed to a value over time. Suppose that a user u has its own public encryption function $\mathcal{E}_u$. For any value y to which u commits, here is a typical commitment scheme that you should remind:

1. u shares ciphertext $c = \mathcal{E}_u(y)$ to concerned users, *i.e.* u commits to y.
2. Other things happen, possibly involving c, that justifies u's commitment.
3. u reveals plaintext y so that other users can compute $\mathcal{E}_u(y)$ and check that $c \overset{?}{=} \mathcal{E}_u(y)$. If so, u proved his commitment to y.

Hence, equipped with Paillier HE and commitment schemes, users perform Algorithm 2 to verify that every user conducted randomization as expected.

Algorithm 2 (Zero-Knowledge Randomization Proof)**.**

1. Before running Algorithm 1, each user $u \in U$ publishes[3] his public key $K_u^{pub} = (n_u, g_u)$ and ciphertext $\mathcal{E}_u(x_u)$ where $\mathcal{E}_u$ denotes Paillier encryption with K_u^{pub}.
2. All $u \in U$ perform Algorithm 1 with the supplement that each time two users u and v exchange noise $\delta_{u,v} = -\delta_{v,u}$, u publishes $\mathcal{E}_u(\delta_{u,v}; r_{u,v})$ and v publishes $\mathcal{E}_v(\delta_{v,u}; r_{v,u})$.
3. Each $u \in U$ publishes $\mathcal{E}_u(\tilde{x}_u)$ and $\mathcal{E}_u(\Delta_u)$ when he completed randomization. However, some users may cheat and not publish the (true) ciphertexts of x_u, Δ_u, *etc.* So, in the sequel, for any value y, we will denote $\mathcal{E}_u^P(y)$ the value published by u that supposedly equals $\mathcal{E}_u(y)$. Then, to reformulate, each $u \in U$ has now published $\mathcal{E}_u^P(x_u)$, $(\mathcal{E}_u^P(\delta_{u,v}; r_{u,v}))_{v \in \mathcal{N}_u}$, $\mathcal{E}_u^P(\Delta_u)$ and $\mathcal{E}_u^P(\tilde{x}_u)$.

[2]In cryptography, a nonce is an arbitrary number that can be used just once.

[3]shares to all other users by a mechanism chosen at implementation

4. $\forall (u,u') \in U^2$, u' can perform ZKPs of (2.2) and (2.3) by verifying that

$$\begin{cases} \mathcal{E}_u^P(\Delta_u) \overset{?}{=} \prod_{v \in \mathcal{N}_u} \mathcal{E}_u^P(\delta_{u,v}) \bmod n_u^2 & \vdash (2.2) \quad (2.8) \\ \mathcal{E}_u^P(\tilde{x}_u) \overset{?}{=} \mathcal{E}_u^P(x_u)\, \mathcal{E}_u^P(\Delta_u) \bmod n_u^2 & \vdash (2.3) \quad (2.9) \end{cases}$$

If not, u' proved that u cheated.

5. A partially ZKP of (2.1) is performed. First, $\forall u \in U$, a subset $V_u \subset \mathcal{N}_u$ is drawn at random such that $|V_u| = \lceil (1-\beta) d_u \rceil$ with $\beta \in (0,1)$. Second, $\forall v \in V_u$, u publishes plaintext $\delta_{u,v}$ and $r_{u,v}$, and v publishes $r_{v,u}$. Consequently, and because K_u^{pub} and K_v^{pub} are public, all other users can compute $\mathcal{E}_u(\delta_{u,v}; r_{u,v})$ and $\mathcal{E}_v(-\delta_{u,v}; r_{v,u})$ and verify that

$$\begin{cases} \mathcal{E}_u^P(\delta_{u,v}; r_{u,v}) \overset{?}{=} \mathcal{E}_u(\delta_{u,v}; r_{u,v}) & \quad (2.10) \\ \mathcal{E}_v^P(\delta_{v,u}; r_{v,u}) \overset{?}{=} \mathcal{E}_v(-\delta_{u,v}; r_{v,u}) & \vdash (2.1) \quad (2.11) \end{cases}$$

On the one hand, verifying (2.10) is verifying that u committed to the $\delta_{u,v}$ he added to x_u. If not, u is cheating. On the other hand, if (2.11) does not hold, it means that either u, v or both lied on the $\delta_{u,v}$ resp. $\delta_{v,u}$ they revealed or on the corresponding ciphertexts $\mathcal{E}_u^P(\delta_{u,v})$ resp. $\mathcal{E}_v^P(\delta_{v,u})$ or on the nonces $r_{u,v}$ resp. $r_{v,u}$ they published. In fact, we can regard both u and v as cheaters.[4]

We call the graph resulting from this procedure the *verified randomization graph*, $G_V = (U, E_V \subset E_R, \phi_V \colon U \to \{0,1\}, \psi_V \colon E_V \to \{0,1\})$. It is a labeled directed graph where:

- edge (u,v) exists if and only if u revealed plaintext $\delta_{u,v}$;
- edge-label $\psi_V(u,v) = \begin{cases} 1 & \text{if } (u,v) \text{ verifies (2.10) and (2.11),} \\ 0 & \text{otherwise;} \end{cases}$
- node-label $\phi_V(u) = \begin{cases} 1 & \text{if } u \text{ verifies (2.8) and (2.9), and } \forall (u,v) \in E_V, \psi_V(u,v) = 1, \\ 0 & \text{otherwise } (u \text{ cheated).} \end{cases}$.

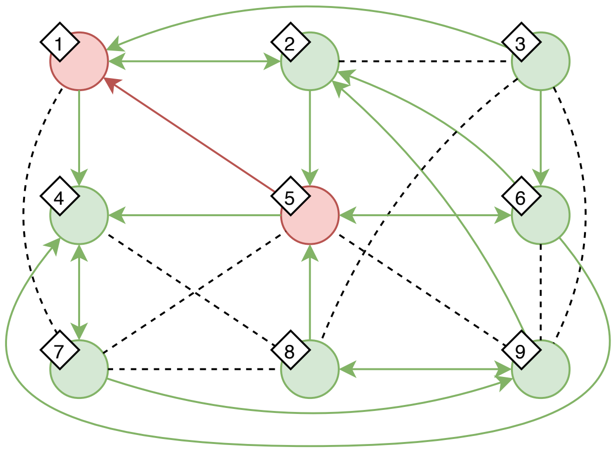


Figure 2.2. Example of a verified randomization graph deriving from Figure 2.1. Green represents label 1 and red represents label 0.

[4]We will justify that in subsection 4.2.3.

Why does Algorithm 2 work? First of all, we can see that the order in which ciphertexts and their respective plaintexts are published make each $u \in U$ commit to x_u before committing to $\delta_{u,v}$, before committing to Δ_u and $\tilde{x}_u$. This prevents u from changing a noise after having chosen or revealed another. Such change would be cheating as it would break the randomness of noises.[5] Moreover, the validity of (2.8) and (2.9) depends on the validity of the published ciphertexts. But, we cannot directly check that u committed to x_u and Δ_u by asking him to reveal x_u and Δ_u—it would break privacy. That is why users perform step 5. Step 5 encourages users to commit to true values, but it is partially ZKP because some of noises are revealed. However, as emphasized in Appendix B, Proposition 3, such revelations allow the detection of a cheater w.p. at least $1 - \beta^{2l}$ where l is the number of noises on which the cheater cheated. The smaller β, the greater the probability of guessing a user's private value, but the greater the one of identifying cheaters too.

Last but not least, we can assume with sufficient probability that each $u \in U$ who passes verification committed to a well-noised $\tilde{x}_u$ by publishing $\mathcal{E}_u^P(\tilde{x}_u)$. Then, when u shares $\tilde{x}_u$ at the beginning of the averaging phase[6], others will be able to check that $\mathcal{E}_u^P(\tilde{x}_u) \overset{?}{=} \mathcal{E}_u(\tilde{x}_u)$, *i.e.* to prove u trustworthy. This concludes the whole randomization phase and we can now progress towards the GOPA's averaging phase.

2.1.3 Gossip averaging phase

This phase provides a decentralized process that allows the averaging of noisy values $(\tilde{x}_u)_{u\in U}$ built during randomization. It is the pairwise gossip algorithm Push-Pull for average consensus. Gossip algorithms are sometimes also called *epidemic* as they spread information in a manner similar to the spread of a virus in a biological community. The principle of Push-Pull is that, at each time $t \in \mathbb{N}$, two users u and v exchange their current average estimates $a_u[t]$ resp. $a_v[t]$ and update them such that

$$a_u[t+1] = a_v[t+1] = \frac{a_u[t] + a_v[t]}{2}, \tag{2.12}$$

while $\forall u' \in U \setminus \{u, v\}, a_{u'}[t+1] = a_{u'}[t]$. Users collaboratively choose one another and repeat the averaging operation (possibly asynchronously and in parallel) until all have converged towards the same estimate. In which case, we say that *consensus* is reached.[7] If and only if there are no churns and no malicious users, this process converges towards $\bar{a} = \frac{1}{|U|}\sum_{u\in U} a_u[0]$, the average of the initial estimates. That is why each user u initializes $a_u[0] = \tilde{x}_u$.

Finally, remark that the graph used for averaging can differ from the randomization graph G_R by its edges. Indeed, all users who can communicate together have not necessarily exchanged noises ($E \neq E_R$). Then, to represent the averaging phase, we use the undirected *averaging graph* $G_A[t] = (U, E, u \mapsto a_u[t])$ whose node-labels represent users' estimates at time t.

This concludes the explanation of the two-phases functioning of GOPA (randomization $\rightarrow$ averaging) as it was designed at the beginning of my internship.

2.2 Study area

2.2.1 Problem statement

Within the objective of the implementation of GOPA, and considering the network disturbance models described below, the problematic of this thesis is: How to ensure privacy and correctness of GOPA's average calculation despite churns and malicious behaviors?

[5]Subsection 4.2.2 explains why such manipulation can reasonably be seen as malicious.

[6]along with the corresponding encryption nonce $r \in \mathbb{Z}_{n_u}^*$

[7]To go further, the exact convergence speed of the algorithm is figured in Appendix B, Proposition 1.

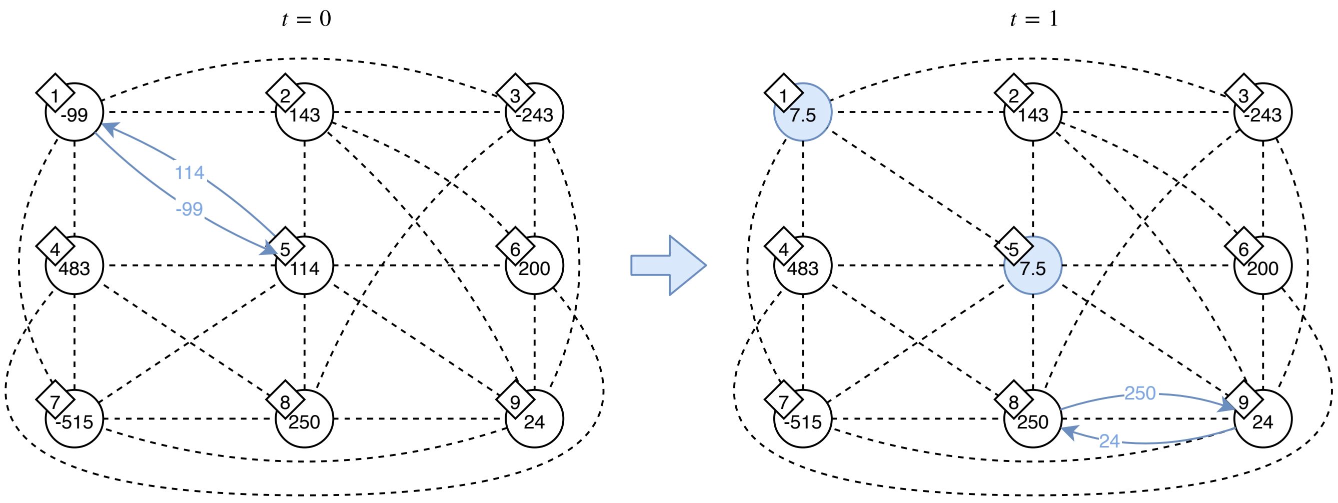


Figure 2.3. Example of the evolution of an averaging graph from $t = 0$ to 1, where users 1 and 5 exchange estimates.

2.2.2 Disturbance models

Churns

In our case, we can say that a churn is the modification of user set U over time. Through next chapters, we will consider churns that may happen at any time and for any reason. We will consider both willful disconnections, called *dropouts*, and unintended disconnections, called *faults*, as well as user late arrivals.

Malicious behaviors

In our study, we are interested with defense against malicious users and potentially external parts whose goals are:

- to guess private values;
- to compromise privacy-preserving subprotocols;
- to compromise the cheating detection mechanisms;
- to reduce the computation precision;
- to take the whole protocol down.

And whose possible actions are:

- to log in or out of a GOPA session;
- to send fake messages or values;
- to reveal private data;
- to collude, *i.e.* to synchronize their actions or to share private information;
- to block honest users thanks to denial of service (DoS) attacks.

On top of that, we will worry about honest-but-curious (HBC) users, *i.e.* users who do not deviate from the expected procedure, but who may take advantage of the available data to guess private values.

Chapter 3

State of the art

A few other works designed protocols similar to GOPA for privacy-preserving distributed computations. For instance, in Mo and Murray [37], initial private values are noised, but noises do not sum to zero. In Dimitriou and Awad [14] and Gupta, Katz, and Chopra [18], private data is hidden behind zero-sum noises, and [14] uses HE too in order to detect cheating. All these works are quite recent, like GOPA which was first published in 2016 [11].

Given the problem stated before, this chapter consists in a literature review on the topics of average consensus, fault-tolerant distributed protocols, fault and attack detection, and defense against malicious behaviors in decentralized networks.

3.1 Fault-tolerant protocols

To perform averaging, GOPA uses a pairwise randomized gossip algorithm, Push-Pull [21], whose τ-averaging time[1] is measured in [4, 10]. But other gossip algorithms were proposed for this task, such as Push-Sum [27], which can also compute the weighted average, and Distributed Random Grouping (DRG) [5], which needs much fewer transmissions than the previous ones. All three have asynchronous versions, however, they reach fault tolerance only at the cost of restart strategies. Lastly, there exists the Push-Adjust algorithm [38] but it has big vulnerabilities w.r.t. our purposes, namely a weak guarantee of convergence. Amongst fault-tolerant aggregation algorithms, we must also cite Flow Updating by Jesus, Baquero, and Almeida [24], which has a 'collect-all' and a pairwise versions for asynchronous aggregation. The algorithm is based upon the notion of flow in graph theory, and average consensus is reached by exchanging not only average estimates but also average flows. Hence, Flow Updating shows very appreciable properties including small mean error, fast convergence, as well as continuous self-adaptability to churns, to input values changes and to message losses, without requiring protocol restarts. In the experiments of [24], Flow Updating showed clearly better performances than Push-Sum, Push-Pull and DRG algorithms in terms of convergence speed and resilience.

Furthermore, fast message dissemination contributes to limiting the impact of churns on average consensus. Then, we can mention the work of Wang, Wang, and Wu [47], the Local Average and Information Exchange algorithm (LAIE), which is designed to be fast and to reduce the number of messages. Its convergence time is $\Omega\left(\frac{(n-1)\log n}{\Delta}\right)$ in random graphs, where n is the order of the graph and Δ its maximum degree. In simulations, the authors show that LAIE is faster than random gossip [4], geographic gossip [13] and broadcast gossip algorithms [48]. Besides, for rumor delivery with deadlines, Georgiou, Gilbert, and Kowalski [17] proposed deterministic and randomized gossip algorithms that tolerate message losses and node failures. These algorithms might not be used for gossip averaging but for decentralized dissemination of messages for the GOPA's ZKRdP, for instance.

[1] A measure of convergence rate and speed.

On a lower layer of the network stack, the overlay and organization mechanisms of the network can be taken into account. For that, we can cite the Adaptive Fault-Tolerant (AFT) overlay [42], whose structure rests on combinations of perpendicular chains and rings on a torus. Its fault tolerance property relies on a systematic reorganization mechanism taking advantage of its homogeneous structure. An advantage is that all AFT's operations (logging on, logging out, unicast and broadcast sharing, and maintenance) are in $O(\sqrt{n})$ where n is the number of nodes of the network. This derives from the topology in which each node has exactly 4 neighbors. However, an inconvenience of this protocol may appear when users compulsively alternate between getting online and offline. If such churns happen at a frequency that locks up the network in its reorganization mode, then users cannot communicate anymore. In fact, this phenomenon may appear in other network reorganization process and is an essential point to oversee. Another drawback of AFT is that reorganizing can require a lot of message exchanges.

In contrast with previous approaches, the famous Newscast gossip protocol [22] features an inherently dynamic structure. Its messages management makes it robust to churns by essence [46] since leaving the network is equivalent to stopping communications. Although the topology of the network is constantly changing, Newscast has been successfully used for gossip averaging consensus in previous works [20, 28, 36]. Furthermore, the protocol forms many small-world subgraphs and guarantees strong connectivity [46], at least in friendly environments.

3.2 Defense against malicious adversaries

A lot of works discuss defense against malicious users in decentralized networks. Most of them focus on detecting malicious behaviors, identifying attackers and limiting or, if possible, annulling their influence. Meanwhile, preventing attacks by a specific protocol or network structure is of strong interest and must be kept in mind. As a starting point, Kumar and Dutta [29] wrote a very comprehensive survey on intrusion detection systems (IDSs) for mobile *ad hoc* networks. Their paper covers statistical-based, heuristic-based, rule-based, state-based, signature-based, reputation-based, routing-information-based, cross-layer-based and graph-based approaches over the three last decades.

Attack strategies in gossip averaging are numerous, but we can separate them in two categories [16, 26]: *data falsification* and *data injection.* The first one happens when users start gossiping with fake values, and the second is when malicious users bias their value at each gossip iteration in order to skew the average towards a specific number or to hinder convergence. All these are a subcategory of Byzantine attacks, *i.e.* when users send values they should not.

While data falsification is handled through the ZKRdP in GOPA, Kailkhura, Brahma, and Varshney [26] analyzed the effects of data falsification in weighted average consensus. In particular, they proposed a robust algorithm for detecting weight manipulations. Further, they presented an expectation-maximization (EM) learning algorithm to not only detect attackers, but also adjust weights so as to mitigate the corruption of the average without excluding Byzantines. As to complete this work, Gentz *et al.* [16] proposed a decentralized statistical method to detect data injections and localize their source user. For both tasks, they gave upper bounds for probabilities of false positives and false negatives. As well, Mousazadeh and Tork Ladani [39] analyzed the impact of data injections on the convergence of random gossip algorithms. More precisely, they studied the harmfulness of an attack according to the mean and variance of injected values. That is, at what moment the gossip algorithm starts diverging and what error of users' estimates can be expected at a given time step. Later on, in [38], they introduced the Pull-Adjust algorithm and proposed a technique to push back malicious users. In their method, each node maintains and steadily updates a list of friends only with whom they gossip. They define friends as neighbors who share similar enough values according to a given threshold. Whereas push algorithms intrinsically allow the hyperactivity of malicious users, enforcing their harmfulness, pull only algorithms prevent them. However, Pull-Adjust can only approximate the average and requires a parameter tuning. Silvestre *et al.* [44] also investigated data injections detection, but through a deterministic and a probabilistic approach. To do that, they introduced the (Stochastic) Set Values-Observers ((S)SVOs) for dynamic analysis of gossiping. These observers allow each node to estimate the probability of occurrence of the values he exchanges with his neighbors and decide whether a neighbor is malicious or not. Nonetheless, the more accurate the estimate, the more costly the computations.

Meanwhile, Toulouse *et al.* [45] studied Bayesian IDSs for weighted average consensus under data injection attacks. They compared two solutions for detecting attackers: an outlier detection method using adaptive thresholds and a fault detection observer-based technique similar to that of [44]. Experimental results showed that the computational overhead of the first method is clearly less than that of the second one. But in return, detection speed of the latter is much lower. To ensure accuracy of averaging when Byzantines are removed, they also proposed a new algorithm for weighted average consensus. More innovative, Zhao, He, and Chen [50] exposed the Mobile Resilient Consensus Algorithm (MRCA) framework to detect malicious nodes while averaging. MRCA mixes a new faster push-pull algorithm with two related detection techniques running on normal static nodes on the one hand, and on a mobile detector node on the other hand. Taking advantage of two-hops information, these enable the systematic detection of malicious nodes even if those have formed cliques. In comparison, cliques blinded a previous detector, SATS [19].

Finally, whereas Newscast is highly resilient to churns, malevolent users can easily corrupt it. A typical threat for Newscast is hub attacks, *i.e.* attacks where malicious users become hubs prior to perform a deadly disturbance of the network. To answer this concern, Jesi, Montresor, and Steen [23] proposed a secure peer sampling in which honest users perform purely exploratory hidden exchanges that prevent the formation of hubs. More recently, to counteract one weakness of this method, Muszyński, Varrette, and Bouvry [40] had convincing results with a simpler defense consisting in reducing the frequency of P2P exchanges. Yet, this gain is at the cost of slowing down message dissemination.

3.3 Summary

The gossip averaging algorithm used by GOPA is very traditional and simple. The use of more recent algorithms, particularly Flow Updating [24] and the one of MRCA [50], may be profitable in terms of fault tolerance. Still, it is not evident how it would behave in the presence of HBC and malicious users.

By way of comparison with the reviewed material, the approach on which GOPA relies for detecting cheaters is quite novel. It is more similar to blockchain techniques for certifying transactions. It avoids statistical computations or dynamics analysis, but cryptographic operations are not necessarily faster or easier to implement, and message protocols can add additive delay. Besides, some papers had some topology-based or reputation-based approaches [23, 40, 50] to avoid malicious behaviors. Those may give inspirations for the improvement of GOPA.

Chapter 4

Analysis and theoretical improvement of GOPA

This chapter treats the main topic of this thesis by providing theoretical material for the improvement of GOPA. We analyze how churns can impact privacy and computations, how malicious users may take advantage of the existing protocol and how we should improve GOPA to manage churns and resist malicious strategies. Also, we will specify some procedures left unclear in the latest version of GOPA. Section 4.1 will expose general reflections about GOPA. Section 4.2 and 4.3 will be about the randomization resp. the averaging phases. And section 4.4 will synthesize all these improvements around user-centric strategies for churn management.

4.1 General aspects

4.1.1 Towards a user-centric protocol

Consider the two following approaches to the design of GOPA. On the one hand, we can think about a *session-centric* protocol where each GOPA session features phases with strict time frames that users must comply with if they want to get the end average result. For example, suppose a GOPA session for averaging the price of real estates. In a public registry lays the entry point of the session (*e.g.* IP address and port number) and the fixed end time of its randomization. The idea is that users who want to join the session must connect and randomize in the allocated delay. Then, they must pass through the ZKRdP where all cheaters get removed and noises are changed as necessary and possible. After, users can take the averaging phase, but with no possible return to randomization. As well, no late joiner is accepted and if the community encounters situations that compromise privacy, either those happen or the whole session is stopped. As we can see, such a session-centric protocol is not very robust because it puts users (and chance) in total charge of their privacy and of the success of averaging.

On the other hand, we can think about GOPA as a *user-centric* protocol in which each user follows the phases of GOPA at his own pace, more or less independently of others. The idea is that some users may randomize or do verifications while others would be averaging, whereas in the session-centric setting, all users must follow the same phase at the same time. Moreover, if there are churns and cheaters, users must be allowed to alternate between randomization and averaging, or doing both at the same time, depending on privacy budget and practical constraints. Of course, they must always begin with randomization. But the next stage to take only depends on their needs and opportunites, knowing that the randomization phase serves privacy while the averaging phase enables averaging. Evidently, the user-centric setting constitutes a much more flexible and robust protocol than the session-centric one presented before.

4.1.2 The semi-decentralized setting

GOPA is a decentralized protocol but there are different types of decentralized networks. On the one hand, we have networks where users connect to one another by their own individual means, *e.g.* wireless sensor networks. These compound the most literal or hardware interpretation of decentralized networks and they usually form incomplete graphs. We will refer to such network as *fully decentralized* due to geographical constraints. On the other hand, we have networks whose nodes may be able to connect to any other one through a central entity, internet alike, but where the central entity is reduced to a mean of communication, *e.g.* a minimalist message server. We refer to this setting as *semi-decentralized* because the decentralized high-level protocol that we are interested with (here, GOPA) is grounded upon a centralized communication layer. See Figure 4.1 as an illustration. This is the setting that we will assume within our contributions. Notably, because publishing data is simpler in semi- than in fully decentralized networks, by leveraging a common publishing server.

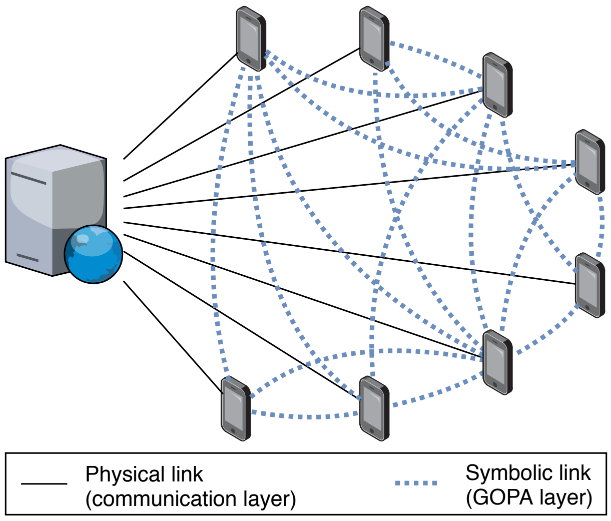


Figure 4.1. Illustration of the semi-decentralized setting. Cellphones represent users of GOPA who communicate through a central server.

By way of comparison, fully decentralized networks support no other but gossip communication. That means that even the publishing operation has to base upon rumor spreading schemes. Rumor spreading is a form of social communications in which an information, the *rumor*, is propagated through population by pairwise communications between spreaders and ignorants. Ignorants become themselves spreaders as soon as they receive the information, so that they transmit it to other ignorants and so on, until every member of the network is aware of the rumor. However, rumor spreading is very sensitive to data falsification, as many experienced through the telephone game[1]. Rumor spreading works only if the network is k-connected with k being big enough to ensure that $\forall(u, v) \in U^2$ there is at least one path p from u to v such that all users in p are honest. The less connected the network and the greater the proportion of malicious users who may falsify or not spread the rumor, the more likely an information is to be hidden from some honest users. This sums up the obstacles to data publication in fully decentralized networks and why we will take the semi-decentralized setting as a base assumption of our next discussions. Nevertheless, further investigations may generalize our solutions to the fully decentralized setting. For instance, refer to [17] for fast rumor spreading.

[1] *Telephone* is a kids game played around the world, in which one person whispers a message to another, who whispers it to another, *etc.* through a line of people until the last player announces the message to the whole group. A perverse effect of this game is that, because of intentional or unintentional intermediate falsifications, the announced message usually differs from the initial one.

4.1.3 Detecting churns

We mentioned churns many times before, but we never discussed how to detect them in GOPA. This is the topic of this subsection.

Detecting disconnections

First of all, if a honest user willfully disconnects, *drops out*, the most straightforward detection technique is that he notifies his neighbors of his quitting of the GOPA session. Then, remaining users can take corrective steps. This being said, let change topic to unintended disconnections, *faults*, and malicious users who would like to disconnect or play dead without honest ones knowing it.

A disconnected user is a user who does not answer anymore. Thus, for a user u to detect the failure of another v, u can decide that v disconnected if v did not participate in any exchange in due time, depending on the phase of the protocol. Such due time should be sufficiently long to avoid false positives. If u and v are expected to actively communicate (*e.g.* in the averaging phase), the absence of a user can be detected as the absence of messages from him. Nevertheless, in gossip averaging, there is a limit case if a user disconnects just before reaching consensus while the due time was not reached to detect his disconnections. Indeed, remaining users would claim consensus but they could not have taken desired corrective steps. To avoid this, one can set that each user systematically retests the presence of others when reaching the end of their current phase. Thus, avoiding complex delayed corrective steps.

If u and v are not expected to often communicate (*e.g.* in the verification phase), they can rely on presence tests by sending messages like “Are you there?”. If the tested user does not answer in the allocated time, the tester deduces that his peer is disconnected. Or, considering the published data of the ZKRdP, a user can confirm the presence of another as long as the latter published recently enough.

Finally, in the semi-decentralized setting, we can take advantage of the knowledge of the central server emulating the communication layer. Indeed, if a user physically loses connection, the server can directly push the updated user set to the still connected users. That means that the server himself should observe the presence of users, which is much easier from its viewpoint since it transmits user-to-user messages.

Detecting connections

Detecting node arrivals is trivial since newcomers just have to notify their presence to their peers. In the semi-decentralized setting, the central entity for communication is the entry point of the network. Then, each time a user joins, it should transmit the updated user set to all users.

4.2 Randomization phase

This section first discusses how churns can easily be managed in the randomization phase and how we can prevent malicious noise sampling. After, it focuses on the ZKRdP along with privacy by providing privacy-preserving methods against curious users and an algorithm to prevent malicious users from compromising cheating detection mechanisms.

4.2.1 Managing churns

Churns are changes in the user set over time. That is why we introduce notation:

- $G[t] = (U[t], E[t])$ describing the graph of users of the concerned GOPA instance at time $t \in \mathbb{N}$;

- $G_R[t] = (U_R[t] \subset U[t], E_R[t] \subset E[t], u \mapsto x_u, (u,v) \mapsto \delta_{u,v})$ describing the corresponding randomization graph over time;
- $\mathcal{N}_u[t] = \{v : (u,v) \in E_R[t]\}$ and $\tilde{x}_u[t] = x_u + \Delta_u[t]$ with $\Delta_u[t] = \sum_{v \in \mathcal{N}_u[t]} \delta_{u,v}$.

For privacy reasons, no user can start averaging if he has not exchanged noise before. Then, joining GOPA equals joining randomization. Also, quitting GOPA implies quitting all phases, including randomization. Thus, $\forall t \in \mathbb{N}, U_R[t] = U[t]$.

To reach correctness of averaging, the only requirement of the randomization phase is that $\forall t \in \mathbb{N}, \sum_{(u,v) \in E_R[t]} \delta_{u,v} = 0$. That is obtained if, when user u disconnects at t, each $v \in \mathcal{N}_u[t-1]$ removes $\delta_{v,u}$ from $\Delta_v[t]$. Moreover, it is not theoretically possible to exchange too much noise, so users can confidently exchange additional noise with one another (including late joiners) at any time.

Finally, to give consistency to the ZKRdP, when u removes $\delta_{u,v}$ because v disconnected, he must withdraw his ciphertext $\mathcal{E}_u^P(\delta_{u,v})$ from publication. Equally, when u adds noise $\delta_{u,v}$ exchanged with a late arrived neighbor v, he should publish ciphertext $\mathcal{E}_u^P(\delta_{u,v})$. In all cases, u has to update $\mathcal{E}_u^P(\Delta_u)$ and $\mathcal{E}_u^P(\tilde{x}_u)$ at the same time, so that $\mathcal{E}_u^P(\Delta_u) = \prod_{v \in \mathcal{N}_u} \mathcal{E}_u^P(\delta_{u,v}) \bmod n_u^2$ and $\mathcal{E}_u^P(\tilde{x}_u) = \mathcal{E}_u^P(x_u)\,\mathcal{E}_u^P(\Delta_u) \bmod n_u^2$ always hold.

Anticipating threats to privacy

Another important question of the randomization phase is about the sufficient number of noises that each user should exchange. Assume that the minimal number of noises that user u should have for strong privacy guarantees (*cf.* Appendix B, Theorem 1) is known. Let γ_1 be this number. Then, in the randomization phase, u should exchange $d_u > \gamma_1$ noises so that if some of his neighbors disconnect, he will still have enough noises to reach his privacy objectives. Hence, let us try to choose d_u^* a safe initial value of d_u.

Let us model that each $v \in \mathcal{N}_u$ may disconnect with *a priori* probability p_v computed by an arbitrary fault prediction system. Also, let r.v. L_u represent the number of noises that user u loses over time, *i.e.* the number of u's neighbors who disconnected. Clearly, L_u follows a Poisson binomial distribution $\mathcal{PB}((p_v)_{v \in \mathcal{N}_u})$ and then, given a small probability α that u loses too much neighbors over time, we can set d_u^* the solution of $\mathbb{P}(L_u \leq d_u^* - \gamma_1) \geq 1 - \alpha$.

4.2.2 Enforcing honesty through randomness

As part of robustness to malicious users, Algorithm 3 presents a method that allows users to exchange random noise. This randomness is crucial, otherwise a malevolent user u could choose a noise $\delta_{u,v}$ that makes x_u easier to guess when $\tilde{x}_u$ is shared at a later stage (*e.g.* $\delta_{u,v} = 0$). Such action becomes even more likely considering collusions of malicious neighbors.

Algorithm 3 (Random noise sampling)**.** For two users $u, v \in U$ willing to exchange noise:

1. u and v draw and exchange random reals $s_{u,v}$ resp. $s_{v,u}$, where $s_{u,v}, s_{v,u} \sim \mathcal{U}(0,1)$:
   (a) u sends ciphertext $c_{u,v} = \mathcal{E}_u(h(s_{u,v}); r_{u,v})$ to v with h an arbitrary hash function, *i.e.* u commits to $s_{u,v}$.
   (b) v sends $s_{v,u}$ to u.
   (c) u sends $s_{u,v}$ and $r_{u,v}$ to v. v can check that $c_{u,v} \overset{?}{=} \mathcal{E}_u(h(s_{u,v}); r_{u,v})$. If not, u is cheating and v can abort the exchange.
2. Each on their own, u and v compute $p = (s_{u,v} + s_{v,u}) - \lfloor s_{u,v} + s_{v,u} \rfloor$.

3. Then, they agree on $\delta_{u,v}$ being the $100p^{\text{th}}$ percentile of P_δ, where P_δ is an arbitrary distribution of noise which users agreed on before entering the randomization phase, *i.e.*

$$\delta_{u,v} = \operatorname{sgn}(s_{u,v} - s_{v,u})\, F_\delta^{-1}(p) \quad \text{where } \operatorname{sgn}(y) = \begin{cases} \dfrac{y}{|y|} & \text{if } y \neq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

and F_δ is the cumulative distribution function (CDF) of noise.

Because $p \sim \mathcal{U}(0,1)$, we have $\delta_{u,v} \sim P_\delta$. Hence, this solution allows the drawing of noises from any distribution whose quantile function is computable. Meanwhile, the sign computation of (4.1) ensures that $\delta_{u,v} = -\delta_{v,u}$. Also, note that v would have taken the risk that u had chosen $s_{u,v}$ with purpose if v had sent $s_{v,u}$ before u had sent $s_{u,v}$. This is counteracted by the commitment made at step 1a.

Gaussian randomization (4.1) allows the drawing of noises from any probability distribution. But in the case of Gaussian noises, *i.e.* $P_\delta \sim \mathcal{N}(\mu_\delta, \sigma_\delta^2)$, we can slightly adapt noise sampling for more straightforward computations. Indeed, two users can draw noise samples $s_{u,v}, s_{v,u} \sim \mathcal{N}\left(\frac{\mu_\delta}{2}, \frac{\sigma_\delta^2}{2}\right)$ and compute

$$\delta_{u,v} = \operatorname{sgn}(s_{u,v} - s_{v,u})\,(s_{u,v} + s_{v,u}). \tag{4.2}$$

Because the mean and the variance of the sum of i.i.d. Gaussians equal the sum of the means and the sum of the variances of these Gaussians, we have $\delta_{u,v} \sim \mathcal{N}(\mu_\delta, \sigma_\delta^2)$. Namely, we are interested in centered noise, *i.e.* $\mu_\delta = 0$, by reference to Appendix B, Theorem 1, about privacy.

4.2.3 Randomization proof and two-party commitment

After having drawn noise $\delta_{u,v}$, u must publish $\mathcal{E}_u^P(\delta_{u,v})$ for the ZKRdP. But, Algorithm 2, step 5, states that if a user u reveals a noise $\delta_{u,v}$ to which his neighbor v did not commit, *i.e.* $\mathcal{E}_v^P(\delta_{v,u}) \neq \mathcal{E}_v(-\delta_{u,v})$, both are regarded as cheaters. In other words, step 5 assumes that both u and v commited to both $\mathcal{E}_v^{-1}(\mathcal{E}_v^P(\delta_{v,u}))$ and $\mathcal{E}_u^{-1}(\mathcal{E}_u^P(\delta_{u,v}))$. This is so because otherwise we cannot differentiate whom between u or v is cheating.

In practice, this two-party commitment should be guaranteed by the fact that u publishes $\mathcal{E}_u(\Delta_u)$ only if every $v \in \mathcal{N}_u$ has published $\mathcal{E}_v^P(\delta_{v,u}) = \mathcal{E}_u(-\delta_{u,v})$. If not, u just has to nullify the noise he exchanged with v (he updates Δ_u and withdraws $\mathcal{E}_u^P(\delta_{u,v})$) and continue randomization as necessary.

4.2.4 Privacy *vs.* completeness of the randomization proof

In the ZKRdP (Algorithm 2, step 5), it is required that each $u \in U$ reveals plaintext noise $\delta_{u,v}, \forall v \in V_u \subset \mathcal{N}_u$ such that $|V_u| = \lceil (1-\beta) d_u \rceil$ where $\beta \in (0,1)$. But, from equation $\tilde{x}_u \triangleq x_u + \sum_{v \in \mathcal{N}_u} \delta_{u,v}$, it is obvious that publishing noises poses a threat to privacy. For instance, if u published $d_u - 1$ of his noises, let say $\{\delta_{u,v}\}_{v \in \mathcal{N}_u \setminus \{v_1\}}$, then his HBC neighbor v_1 could guess $x_u = \tilde{x}_u - \sum_{v \in \mathcal{N}_u \setminus \{v_1\}} \delta_{u,v} + \delta_{u,v_1}$. In addition, suppose that it exists $v_2 \in \mathcal{N}_u \setminus \{v_1\}$ who colludes with v_1 in curious purposes. Then, if u had published the adequate $d_u - 2$ noises, $\{\delta_{u,v}\}_{v \in \mathcal{N}_u \setminus \{v_1, v_2\}}$, v_1 and v_2 could guess x_u by joining their information.

Hence, an approach for guaranteeing privacy is to choose a value of β that ensures a high probability $1-\alpha$ of hiding enough noises γ_0 per user. That is, if B_u^δ denotes the number of user u's noises published during the verification phase (either by u or his neighbors), we are looking for the value of β that verifies

$$\forall u \in U, \mathbb{P}(B_u^\delta \leq d_u - \gamma_0) \geq 1 - \alpha. \tag{4.3}$$

Within this objective, γ_0 is the number of unpublished noises that a user needs to know in order to guess the private value of one of his neighbor. γ_0 is also the size of the smallest curious collusion in $\mathcal{N}_u$ who may guess x_u from $\tilde{x}_u$. Hence, it can be named the *direct privacy level.* Considering the HBC setting, we must have $\gamma_0 \geq 2$ in all cases.

Algorithm 2 makes each user u independently disclose $\lceil(1-\beta)d_u\rceil$ noises (one noise could be published twice, firstly as $\delta_{u,v}$ by u, secondly as $\delta_{v,u}$ by v). Thus, the number of u's disclosed noises is not $\lceil(1-\beta)\rceil$. Instead, we estimate this number in Proposition 1 and Remark 1.

Proposition 1. *Let r.v. B_u^δ the number of user u's noises disclosed following Algorithm 2. If and only if noises are disclosed at random, Algorithm 2 ensures that B_u^δ follows a Poisson binomial distribution $\mathcal{PB}((p_{u,v})_{v\in\mathcal{N}_u})$ where success probability $p_{u,v} = 1 - \frac{\lfloor\beta d_u\rfloor}{d_u}\frac{\lfloor\beta d_v\rfloor}{d_v}$.*

Proof. Let event $B_{u,v}^\delta$ be "u reveals $\delta_{u,v}$ or v reveals $\delta_{v,u}$". According to Algorithm 2, each user u independently reveals $\lceil(1-\beta)d_u\rceil$ of his noises chosen at random. In other words, u independently keeps secret only $\lfloor\beta d_u\rfloor$ noises out of d_u. Therefore, $B_{u,v}^\delta$ models a Bernouilli trial w.p. $p_{u,v} = 1 - \frac{\lfloor\beta d_u\rfloor}{d_u}\frac{\lfloor\beta d_v\rfloor}{d_v}$. Of course, B_u^δ is the sum of all successful $B_{u,v}^\delta$, and hence $B_u^\delta \sim \mathcal{PB}((p_{u,v})_{v\in\mathcal{N}_u})$. □

Remark 1. Let r.v. B_u^δ the number of user u's noises disclosed following Algorithm 2. If and only if G is d-regular and noises are disclosed at random, Algorithm 2 ensures that B_u^δ follows a binomial distribution $\mathcal{B}\left(d, 1-\frac{\lfloor\beta d\rfloor^2}{d^2}\right)$.

Proof. Following the proof of Proposition 1, notice that the binomial distribution is a special case of the Poisson binomial distribution when all success probabilities are the same. This is the case when G is d-regular $\iff \forall u \in U, d_u = d$. □

As an illustration of Proposition 1 and Remark 1, Figure 4.2 shows the influence of β on B_u^δ.

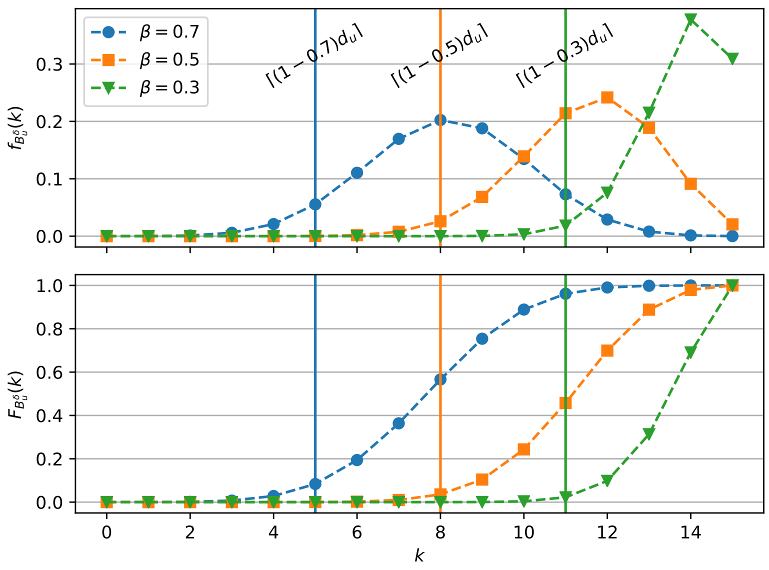


Figure 4.2. Probability mass function (PMF) and CDF of B_u^δ for different settings of β where $d_u = 12$ and $\forall v \in \mathcal{N}_u, d_v \in [5, 60]$. Here, for $\gamma_0 = 5$ and $\alpha = 0.2$, and according to (4.3), one can choose $\beta = 0.7$, which will result in at least $1 - 0.7^2 = 0.51$ of probability of detecting each cheater. We can see on the upper subplot that the number of noises that user u individually discloses, $\lceil(1-\beta)d_u\rceil$, is not that close to the mean of the distribution, $\mathbb{E}[B_u^\delta] \gtrapprox (1-\beta^2)d_u$. This motivated the analysis above.

As a result of Proposition 1 and Remark 1, one can set up β according to (4.3) given privacy budget γ_0 and threshold α for any *(a priori)* known topology. Or reciprocally, given

β which relates to the probability of detecting cheaters (*cf.* Appendix B, Proposition 3), one can compute probability α of breaking direct privacy level γ_0. In sum, we demonstrated a probabilistic guarantee for privacy against curious neighbors collusions of size γ_0.

4.2.5 Ensuring detection of cheaters

As we have seen before, some properties on privacy and probability of cheating detection hold on the ZKRdP only if noises are disclosed in plaintext at random. Then, this subsection is dedicated to the random sampling of $(V_u)_{u \in U}$. As first introduced in Algorithm 2, step 5, V_u denotes the subset of u's neighbors such that $\forall v \in V_u$, u reveals plaintext $\delta_{u,v}$.

First of all, notice that we cannot let a user u choose his own V_u, because if u were malicious, he could choose one that avoid him from getting detected. As illustrated in Figure 4.3, this phenomenon is even amplified by collusions. But also, assuming the semi-decentralized setting, we cannot let a third part alone choose $(V_u)_{u \in U}$, because no users could claim that they were selected without malicious purposes. These risks justify the need to choose $(V_u)_{u \in U}$ fully at random.

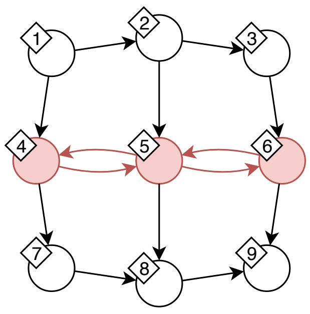


Figure 4.3. Example of a randomization graph where users 4, 5 and 6 are malicious and agreed to cheat on $\delta_{5,4}$ and $\delta_{5,6}$. If each user u had the ability to choose his own V_u, then, for $\beta = 0.25$, malicious users could obviously choose $V_4 = \{1\}$, $V_5 = \{2\}$ and $V_6 = \{3\}$, so that their fraud would not be detected.

In fact, there is always a risk that others cheat. Consequently, if a user wants to ensure the randomness of $(V_u)_{u \in U}$, he must inject his own randomness into the process. We then propose the protocol defined through Algorithm 4. Its idea is that users collectively draw a random number N which maps to a unique set among subsets of k neighbors among d. For a user of arbitrary degree d, such a mapping is the inverse of

$$\kappa_{d,k} \colon \bigtimes_{j=1}^{k} [0, d-j] \to \left[0, \binom{d}{k} - 1\right]$$
$$(i_1, \ldots, i_k) \mapsto ((i_1 d + i_2)(d-1) + i_3)(d-2) + \ldots)(d+1-k) + i_k$$

where $i_1, \ldots, i_k$ relates to the indexes of the k chosen neighbors (line 6). Note that N, and then $(V_u)_{u \in U}$, is drawn at random (line 3) if there is at least one honest user. Therefore, Algorithm 4 leverages Proposition 1 and Proposition 3 of Appendix B, which states that a user who cheated l times during the randomization process is detected at least w.p. $1 - \beta^{2l}$.

4.2.6 When restarting randomization becomes necessary

Subsection 4.2.1 explains that churns should be managed by nullifying the noises of disconnected users, adding those of neighboring newcomers and updating the corresponding published ciphertexts. Equally, if any $w \in U$ disconnected, each $u \in \mathcal{N}_w$ should withdraw the publications of plaintext noise $\delta_{u,w}$ and may reveal noise $\delta_{u,v}$ exchanged with new neighbors v w.r.t. privacy budget β. When all data required for verification are up to date, verifiers can check the validity of u's randomization. Still, if such procedure is not sophisticated enough, malicious users could break privacy. This subsection focuses on this aspect.

Let user u has h_u honest neighbors $\mathcal{N}_u^H = \{v_1, \ldots, v_{h_u}\}$ and malicious neighbors $\mathcal{N}_u^M = \mathcal{N}_u \setminus \mathcal{N}_u^H = \{v_{h_u+1}, \ldots, v_{d_u}\}$.

Algorithm 4 Randomized selection of $(V_u)_{u \in U}$

```
Input: $\beta \in (0,1)$ the proportion of noises that each user individually keeps secret
 1: for all $u \in U$ do
 2:     $V_u \leftarrow \emptyset$
 3: $N \leftarrow$ collectivelyRandomBigInteger($U$)
 4: for all $u \in U$ do
 5:     $k \leftarrow \lceil (1-\beta) d_u \rceil$
 6:     $(i_1, \ldots, i_k) \leftarrow \kappa^{-1}(d_u, k, N \bmod \binom{d_u}{k})$
 7:     for $j \leftarrow 1, \ldots, k$ do
 8:         Add the $(i_j + 1)^{\text{th}}$ neighbor of $\mathcal{N}_u \setminus V_u$ to $V_u$.
 9: return $(V_u)_{u \in U}$
10: function collectivelyRandomBigInteger($U$: Set<User>): Integer
11:     $C \leftarrow \emptyset$ /* A temporary cheater list */
12:     for all $u \in U$ do
13:         $u$ draws random number $N_u \in \mathbb{Z}_{n_u}$ and publishes $\mathcal{E}_u^P(N_u; r_u)$.
14:     for all $u \in U$ do
15:         $u$ reveals $N_u$ and $r_u$.
16:         Others check that $\mathcal{E}_u^P(N_u; r_u) \overset{?}{=} \mathcal{E}_u(N_u; r_u)$. If not, add $u$ to $C$.
17:     return $\sum_{u \in U \setminus C} N_u$
18: function $\kappa^{-1}$($d$, $k$, $N$: Integer): List<Integer>
19:     for $j \leftarrow k, \ldots, 1$ do
20:         $i_j \leftarrow N \bmod (d + 1 - j)$
21:         $N \leftarrow N \div (d + 1 - j)$
22:     return $(i_1, \ldots, i_k)$
```

Case 1 ($\mathcal{N}_u^M$ wants to guess x_u)**.** Suppose that malicious neighbors are colluding and that u's first average exchange will be with one of them. That is, malicious neighbors know $\{\delta_{u,v_j}\}_{j=h_u+1}^{d_u}$ through randomization and they will know $\tilde{x}_u$ when starting the averaging phase. To guess x_u, they need to know $\{\delta_{u,v_i}\}_{i=1}^{h_u}$. In the ZKRdP, the number k of honest neighbors' noises δ_{u,v_i} published by u in plaintext follows a hypergeometric distribution $\mathcal{H}(d_u, h_u, \lceil (1-\beta) d_u \rceil)$ modelling the drawing of $\lceil (1-\beta) d_u \rceil$ noises among d_u of which h_u are honest neighbors' noises. To boost the probability that k be high, each malicious user $v_j \in \mathcal{N}_u^M$ could wait and not publish his noise $\delta_{v_j,u}$ until enough u's honest neighbors' noises $\delta_{v_i,u}$, have been revealed. If u published some malicious neighbors' noises δ_{u,v_j} that prevented the disclosure of enough honest neighbors' noises δ_{u,v_i}, a malicious neighbor $v_j \in \mathcal{N}_u^M$ could disconnect so that u would unpublish δ_{u,v_j} and possibly publish a noise exchanged with an honest neighbor. Otherwise, additional colluding users could exchange noises with u to enable the publishing of more noises, so that u may publish honest neighbors' noises also. Such procedure may be repeated until enough u's honest neighbors' noises δ_{u,v_i} are revealed to guess x_u.

Fortunately, we can counteract this problem case. First, note that the privacy of x_u is at sake only when u shares $\tilde{x}_u$. Then, if u estimates that sharing $\tilde{x}_u$ is too risky for privacy, he should be able to cancel all his previous noise exchanges and retake randomization as if he never passed it.

Moreover, the number of plaintext noises needed by malicious users to guess a private value depends on the capability of malicious neighbors to infer x_u. Deterministically speaking, this amount is of h_u noises. Probabilistically speaking, suppose that k u's honest neighbors' noises $\{\delta_{u,v_i}\}_{i=1}^{k}$ have been disclosed (either by u or $v_i \in \mathcal{N}_u^H$). Then, knowing the probability distribution of unrevealed honest neighbors' noises $\{\delta_{u,v_j}\}_{j=k+1}^{h_u}$, malicious neighbors might infer

$x_u \in \mathcal{D}_x$ w.p.

$$\mathbb{P}(x_u = y \mid \tilde{x}_u, \delta_{u,v_{k+1}}, \dots, \delta_{u,v_{d_u}}) = \mathbb{P}\left(\sum_{j=1}^{k} \delta_{u,v_j} = \tilde{x}_u - \sum_{i=k}^{d_u} \delta_{u,v_i} - y \right). \tag{4.4}$$

However, this should be assumed not helpful as the variance of the noise distribution is to be great enough to make $\mathbb{P}(x_u = y \mid \tilde{x}_u, \delta_{u,v_{k+1}}, \dots, \delta_{u,v_{d_u}})$ almost constant over $\mathcal{D}_x$.

4.2.7 How malicious users may break privacy

Let us analyze another problem case featuring the same notation as above.

Case 2 ($\mathcal{N}_u^M$ wants to make x_u public)**.** To make x_u public, u's malicious neighbors can wait that, by chance, all u's honest neighbors' noises $\{\delta_{u,v}\}_{i=1}^{h_u}$ are published. And then, all malicious user $w \in \mathcal{N}_u^M$ could publish $\delta_{w,u}$. At this moment, if only if no $w \in \mathcal{N}_u^M$ cheated on $\delta_{w,u}$, all $u' \in U$ potentially knows Δ_u, and who knows $\tilde{x}_u$ can guess x_u.

This second problem case is harder to solve because malicious users could reveal $\{\delta_{u,v_j}\}_{j=h_u+1}^{d_u}$ after that u had revealed $\tilde{x}_u$. In fact, not only do all depend on users' curiosity, but also, in the semi-decentralized setting, the feasibility of such attack depends on the honesty and the power of the publication server. For instance, we may asked the server to filter the users' messages in order to avoid the publishing of illegitimate values. We stop our analysis of Case 2 here and let the development of solutions to further studies.

Summary

To summarize our contributions to the randomization phase, this section answered:

- how to manage churns for correctness of averaging;
- how many noise exchanges each user should perform to preserve privacy;
- how to sample noises to enable privacy;
- how to set β w.r.t. to privacy budget and malicious neighbors collusions;
- how to choose $(V_u)_{u \in U}$ at random so that malicious users cannot compromise cheating detection mechanisms;
- how to protect privacy against malicious users exploiting churns;
- and how malicious users may break privacy.

4.3 Averaging phase

We now change topic to present our contributions to the GOPA's averaging phase. This section develops solutions for the improvement of the speed and the correctness of the averaging process.

4.3.1 Choosing between public or gossip averaging

GOPA originally performs gossip averaging as presented in subsection 2.1.3. But, considering the users' ability to publish data (*cf.* the ZKRdP), we could use another strategy. This strategy is that all $u \in U$ publishes $\tilde{x}_u$ in plaintext so that every user can collect $(\tilde{x}_u)_{u \in U}$ and then compute $\bar{x} = \frac{1}{|U|} \sum_{u \in U} \tilde{x}_u$ in only one operation. We refer to this approach as *public averaging.*

Multiple criteria must be taken into account when deciding which approach to use. For GOPA, the best solution is the one that maximizes privacy, convergence speed and correctness of averaging. If all users are honest, public averaging is clearly faster than gossip averaging.

Indeed, public averaging converges with no error in only one iteration, whereas gossip averaging needs several and its correctness depends on the time at which users stop gossiping.[2] In the case some users are malicious, correctness of averaging depends on mechanisms for cheating detection, for cheater removals and for restoring the network to a correct state (zero-sum noises). For public averaging, restoring correctness costs no more than correcting the data published and verified through the ZKRdP. Meanwhile, gossip averaging involves additional costs due to the verification of many average exchanges.

Regarding privacy of public averaging, it mainly depends on the magnitude of the noise added during the randomization phase. In contrast, gossip averaging increases privacy through the multiple computations separating average estimate $a_u[t]$ from $a_u[0] = \tilde{x}_u$. We can reformulate by arguing that each gossip iteration is like adding even more secret noises to private data, hardening the inference x_u. On the one hand, this is assumed not very helpful as the randomization phase should ensure that all possible values of x_u have a similar posterior probability. Indeed, $\tilde{x}_u$ is to be known by at least one u's neighbor when performing gossip averaging as well, but this neighbor should not be able to guess x_u from $\tilde{x}_u$. On the other hand, one can argue that time and privacy could be relocated from the randomization phase (by doing fewer noise exchanges) to the average one. By intuition, such strategy would facilitate defense against the privacy breach identified in subsection 4.2.7. Furthermore, fully decentralized networks support no other but gossip methods for averaging. On balance, discarding gossip averaging is not self-evident. That is why the next subsections develops solutions to manage churns and to detect frauds in gossip average consensus.

4.3.2 Robustness to churns

Beforehand, recall notation $G[t] = (U[t], E[t])$ describing the graph of users at a global level of GOPA at time $t \in \mathbb{N}$. This graph depends on external factors (*e.g.* technical failures) as well as internal factors (*e.g.* cheaters removal). Our objective is to make the averaging calculation up to date with user set $U[t]$ at any time $t \in \mathbb{N}$. In other words, each time a churn happens at t, *i.e.* $U[t] \neq U[t-1]$, the community should start computing the *rectified average* $\bar{x}[t] = \frac{1}{|U[t]|}\sum_{u \in U[t]} x_u$ in a churn-tolerant fashion. Figure 4.4 exemplifies the typical situation we want to avoid.

Now, let me introduce $G_A[t] = (U_A[t] \subset U[t], E_A[t] \subset E[t], u \mapsto a_u[t])$ the node-labeled graph representing the averaging process over time. $U_A[t]$ may differ from $U[t]$ as late arrived users of GOPA cannot directly join averaging. Instead, they must first pass randomization as explained in subsection 4.2.1. Then, to give consistency to the next discussion, begin with a clear definition of churn tolerance for generic gossip average consensus.

Definition 1 (Churn tolerance)**.** Suppose that a gossip averaging algorithm $\mathcal{A}$ is running on G_A and a churn suddenly happens at time t. Then, $\mathcal{A}$ is *churn-tolerant* if and only if:

1. $\mathcal{A}$ integrates new users into averaging, *i.e.* $\exists t' > t, U[t] \subset U_A[t']$ where $|t' - t|$ is reasonably small. Assuming that this is true, let $t_{u,0}$ be the time at which user u joins U_A, thus initializing his average estimate to $a_u[t_{u,0}]$. Note that $\forall u \in U_A[t] \setminus U_A[t-1], t_{u,0} = t$.
2. $\mathcal{A}$ now averages $\bar{a}[t] = \frac{1}{|U_A[t]|}\sum_{u \in U_A[t]} a_u[t_{u,0}]$,
3. but without requiring a restart, *i.e.* $\forall u \in U_A[t-1], \forall t' > t, u$ never do the update $a_u[t'] = a_u[t_{u,0}]$ (except by chance as part of an average exchange).

Define as well:

- *Fault tolerance*, a particular case of churn tolerance where $U[t] \subsetneq U_A[t-1]$.
- *Late arrival tolerance*, a particular case of churn tolerance where $U_A[t-1] \subsetneq U[t]$.

[2] Refer to the algorithm's τ-averaging time computed in Appendix B, subsection 4.2.

Example 1. The versions of Push-Pull presented in [24, section 4.2] are not because they use restart strategies. In contrast, Flow Updating [24] is churn-tolerant. As the data are regularly updated and recomputed, churns are transparent; the updates are almost as dynamic as the topology.

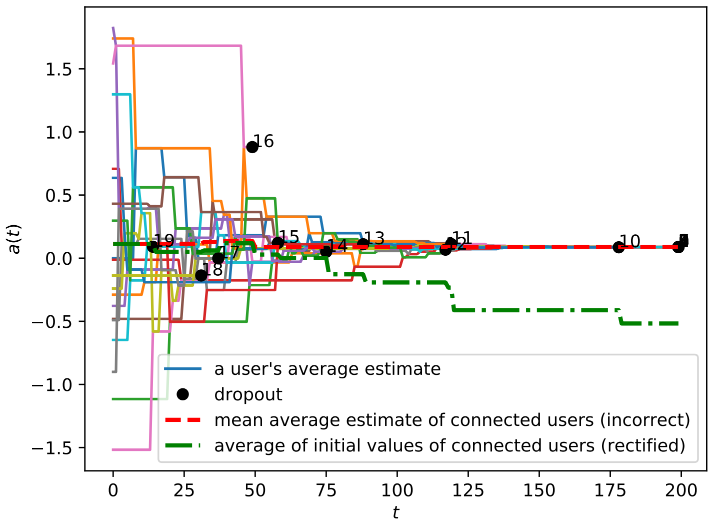


Figure 4.4. Example of the impact of unmanaged dropouts on the convergence of gossip averaging. The red dashed line represents the incorrect value that users converge towards when they ignore dropouts. The dash-dotted green line represents the average that connected users should converge towards.

Following Definition 1, we can state generic Lemmas 2 and 3.

Lemma 2. *Let $\mathcal{A}$ be a gossip averaging algorithm verifying condition 1 of Definition 1. Suppose that node w joined $\mathcal{A}$ at $t_{w,0}$, exchanged averages at least once, and quits $\mathcal{A}$ at t, i.e. $w \notin U_A[t]$. Assume also that no other churn happens later, i.e. $U_A[t] = U_A[t+1]$. Then, $\mathcal{A}$ is fault-tolerant if users subtract $a_w[t_{w,0}]$ from their total estimates, i.e.*

$$\sum_{u \in U[t+1]} a_u[t+1] = \sum_{u \in U[t-1]} a_u[t] - a_w[t_{w,0}]. \tag{4.5}$$

If w never exchanged averages, it is like if he never existed so fault-tolerance is reached even if other users ignore his disconnection.

Note that for pairwise gossip, only v_1, the user with whom w first averaged, knows $a_w[t_{u,0}]$. Hence, $\mathcal{A}$ is made churn-tolerant if v_1 is still active at that point in time and if he updates his estimate such that $a_{v_1}[t+1] = a_{v_1}[t] - a_w[t_{w,0}]$. Otherwise, $a_w[t_{w,0}]$ should be revealed to other users so that they can remove it.

Proof. By design, a base requirement of any gossip averaging algorithm $\mathcal{A}$ is that $\forall t'' \in \mathbb{N}, \sum_{u \in U_A[t'']} a_u[t''] = \sum_{u \in U_A[t'']} a_u[t_{u,0}]$ because it averages $(a_u[t_{u,0}])_{u \in U_A[t'']}$. Therefore, if w quits averaging at t, and users subtract $a_w[t_{w,0}]$ from their estimates (Definition 1, item 3), we have

$$\sum_{u \in U_A[t+1]} a_u[t+1] = \sum_{u \in U_A[t-1]} a_u[t] - a_w[t_{w,0}] = \sum_{u \in U_A[t]} a_u[t_{u,0}].$$

That is, $\mathcal{A}$ now converges towards $\bar{a}[t] = \frac{1}{|U_A[t]|} \sum_{u \in U_A[t]} a_u[t_{u,0}]$ (Definition 1, item 2). □

Lemma 3. *Any gossip averaging algorithm $\mathcal{A}$ is late-arrival-tolerant if and only if it verifies item 1 of Definition 1.*

Proof. The fact that w joins an ongoing averaging process is the same as if he was connected since the beginning but had not exchanged average estimates yet. Therefore, convergence towards the rectified average starts as soon as w gossips with others. □

To sum up, a gossip algorithm $\mathcal{A}$ verifying Lemmas 2 and 3 is churn-tolerant. Next, recall that $G_R[t] = (U[t], E_R[t], u \mapsto x_u, (u,v) \mapsto \delta_{u,v})$ is the randomization graph at time t. This graph evolves apart from G_A. Consequently, we can claim Theorem 1 for churn tolerance in GOPA.

Theorem 1 (Churn-tolerant GOPA)**.** *Suppose that user w quits or late joins an ongoing GOPA's averaging phase at time t. If w joins, assume that he previously achieved randomization with other users in parallel, so that $\exists \mathcal{N}_w[t-1] = \{v : (w,v) \in E_R[t-1]\}$. Hence, if all users are honest and GOPA's gossip averaging algorithm $\mathcal{A}$ is churn-tolerant, it converges towards the rectified average $\bar{x}[t] = \frac{1}{|U[t]|}\sum_{u\in U[t]} x_u$ only if it has run its mechanisms for fault tolerance, let say from t to t', and if each $v \in \mathcal{N}_w[t-1]$ has updated his estimate such that*

$$a_v[t'+1] = a_v[t'] + c_w\delta_{v,w} \quad \textit{where } c_w = \begin{cases} +1 & \textit{if } w \textit{ joined,} \\ -1 & \textit{if } w \textit{ disconnected.} \end{cases} \tag{4.6}$$

Proof. The intuition is that users update their average estimates in accordance with the current noise exchanges. Let t be the time at which w disconnects or joins averaging, *i.e.* $U_A[t] = U[t]$. Recall notation $\tilde{x}_u[t''] = x_u + \Delta_u[t'']$ and $\Delta_u[t''] = \sum_{v\in\mathcal{N}_u[t'']}\delta_{u,v}$ for u's noisy value and noises at time t''. If all users are honest and assuming that no churn happened before, $\forall t'' < t \in \mathbb{N}, \sum_{u\in U[t'']} a_u[t_{u,0}] = \sum_{u\in U[t'']}\tilde{x}_u[t''] = \sum_{u\in U[t'']} x_u$. Next, at $t' > t$, the churn tolerant mechanisms of $\mathcal{A}$ sets that $\sum_{u\in U_A[t']} a_u[t'] = \sum_{u\in U_A[t-1]} a_u[t] + c_w a_w[t_{w,0}]$ where $a_w[t_{w,0}] = \tilde{x}_w[t-1]$ because users want to average $(\tilde{x}_u[t])_{u\in U_A[t]}$. Therefore, by applying (4.6), we have

$$\begin{aligned}\sum_{u\in U_A[t'+1]} a_u[t'+1] &= \sum_{u\in U_A[t'+1]} a_u[t'] + \sum_{v\in\mathcal{N}_w[t-1]} c_w \underbrace{\delta_{v,w}}_{=-\delta_{w,v}} \\ &= \left(\sum_{u\in U_A[t-1]} a_u[t] + c_w\tilde{x}_w[t-1]\right) + c_w\Delta_w[t-1] \\ &= \underbrace{\sum_{u\in U_A[t-1]} a_u[t_{u,0}]}_{=\sum_{u\in U[t-1]} x_u} + c_w x_w = \sum_{u\in U_A[t'+1]} x_u.\end{aligned}$$ □

Notice that Theorem 1 extends to churns concerning multiple users at the same time and multiple churns over time, thus enabling a dynamic solution for correctness of averaging. Theorem 1 is formulated for gossip averaging, but it is obvious that it also applies to public averaging by replacing average estimates with noisy values.

4.3.3 Robustness to data injections

In addition to churns, a major threat to correctness of gossip averaging is data injections, that we mentionned in chapter 3, “State of the art”. Let us formalize it through Definition 2.

Definition 2 (Data injection)**.** Let $a_u[t]$ denote the average estimate of user u at time t. Suppose that user u engages in a gossip average iteration with neighbor v. Then, we call *data injection*, the fact that u sends $a_u[t] + \epsilon$ to v instead of $a_u[t]$, where $|\epsilon| > 0$, and all participants update their averages as if $a_u[t] + \epsilon$ were $a_u[t]$. In the context of the push-pull algorithm, it means that u and v update their estimates such that $a_u[t+1] = a_v[t+1] = \frac{1}{2}(a_u[t] + a_v[t] + \epsilon)$.

At first view, such event makes the protocol converge towards $\frac{1}{|U|}\left(\sum_{u\in U} a_u[0] + \epsilon\right)$. Obviously, repeated injections can soon become dangerous if no corrective action is taken (*cf.* the detailed analysis of [39]). That is why we dedicate this subsection to their detection.

First of all, notice that HE can help us here as it did for the ZKRdP. Indeed, we can make a ZKP of the averaging equation

$$a_u[t+1] = \frac{a_u[t] + a_v[t]}{2} \iff 2a_u[t+1] = a_u[t] + a_v[t] \tag{4.7}$$

by proving that

$$\mathcal{E}(a_u[t+1]; r_u r_v \bmod n)^2 = \mathcal{E}(a_u[t]; r_u)\,\mathcal{E}(a_v[t]; r_v)(r_u r_v)^n \mod n^2 \tag{4.8}$$

for a given Paillier cryptosystem and any pair $\{r_u, r_v\} \in \mathbb{Z}_n^{*2}$. Generalizing this ZKP to all the average exchanges, we come up with a two-phases secure averaging protocol made up of Algorithms 5 and 6.

The idea behind Algorithm 5 is to enable traceability of all the computations that lead to average consensus. When consensus is reached, users can run Algorithm 6 so as to verify that no data injection happened. We call this verification algorithm the Zero-Knowledge Average Consensus Proof (ZKACP). As a result, we claim Theorem 2. Regarding the notation of the algorithms, recall that for any user u, value y and nonce r, $\mathcal{E}_u^P(y; r)$ is the previously published ciphertext that supposedly equals $\mathcal{E}_u(y; r)$ where $\mathcal{E}_u$ is the Paillier encryption function of u. Yet, within this subsection, we will make a change in the meaning of $a_u[t]$ as it will represent the u's estimate after his tth average exchange.

Algorithm 5 Modified asynchronous pairwise push-pull for preparing the ZKACP

```
Input:  • Key pair $((K_u^{pub} = (n_u, g_u), K_u^{priv}))_{u\in U}$
        • Published ciphertexts $(\mathcal{E}_u^P(a_u[0]; r_u[0]) = \mathcal{E}_u(\tilde{x}_u; r_u[0]))_{u\in U}$
 1: for all $u \in U$ do
 2:     $t_u \leftarrow 0$
 3: while consensus is not reached do
 4:     Two users $u$ and $v$ choose one another: $u$ publishes $\mathcal{E}_v^P(a_u[t_u]; r_u[t_u])$ and $v$ publishes
        $\mathcal{E}_u^P(a_v[t_v]; r_u[t_v])$.
 5:     $u$ decrypts $\mathcal{E}_v^P(a_u[t_u]; r_u[t_u])$ to get $a_v[t_v]$ and $r_v[t_v]$. $v$ decrypts $\mathcal{E}_u^P(a_v[t_v]; r_u[t_v])$ to get $a_u[t_v]$
        and $r_u[t_v]$.
 6:     $u$ evaluates $c_v$ = CHECKDATAINJECTION($v$, $t_v$, $a_v[t_v]$, $r_v[t_v]$). $v$ evaluates $c_u$ = CHECKDATAIN-
        JECTION($u$, $t_u$, $a_u[t_u]$, $r_v[t_u]$).
 7:     if $c_u \wedge c_v$ then
 8:         $u$ and $v$ update their averages such that $a_u[t_u+1] = a_v[t_v+1] = \frac{1}{2}(a_u[t_u] + a_v[t_v])$.
 9:         $u$ computes $r_u[t_u+1] = r_u[t_u]r_v[t_v] \bmod n_u$. $v$ computes $r_v[t_v+1] = r_v[t_v]r_u[t_u] \bmod n_v$.
10:         $u$ publishes $\mathcal{E}_u(a_u[t_u+1]; r_u[t_u+1])$. $v$ publishes $\mathcal{E}_v(a_v[t_v+1]; r_v[t_v+1])$.
11:         $t_u \leftarrow t_u + 1; t_v \leftarrow t_v + 1$
12:     else
13:         Abort the exchange: $u$ and $v$ withdraw $\mathcal{E}_v^P(a_u[t_u]; r_u[t_u])$ and $\mathcal{E}_u^P(a_v[t_v]; r_u[t_v])$ from publica-
        tion. The honest user between $u$ and $v$ publishes a report of data injection so that other users can
        see the fraud attempt.
14: function CHECKDATAINJECTION($u$: User, $t$, $a$, $r$: Integer): Boolean
15:     $c_1 \leftarrow \mathcal{E}_u^P(a_u[t]; r_u[t]) = \mathcal{E}_u(a; r)$
16:     $c_2 \leftarrow t = 0$
17:     if $c_1 \wedge \neg c_2$ then
18:         $v' \leftarrow$ the user with whom $u$ did his $t$th average exchange
19:         $t' \leftarrow$ the corresponding exchange number according to $v'$
20:         $c_2 \leftarrow (\mathcal{E}_u^P(a_u[t-1])\,\mathcal{E}_u^P(a_{v'}[t'-1])r \bmod n_u^2 = \mathcal{E}_u(a; r)^2) \wedge (r_u[t-1]r_{v'}[t'-1] \bmod n_u = r)$
21:     return $c_1 \wedge c_2$
```

Algorithm 6 Zero-Knowledge Average Consensus Proof

```
Require: Consensus is reached
Input:   • $(t_u)_{u\in U}$ the number of average exchanges made by each user until now
         • Published ciphertexts $((\mathcal{E}_u^P(a_u[t]; r_u[t]))_{t=0}^{t_u})_{u\in U}$
 1: for all $u \in U$ do
 2:     $u$ publishes $a_u[t_u]$ and $r_u[t_u]$ in plaintext.
 3:     if $\mathcal{E}_u^P(a_u[t_u]; r_u[t_u]) = \mathcal{E}_u(a_u[t_u]; r_u[t_u])$ then
 4:         Add $u$ to cheaters.
 5:     for $t \leftarrow t_u, \ldots, 1$ do
 6:         $v \leftarrow$ the neighbor with whom $u$ did his $t^{\text{th}}$ average exchange
 7:         $t' \leftarrow$ the corresponding exchange number according to $v$
 8:         $u$ publishes $r_u[t]$ and $r_u[t-1]$. $v$ publishes $r_v[t']$ and $r_v[t'-1]$. If $v$ does not agree with
    $r_u[t]$ and $r_u[t-1]$, he reveals the true values, those that verify $\mathcal{E}_u^P(a_u[t]) = \mathcal{E}_u(a_u[t]; r_u[t])$ and
    $\mathcal{E}_u^P(a_u[t-1]) = \mathcal{E}_u(a_u[t-1]; r_u[t-1])$, and $u$ is added to the list of cheaters. This proof costs the
    disclosure of $a_u[t]$ and $a_u[t-1]$, but it avoids $u$ to make a honest user wrongly accused of injection.
    Do the same if $u$ does not agree with $r_v[t']$ and $r_v[t'-1]$.
 9:         if $(\mathcal{E}_u^P(a_u[t]))^2 \neq \mathcal{E}_u^P(a_u[t-1])\,\mathcal{E}_u^P(a_v[t'-1])r_u[t] \bmod n_u^2) \vee (r_u[t] \neq r_u[t-1]r_v[t'-1] \bmod n_u)$
    then
10:             Add $u$ to the list of cheaters.
11:         if $(\mathcal{E}_v^P(a_v[t']))^2 \neq \mathcal{E}_v^P(a_v[t'-1])\,\mathcal{E}_v^P(a_u[t-1])r_v[t'] \bmod n_v^2) \vee (r_v[t'] \neq r_v[t'-1]r_u[t-1] \bmod n_v)$
    then
12:             Add $v$ to the list of cheaters.
```

Theorem 2. *By executing Algorithms 5 and 6, every attempted and effective data injection is detected and the cheaters are identified. For a data injection involving $u, v \in U_A$ at exchange t_u resp. t_v, i.e. $a_u[t_u] \neq \frac{1}{2}(a_u[t_u-1] + a_v[t_v-1]) \neq a_v[t_v]$, the cheaters are u, v and, if they exist, the users who complete an average exchange with either u or v at t_u+1 resp. t_v+1.*

Proof. We assume that $\forall u \in U, \mathcal{E}_u^P(a_u[0]) = \mathcal{E}_u(\tilde{x}_u)$ thanks to the verification phase. That is, u cannot falsify the ciphertext of his initial average. Thus, a data injection attempt at first average exchange is systematically detected and its cheater identified through function CHECKDATAINJECTION of Algorithm 5. It corresponds to the case where c_1 is false and $t = 0$.

For $t > 0$, first of all, every attempted and effective injection is detected because of the systematic execution of function CHECKDATAINJECTION. Indeed, this function checks that the average value sent by a user matches with the value he committed to at the end of his last exchange, through condition c_1. If c_1 is false, there is a data injection attempt. If c_1 is true, it means that the tested user, u, has published an authentic ciphertext $\mathcal{E}_u^P(a_u[t])$ so that we can evaluate c_2 confidently. Which, if false, proves an effective data injection.

Secondly, because CHECKDATAINJECTION is run before accepting an average exchange, no user can complete an exchange without being aware of a potential or effective data injection. This implies that any user who completes an exchange despite an injection attempt becomes a cheater himself, and that any user who completes an exchange with any of two cheaters who injected data just before becomes an accomplice to the injection.

However, for a data injection of user u at t_u, Algorithm 5 does not enable the user with whom u would exchange averages at $t > t_u + 1$ to detect the injection at t_u. This case is handled by Algorithm 6 which traces back all exchanges so that everyone can detect all cheaters afterwards. □

Last but not least, remark that a similar ZKP can be performed to prove that each user correctly updated his estimate when a churn happened (*cf.* Theorem 1). Indeed, one can prove that $a_v[t+1] = a_v[t] + \delta_{v,w}$ through equation $\mathcal{E}_v(a_v[t+1]) = \mathcal{E}_v(a_v[t])\,\mathcal{E}_v(\delta_{v,w}) \bmod n_v^2$. This operation could be incorporated to Algorithms 5 and 6 as part of future works.

Consensus detection

The framework hereby presented requires detecting consensus and revealing all average estimates. Consensus detection is not trivial in the fully decentralized setting and it was specifically addressed in [34, 35, 49], for instance. However, in our case, users could publish their average estimate $a_u[t_u]$ in plaintext when they think they reached consensus. For example, a user may assume that the community reached *(global)* consensus when his estimate equals all his neighbors' estimates *(local consensus)* at their last exchange. If users publish their estimates when they effectively reached consensus, no private information is disclosed. Meanwhile, the case where users would publish their estimates before having reached global consensus should not be a serious threat. Indeed, privacy is already handled by the randomisation phase and many gossip average iterations.

When a user publishes his average estimate in plaintext, other users can run Algorithm 6 to detect data injections. Afterwards, supposing that no user injected data, there are two cases. If some estimates are still different, users can compute the average in one step, $\bar{x} = \sum_{u \in U} a_u[t_u]$, *i.e.* they can use the public averaging strategy explained in subsection 4.3.1. Otherwise, users reached consensus as expected and they know the average value $\bar{x}$.

Furthermore, this system is robust to malicious users who would try to delay consensus by not publishing their estimates. Indeed, suppose that honest user u and malicious one v have reached consensus. We can see that estimate $a_u[t_u]$ relates to ciphertexts $\mathcal{E}_u^P(a_u[t_u])$ and $\mathcal{E}_v^P(a_v[t_v])$. Then, although we cannot ask u to commit to $\mathcal{E}_v^P(a_v[t_v])$, his publishing of $a_u[t_u]$ suggests that v reached consensus as well. This being the same with other neighbors of v, there exist a time at which the whole community knows that v should have published his estimate. If he does not, other users can consider that he disconnected.

Summary

This concludes this section about the averaging phase of GOPA. In summary, we debated whether gossiping was so important in the semi-decentralized setting, and we provided two frameworks for churn tolerance and for systematic detection of data injection within gossip average consensus.

4.4 Synthesis: Managing churns at a global level

We detailed the user-centric setting of GOPA in subsection 4.1.1 and we proposed many improvements for GOPA's robustness to churns and malicious adversaries in the previous sections. This section unifies these contributions around a globally churn-tolerant version of GOPA.

For the sake of simplicity, suppose here that the GOPA's averaging phase works with public averaging, *i.e.* users directly publish their noisy value (*cf.* subsection 4.3.1). Additionally, assume that:

1. all cheating detection mechanisms work well;
2. all churn detectors work well such that permanent disconnections are not confused with minor perturbations (*e.g.* casual message losses);
3. each time a fraud is detected, cheaters get removed from the protocol, which is the same as if they disconnected.

Furthermore, as we have seen before, it is essential that:

- users exchange noise with enough neighbors during the randomization phase;
- the ZKRdP allows users to reach an acceptable trade-off between privacy and probability of detecting cheaters;

- the noisy values shared during the averaging phase are up to date with all data related to noises. For example, if $w \in U$ dropped out, we cannot see $\tilde{x}_w$, nor the ciphertexts of the noise he exchanged, *etc.*

This being said, suppose that a GOPA session is running in a valid state, *i.e.* a registry of verified noisy values whose noises sum to zero is already published. Then, we can manage churns through the two following strategies.

Strategy 1 (Global management of disconnections)**.** For any $w \in U$ who disconnects:

1. $\tilde{x}_w$ is removed from the public registry of noisy values, which is then labeled 'invalid' (its hidden noises no longer sum to zero).
2. Users start to take corrective steps. For all $u \in \mathcal{N}_w$:
   (a) u removes $\delta_{u,w}$ from Δ_u.
   (b) If u realizes that he lacks noises w.r.t. to privacy threats, he exchanges additional noises with new neighbors, or even retakes randomization phase from the beginning.
   (c) u checks that enough of his noises are revealed in plaintext to comply with the settings of the verification phase (privacy budget β). If u exchanged new noises, at least a part of them should be revealed and verified.[3]
   (d) u updates ciphertexts $\{\mathcal{E}_u^P(\delta_{u,v})\}_{v \in \mathcal{N}_u}$, $\mathcal{E}_u^P(\Delta_u)$, $\mathcal{E}_u^P(\tilde{x}_u)$ and passes through the verification phase again. His status (honest or malicious) is recorded in an adequate public registry.
   (e) Assuming that u was proved honest, he publishes $\tilde{x}_u$.
3. The registry of noisy values is then labeled 'valid': all users can compute the exact average $\bar{x} = \frac{1}{|U|}\sum_{u \in U} x_u = \frac{1}{|U|}\sum_{u \in U} \tilde{x}_u$. Previous steps are taken as many times as necessary until every concerned users have successfully passed the ZKRdP.

Strategy 2 (Global management of late arrivals)**.** Late arrivals are easier to manage than disconnections. The difference is that when a user w late joins the network, the registry of noisy values stays valid all along the integration (or rejection) of w. The registry just has to be updated and enriched with $\tilde{x}_w$ through the following procedure:

1. Already connected users exchange additional noises with w.
2. For all $u \in \mathcal{N}_w \cup \{w\}$: u adds $\delta_{u,w}$ to Δ_u and takes steps 2b to 2e of Strategy 1.

Of course, if w cheated during the randomization phase but $u \in \mathcal{N}_w$ did not, w gets blacklisted and u just has to cancel all the exchanges he did with him.

As a result, Strategies 1 and 2 constitute a robust and flexible user-centric protocol against churns. One of their limitations is that some malicious users may disconnect and reconnect very often to avoid that the network reaches a valid state. Nonetheless, stability can be increased by saving successive valid noisy values registries so that resulting averages stay operable in a manner. The idea is that each user assigns his noisy value to an *epoch* number. When a user late joins a GOPA session, he starts from the current epoch number. And each time a user updates his noisy value, he increments his epoch number. Together with, to allow churn management, each epoch number t should map to a pair $(G_R[t], G_V[t])$ of randomization and verified randomization graphs. Such mapping should be community-wide to avoid confusion in the case a churn happens while some users are already trying to recover from a previous one.

[3]We encourage a deeper analysis of such procedure, namely considering malicious curious users.

Chapter 5

Discussion

The objectives of our study were vast and numerous and so we could not address all of them in this Master internship. The initial topic of my internship was to improve GOPA's robustness to dropouts and malicious adversaries, to care about consistency of input values[1], and lastly, to implement a prototype of GOPA (*cf.* chapter 6). My research work not only consisted in finding solutions but also in identifying problems. Listing threats and delineating a study area was a progress in itself. Namely, by looking for flaws in GOPA, I realized that we could treat the problem of dropouts as a special case of churns. Also, the malevolent purposes and actions that we listed in section 2.2 were identified as my work progressed. In the next section, we synthetize how chapter 4, "Analysis and theoretical improvement of GOPA" answered our problematic. When possible, we also compare our solutions to the literature.

5.1 Theoretical results and comparison with the literature

We made a minimal effort regarding the detection of churns as we considered a semi-decentralized architecture where logging in or out was simple when using a communication server. Still, we provided a state of the art that may help with fully decentralized networks as well. Regarding the randomization phase and churns, we gave guidelines to decide a privacy-preserving number of noises that each user should exchange at initialization. Meanwhile, we built a theoretical framework to rectify averaging errors due to churns at the stage of gossip averaging. The solution was designed around the push-pull algorithm. But in fact, it is generalizable to any average consensus algorithm since it mainly consists in an update of peers' average estimates. Eventually, we unified our phase-wise solutions in an effort to make GOPA churn-tolerant at a global level. These contributions are specific to GOPA, but they may benefit to similar protocols [14, 18]. Aside from these references, it is hardly possible to compare our contributions to the literature because GOPA is already quite innovating.

Regarding robustness against malicious adversaries in gossip averaging, we built a HE-based framework to systematically detect data injections and identify their source users. Our approach is quite novel and more similar to blockchain techniques for certifying transactions than the statistical-based approaches we reviewed [16, 19, 33, 38, 44, 45, 50]. However, as we already mentioned before, cryptographical operations can be heavy. To give an order of magnitude, secure public keys are usually encoded on at least 2048 bits, whereas most of statistical applications encode numbers on 64 or 32 bits.[2] In addition, our framework requires interdependent exchanges between users whose individual delays may combine together and affect the speed of

[1] This third topic could not be treated. In short, it relates to ZKPs that each users' private data y falls into its allocated range $[y_{min}, y_{max}]$ in order to be consistent with the averaging goal and the privacy budget of the community.

[2] For more details, we benchmarked achievable performance for Paillier encryption on Android devices in subsection 6.2.1.

the whole protocol in a snowball effect. This may particularly happen if we expanded function checkDataInjection of Algorithm 5 to the verification of multiple average exchanges. Indeed, it would appear a trade-off between detecting cheaters earlier and augmenting the duration of each verification. Actually, the idea of using HE for average consensus is not that new. In 2014, Lazzeretti *et al.* [32] proposed a privacy-preserving gossip consensus algorithm which does all the exchanges in the encrypted space through multiparty computations; Ambrosin *et al.* [1] revisited the protocol in 2017. However, it is not robust to churns, and authors did not consider data injections. Their protocol uses HE for privacy reasons, whereas our framework uses HE for honesty also. Alternatively, Ruan, Ahmad, and Wang [43] proposed a solution based on digital signature to prevent data injections, but it is only against those due to man-in-the-middle (MITM) attacks. In comparison, our solution targets data injections caused by internal users of the protocol.

On privacy aspects of GOPA, we designed a commitment scheme which prevents users from sampling noise with malicious purposes. However, this scheme remains vulnerable to MITM attacks, and it would benefit from using digital signatures as in Ruan, Ahmad, and Wang [43, Figure 3]. Besides, we provided an analysis for parameterizing the ZKRdP so that, with desired probability, it can resist too curious malicious collusions. Moreover, we built an algorithm that prevents cheaters from compromising the verification mechanisms. Last but not least, we would like to highlight the fact that all the verification procedures we proposed respect the following properties:

- cheating detection procedures are ZKP challenges, thus leveraging privacy, at least partially for the verification of P2P noises;
- any user can verify any operation (noise exchange, average exchange) by himself, thus making the protocol robust to proof falsification.

These properties are at the core of privacy and decentralization challenges. Albeit, there is a trade-off in the ZKRdP between privacy and chance of detecting cheaters. When conducting our research, we looked for verification strategies that would allow the verification of all noises. Still, because of privacy, such mechanisms may require that each subset of a user's noises be audited by a part of users, such that other users' knowledge about their neighbors' noises never exceed the privacy budget. Not only is it hard to build such mapping from a noises subset to a verifiers subset, but also it makes the ZKRdP particularly vulnerable to curious users collusions. Indeed, if there is at least one malicious user among the auditors of each subset of user u's noises, those may gather all noises $\{\delta_{u,v}\}_{v\in\mathcal{N}_u}$ and then guess $x_u = \tilde{x}_u - \sum_{v\in\mathcal{N}_u} \delta_{u,v}$ when $\tilde{x}_u$ is revealed during the averaging phase. Another drawback of this technique is that malicious users could return false results but no one could prove them wrong. That would not respect the second property listed above.

This concludes our theoretical results, which remain to be experimented.

5.2 Research prospects

Henceforth, I would like to discuss future perspectives for this research.

First of all, we considered the semi-decentralized setting (*cf.* subsection 4.1.2) in almost all this study. However, chapter 3, "State of the art", shows that many researchers are interested with fully decentralized architectures. Adapting GOPA to these architectures may be interesting for those people.

Secondly, subsection 4.2.7 identifies a potential privacy breach in the case malicious users reveal more noises than expected. This is a problem of major importance for me.

Thirdly, albeit we debated the need of gossip algorithms themselves in subsection 4.3.1, our analysis was only textual. Considering the heaviness of the ZKACP (*cf.* subsection 4.3.3), a numerical analysis to decide when to use public averaging instead of gossip averaging will highly benefit to the speed of the protocol.

Regarding gossip algorithms again, GOPA uses Push-Pull, which is very traditional and simple. The use of more recent algorithms, particularly Flow Updating [24] or Push-Flow [15] and the one of the MRCA framework [50, subsection III.B], may be profitable in terms of fault tolerance and convergence speed, or even for attack detection. Before this study, it was not evident how such algorithms would behave in environments hostile towards privacy. However, considering the improvements proposed in section 4.3, their integration to GOPA appears more achievable. Besides, a further improvement is to complete the algorithms of subsection 4.3.3 bringing robustness to data injections so as to make them consistent with the framework for churn management detailed in subsection 4.3.2. Furthermore, we could make these two solutions robust to private values changes while averaging (a problem also known as input values changes [15, 24]).

Regarding the privacy of the ZKRdP, a future work may investigate whether it is much needed that all users reveal the same fraction $1-\beta$ of their noises in plaintext. Indeed, I introduced direct privacy level γ_0 in subsection 4.2.4. Since it relates to the size of malicious collusions who want to guess private values from the data published for the ZKRdP, γ_0 is more related to privacy than β, which in turn directly relates to the chance of detecting cheaters. Supposing a fixed γ_0, users having a greater number of noises may have a greater β. Then, the more a user would perform noise exchanges, the more likely his potential frauds would be detected. On another note, we may design a subprotocol of the ZKRdP in which each user u had to disclose exactly $\lceil (1-\beta)d_u \rceil$ noises. By way of comparison, in subsection 4.2.4, we showed that the number of disclosures caused by Algorithm 2 exceeded $\lceil (1-\beta)d_u \rceil$ and followed a Poisson binomial distribution.

Ultimately, we mentioned DoS attacks as part of our disturbance models. Such attacks would allow malicious users to cause failure of honest ones, thus fostering new malicious strategies. However, we ran out of time to handle them and so we let it to further studies.

Chapter 6

Prototype implementation

As we have seen in the previous chapters, a big part of this internship was theoretical research. But of course, the MAGNET team is also interested in putting its theories into practice. To do so, my supervisor decided to implement an Android prototype of GOPA. A typical use case that we considered was that of users who conduct a survey on a mobile application and then compute statistics with GOPA. No prototype of GOPA was implemented before the beginning of my internship, so we essentially limited our work to the development of the initial version of the protocol (*cf.* section 2.1 and/or Appendix B). Our implementation is the subject of this chapter. Section 6.1 first presents its architecture. Section 6.2 presents a benchmark of Paillier cryptosystem libraries as well as other implementation tips. Section 6.3 details our implementation of GOPA. And section 6.4 synthesizes our results and provides next implementation steps.

I would like to outline that this prototype could not have been possible without the assistance of Nathalie Vauquier. She wrote about 90 % of the code, she set up the development environment, she installed all the prototype's dependencies and she gave me feedback on the mid-level algorithms we were conceiving. My contributions were mainly to provide Nathalie Vauquier with the pseudo-code of our prototype and to code a part of gossip averaging. Cesar Sabater helped us on the design of the algorithms and the architecture, and on questions related to Paillier cryptosystem. Additionally, Carlos Zubiaga Pena joined the project on the last month and implemented a part of the ZKRdP.

6.1 Overall architecture and messaging protocol

We built the prototype under the semi-decentralized setting in which users communicate through a message and publication server (*cf.* subsection 4.1.2). As a quick and easy solution, we developed a RESTful API to connect the Java client of the Android devices to a Python 3 server based upon the Flask library. To allow users communication, we chose to implement an asynchronous mail system illustrated on Figure 6.1. The principle of this system is that the server stores all the users' messages, updates its message database when a user wants to send a message to another, and delivers messages only when their recipients ask for them. We might have implemented another messaging protocol in which the server would directly forward messages to their recipients, but it was harder to code.

6.2 Homomorphic encryption

6.2.1 Implementing Paillier cryptosystem

One of the practical challenges in cryptography is about the computation cost of encryption and decryption operations. With that in mind, Paillier [41] proposed three versions of his cryptosystem (whose last one, Scheme 3, is the fastest) and additional implementation tips

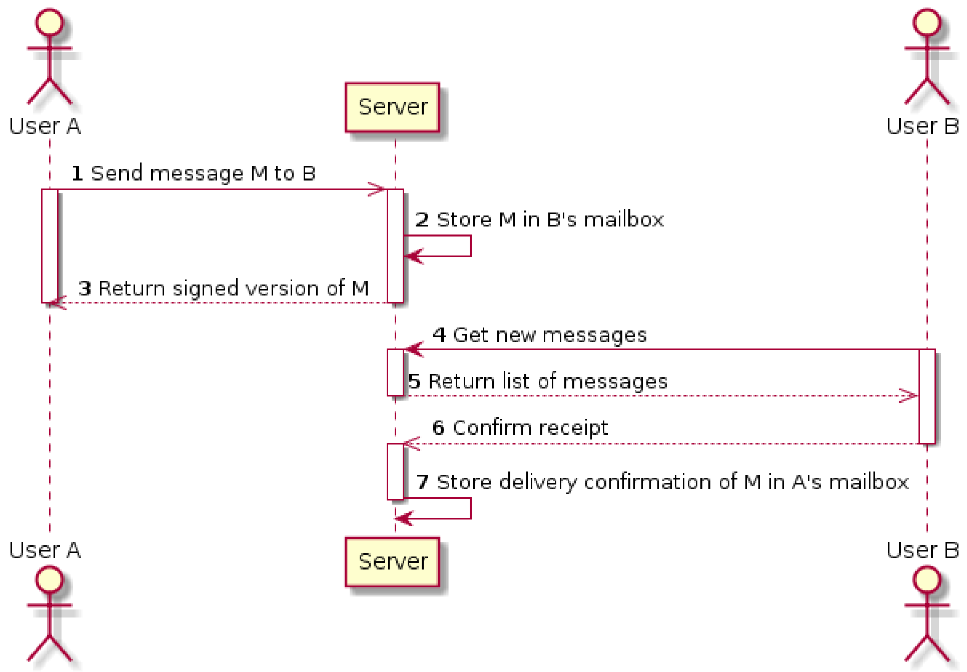


Figure 6.1. Sequence diagram of use case "user A sends message M to user B". As additional attributes, the signed version of M (step 3) contains a unique ID and a send date set by the server. By requesting its new messages to the server after step 7, user A will have the confirmation that B received M.

[41, section 7], which make its computational load theoretically smaller than that of the RSA cryptosystem. Nonetheless, research did not stop there and more optimizations were proposed in, *e.g.*, [3, 9, 25]. Since implementing the whole Paillier scheme is not easy, we searched open Paillier libraries for our prototype, computation speed being an important decision factor.

We first looked at generic cryptography libraries, namely Botan, BouncyCastle, Cripto-Comply, Crypto++, libsodium, OpenSSL, RELIC, or others mentioned in [30]. Not all are certified under the well-known U.S. Federal Information Processing Standard 140-2 for cryptological security. However, we did not limit our choice to such certification in the context of a proof of concept. Moreover, to our knowledge, no FIPS 140-2 certified libraries support Paillier encryption. Among libraries for Paillier cryptosystem, we found RELIC (C), libpaillier (C), go-go-gadget-paillier (Go), javallier (Java), jpaillier (Java), python-paillier (Python 3), rust-paillier (Rust) and jspaillier (JavaScript), which are not all optimized.

We first tested jpaillier [31] for starting with easy-to-run code on Android, but we would definitively give up the library because of excessive arithmetic exceptions. Indeed, the Java prime number generator used by jpaillier sometimes generates non-prime numbers, which makes common computations fail at random. We then oriented towards new solutions. In a benchmark of some Paillier libraries [7], we identified libpaillier [2], and an optimized version of it, mcornejo/libpaillier [6], as the potentially fastest and most stable library. The advantage of a C library is that it can natively be interfaced with Android devices through the Android Native Development Kit (NDK). We did so, even if it was not so easy to install the dependency of libpaillier on Android devices. The dependency at issue was the GNU Multi-Precision (GMP) library for arbitrary precision arithmetic. Since we developed a Python 3 server with Flask, we tested the easy-to-install python-paillier library [12]. We did so to see whether python-paillier would be fast enough to be used on the server, or we would wrap C library mcornejo/libpaillier in Python, which may require more implementation steps.

We present our own benchmark of all these solutions in Figure 6.2. As you can see, mcornejo/libpaillier is the best library in terms of computation speed. It is at least twice faster than libpaillier for decryption, while python-paillier makes no difference between encrypting an integer of 64 bits (`int64`) or of arbitrary size (`BigInteger`). However, python-paillier is approximately 8 times slower than mcornejo/libpaillier. It is even slower than jpaillier, but to our knowledge, it does not have this major drawback of throwing unsolvable exceptions.

In conclusion, we chose to use mcornejo/libpaillier on Android devices and python-paillier on the server. If python-paillier reveals too slow for the server's tasks in further development, it will likely be worth it to wrap mcornejo/libpaillier in Python. Notwithstanding, mcornejo/libpaillier does not allow the programmer to choose the encryption nonce $r \in \mathbb{Z}_n^*$. Then, when we say that we used it, we actually forked it to add needed functionalities without modifying its optimizations. Last but not least, we coded GOPA in Java but the Paillier library in C. However, calling C from Java involves context switch costs. It might be interesting to monitor this cost if the prototype needs higher performance.

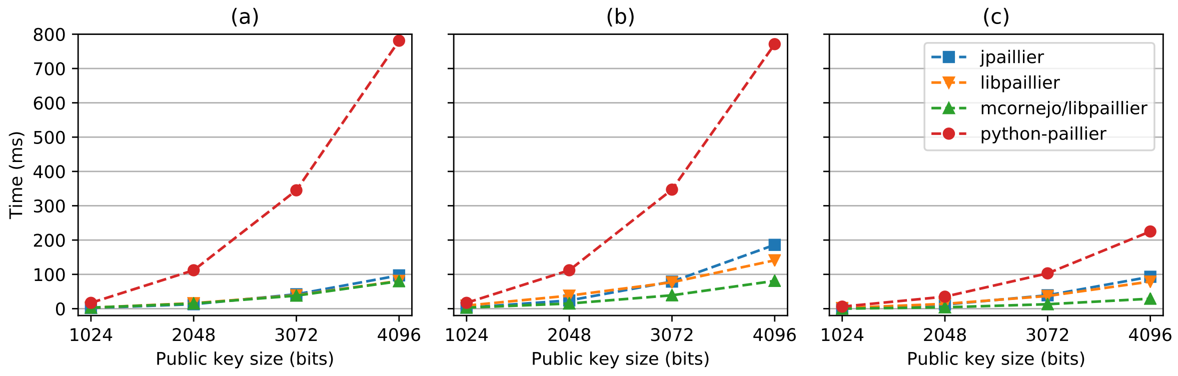


Figure 6.2. Performance benchmark of Paillier libraries showing (a) the mean encryption time of an `int64`, (b) the mean encryption time of a `BigInteger` of the same size as the public key and (c) the mean decryption time of a ciphertext. Tests of python-paillier have been run on a desktop computer (MacBook Pro, 2.5 GHz Intel Core i7, 16 Go 1600 MHz DDR3 RAM) hosting the server, while tests of jpaillier, libpaillier and mcornejo/libpaillier have been run on Android devices emulated on the same computer.

6.2.2 Encryption of negative integers and floats

Paillier encryption is defined over messages in $\mathbb{Z}_n$, but there are numerous applications where we would like to encrypt real numbers. This would allow us to verify additive relations between some real values, *e.g.*, $\tilde{x}_u = x_u + \Delta_u$ through Paillier homomorphic property $\mathcal{E}_u(\tilde{x}_u) \overset{?}{=} \mathcal{E}_u(x_u)\,\mathcal{E}_u(\Delta_u) \bmod n_u^2$. This is expected to be a standard case in machine learning applications.

Bits representation

The most basic encoding from floats to integers uses their bits representation. It is so basic that it is implemented within native methods `floatToIntBits(float value)` and `intBitsToFloat(int bits)` of the `Float` class of Java. The encoding consists in two steps:

1. Get $(b_1, \ldots, b_n) \in \{0,1\}^n$ the n bits of the float number.
2. Return integer $\sum_{i=1}^{n} b_i \cdot 2^{i-1}$.

The corresponding decoding function is trivial, knowing that according to the IEEE Standard for Floating-Point Arithmetic (IEEE 754), a normalized float encoded on 64 bits has value $(-1)^s 2^e (1+m)$ where $s \in \{0,1\}$ is its sign bit, $e \in [-1022, 1023]$ is an integer encoded on its 11 exponent bits and $m \in [1,2)$ is the *mantissa*, an integer encoded on its 52 remaining bits such that $m = \sum_{i=1}^{52} m_i \cdot 2^{-i}$ where m_i is the ith bit of the mantissa.

This simple encoding may be used for noise sampling in the randomization phase (*cf.* the hash function of subsection 4.2.2). Nonetheless, it is not convenient for encoding floats whose verification will rely on Paillier homomorphic property, *e.g.* $\mathcal{E}_u(\tilde{x}_u) \overset{?}{=} \mathcal{E}_u(x_u)\,\mathcal{E}_u(\Delta_u) \bmod n_u^2$. Indeed, let $\text{enc}\colon \mathbb{R} \mapsto \mathbb{Z}_n$ be an encoding function. When encrypting message $m \notin \mathbb{Z}_n$, we cannot do $\mathcal{E}(m)$ but only $\mathcal{E}(\text{enc}(m))$. Thus, $\forall (m_1, m_2) \notin \mathbb{Z}_n^2$ we want a function enc that verifies

$$\mathcal{E}(\text{enc}(m_1 + m_2)) = \mathcal{E}(\text{enc}(m_1))\,\mathcal{E}(\text{enc}(m_2)) \mod n^2. \tag{6.1}$$

Shifting-rescaling

A first encoding that would respect this condition for both floats and negative integers is $\text{enc}(m; r, s) = r(m + s)$ where r and s are rescaling and shifting factors. But actually, we cannot use the same shifting factor for different values. Let us explain with an example. Suppose that we set ranges $\forall u \in U, |x_u| \leq B_x$ and $\forall (u,v) \in E, |\delta_{u,v}| \leq B_\delta$ for private values and noises, and that we set a maximum of B_d neighbors per user, *i.e.* $\forall u \in U, d_u \leq B_d$. As a result, total noises and noisy values have ranges $\forall u \in U, |\Delta_u| \leq B_\Delta, |\tilde{x}_u| \leq B_{\tilde{x}}$ where $B_\Delta = B_d B_\delta$ and $B_{\tilde{x}} = B_x + B_\Delta$. Also, suppose that all values have a fixed precision of p decimal points ($p = 0$ if values are integers). Then, we have

$$\begin{aligned} & \tilde{x}_u = x_u + \Delta_u \\ \iff & \underbrace{10^p(\tilde{x}_u + B_{\tilde{x}})}_{\in \mathbb{N}} = \underbrace{10^p(x_u + B_x)}_{\in \mathbb{N}} + \underbrace{10^p(\Delta_u + B_\Delta)}_{\in \mathbb{N}} \\ \iff & \text{enc}(\tilde{x}_u; 10^p, B_{\tilde{x}}) = \text{enc}(x_u; 10^p, B_x) + \text{enc}(\Delta_u; 10^p, B_\Delta). \end{aligned}$$

Therefore, if and only if $10^p(\tilde{x}_u + B_{\tilde{x}}) < n_u$ where n_u is the first parameter of u's Paillier public key, we can verify the equality above by checking that

$$\mathcal{E}_u(\text{enc}(x_u; 10^p, B_{\tilde{x}})) \overset{?}{=} \mathcal{E}_u(\text{enc}(x_u; 10^p, B_x))\,\mathcal{E}_u(\text{enc}(\Delta_u; 10^p, B_\Delta)).$$

QED. Note that $10^p(\tilde{x}_u + B_{\tilde{x}}) < n_u$ should not be hard to verify since n_u has to be a very great number.

Modular encoding

Paillier arithmetic is modular, so a slightly different encoding function would be $\text{enc}(m; r, n) = rm \bmod n$. The multiplication by r allows to get an integer and the modulo is made to transform negatives into positives. Because of the properties that hold on $\mathbb{Z}_{n^2}^*$, this encoding verifies (6.1).

For a more advanced modulo-based encoding of floats, refer to python-paillier[1]. Its encoding enables to encrypt and add encrypted numbers with different levels of precision.

Having presented encodings for encrypting floats, we might make abuses of notation by writing $\mathcal{E}(y)$ where $y \notin \mathbb{N}$ in the sequel. In the next section, we go to the main topic of the chapter, the GOPA prototype.

6.3 GOPA

This section is made to help other persons to reproduce our prototype of the GOPA protocole. Our implementation supports almost none of our contributions of chapter 4, "Analysis and theoretical improvement of GOPA". It is just an implementation of the latter version of GOPA featured in Appendix B. It works only in a perfect network without churns and cheaters. When a user runs a GOPA instance, he can only go from randomization to averaging. Some mechanisms for detecting cheaters are implemented, but none regarding their removal.

[1] `https://python-paillier.readthedocs.io/en/stable/phe.html#module-phe.encoding`

6.3.1 Randomization phase

Appendix A, Algorithm 8 contains the pseudo-code on which we based our Android implementation of GOPA's randomization process. It is the first algorithm that each user runs during the randomization phase. Refer to Table A.1 for the description of its symbols. Currently, the algorithm does not allow a user to restart the randomization process if he has completed it once.

Because we want a decentralized protocol, the algorithm is presented from a user's first person perspective. It works in two phases. First, it manages incoming messages knowing that a noise exchange respects the sequence described in Figure 6.3. This is the only contribution of the previous chapter that we implemented. In particular, we coded Gaussian randomization. Second, the algorithm requests neighbors who did not already engaged in an exchange. Additionally, it resets the status of neighbors who may not have answered (messages may have been lost, some temporary faults may have happened). These two stages are repeated until noise have been exchanged with the expected number of neighbors.

Also, we establish the convention that a noise exchange starts with the request preceding the firstly sent noise sample. Then, if two noise requests cross each other, the initiator is the one whose request is the more recent. A noise sample is either a response to a noise request or a response to a noise sample (if the sender is the guest of the exchange). Moreover, an exchange terminates with a noise confirmation, which is a receipt confirmation of the lastly sent noise sample. In addition, to make the protocol fluent and because the randomization is the first stage of GOPA, we set that a noise request acts as a neighboring invitation. Finally, by checking the coherence of every message (type, ID, *etc.*), Algorithm 8 is made robust to malicious users who would request illegitimate information.

6.3.2 Randomization proof and cheaters removal

The ZKRdP can be divided in two parts. First, the ZKPs of $\Delta_u = \sum_{v \in \mathcal{N}_u} \delta_{u,v}$ and $\tilde{x}_u = x_u + \Delta_u, \forall u \in U$, which requires no more than published ciphertexts (Algorithm 2, step 4). Second, the verification of zero-sum noises, *i.e.* $\delta_{u,v} = -\delta_{v,u}$, which requires the disclosure of a fraction of at least $1 - \beta$ of each users' noises in plaintext (Algorithm 2, step 5). This makes a total of $f(d_u) = 2 + 2\lceil(1-\beta)d_u\rceil$ verifications for user u. Thus, running an instance of the ZKRdP costs $\sum_{u \in U} f(d_u) = 2|U| + 2\sum_{u \in U}\lceil(1-\beta)d_u\rceil$ verifications. That is, at least $f(d_{min})|U|$ and at most $f(d_{max})|U|$, which leads to an asymptotic complexity of $\Theta(|U|)$ verifications. Of course, if we let all users verify values each on their own, the verification process would total up to $\Theta(|U|^2)$ verifications, which becomes quite heavy. However, remark that we can keep the cost of the ZKRdP to $\Theta(|U|)$ by delegating some verifications to the server. Furthermore, such solution allows users to profit from the server's computational power.

The exact tasks we can delegate to the server are those of testing equations (2.8) to (2.11) and publishing the corresponding cheater list. After what, users can remove cheaters from the protocol, not by asking the server to cut out physical connections with cheaters, but by stopping communicating with them and taking the adequate corrective steps[2]. Indeed, we want a decentralized protocol in which users are their own community administrators. This is also motivated by the potentiality of being in the presence of a cheating server. Actually, the less powerful is the server, the less it can benefit from cheating. First, remark that because all testing information are public, anyone can run the ZKRdP by itself. Then, by making two distinct tasks of cheating detection and cheaters removal, we enable any wrongly accused user to prove an error in the server's results. Hence, the server can never deceive users. In real life, a cheating server would quickly gain a poor reputation and be avoided by potential users of GOPA.

The procedure we implemented for the ZKRdP is Algorithm 7; it is described at a high level. This procedure takes as an input $(V_u)_{u \in U}$, the subsets of users whose noises are to be revealed.

[2]In Figure 4.1, this is equivalent to cutting symbolic links.

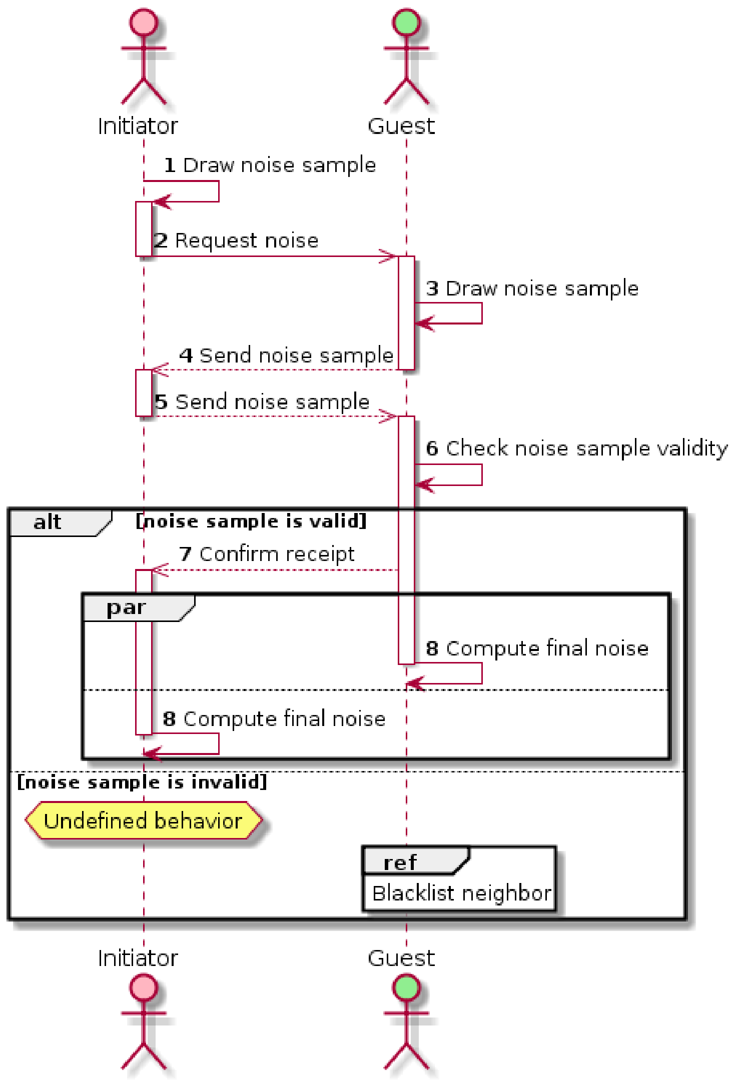


Figure 6.3. Sequence diagram of a noise exchange in the case noise request do not cross each others. W.r.t. the notation of subsection 4.2.2, if we name u the initiator and v the guest of the exchange, the 'noise request' is $\mathcal{E}_u(s_{u,v})$, 'noise samples' are $s_{u,v}$ resp. $s_{v,u}$ and 'final noises' are $\delta_{u,v}$ resp. $\delta_{v,u}$.

Subsection 4.2.5 presents a development path for how to choose them. Yet, we hard-coded $\forall u \in U, V_u = \mathcal{N}_u$ because of time constraints.

Last but not least, note that the verification of an individual user u can start as soon as u published all the required ciphertexts $\mathcal{E}^P_u(x_u)$, $\{\mathcal{E}^P_u(\delta_{u,v}); r_{u,v})\}_{v\in\mathcal{N}_u}$, $\mathcal{E}^P_u(\Delta_u)$ and $\mathcal{E}^P_u(\tilde{x}_u)$ whenever other users have not finished their randomization. In our current implementation, both the ZKRdP and the averaging phase start at the same time and run in parallel. In all cases, it is better for speed performance. Indeed, if all users are honest, no corrective step is needed and average consensus is just reached sooner. Else, if some users are malicious, corrective steps would be needed even if averaging had begun strictly after ZKRdP had passed. Moreover, corrective steps have the same costs whether users are averaging or not.

6.3.3 Gossip averaging phase

In this subsection, we now detail our implementation of gossip averaging, which is the basic pairwise push-pull algorithm. It consists in randomized exchanges and updates of users' average estimates until all users reached consensus. In our implementation, an average exchange between two users $u, v \in U$ requires three messages:

Algorithm 7 Server-aided ZKRdP

```
Input:  • Paillier public keys $(K_u^{pub} = (n_u, g_u))_{u \in U}$
        • Published ciphertexts $(\mathcal{E}_u^P(x_u), \{\mathcal{E}_u^P(\delta_{u,v}; r_{u,v})\}_{v \in \mathcal{N}_u}, \mathcal{E}_u^P(\Delta_u), \mathcal{E}_u^P(\tilde{x}_u))_{u \in U}$
        • $(V_u)_{u \in U}$ the neighbors whose noises are to be revealed by $u$
1: for all $u \in U$ do
2:     The server checks that $\mathcal{E}_u^P(\Delta_u) \overset{?}{=} \prod_{v \in \mathcal{N}_u} \mathcal{E}_u^P(\delta_{u,v}) \bmod n_u^2$ and $\mathcal{E}_u^P(\tilde{x}_u) \overset{?}{=} \mathcal{E}_u^P(x_u)\,\mathcal{E}_u^P(\Delta_u) \bmod n_u^2$. It publishes its results.
3:     for all $v \in V_u$ do
4:         $u$ publishes $\delta_{u,v}$ and $r_{u,v}$. $v$ publishes $r_{v,u}$.
5:         The server checks that $\mathcal{E}_u^P(\delta_{u,v}; r_{u,v}) \overset{?}{=} \mathcal{E}_u(\delta_{u,v}; r_{u,v})$ and $\mathcal{E}_v^P(\delta_{v,u}; r_{v,u}) \overset{?}{=} \mathcal{E}_v(-\delta_{u,v}, r_{v,u})$. It publishes its results.
6: Users consult the server's verification results. Then, they blacklist cheaters, or dispute the results, etc.
```

1. an average request, containing the u's current estimate;
2. an average response, containing the v's current estimate;
3. an average confirmation, which is a receipt confirmation of the average response.

After what, users update their estimates according to (2.12). Then, the exchange is said 'completed'. In the case two requests cross each other, users directly send an average confirmation instead of average responses.

Of course, when engaged with v, u cannot average with another user. But, to avoid getting indefinitely isolated in case v would not answer, we set that v has maximum T_R time units to answer to u before u disengages. Also, regarding the stopping criterion of the algorithm, we implemented an inexact but simple method where a user stops averaging when:

1. his average estimate is the same as all his last known neighbors', *i.e.* he reached *local consensus*;
2. and, he has not received an average request from anyone in T_P time units since his last exchange.

This method is inexact because some users could reach local consensus whenever global consensus (with all users) was not. Still, in practice, it can work for big enough values of T_P.

Figure 6.4 shows the user state diagram that sums up the averaging process. Eventually, Appendix A features Algorithm 9 as the base pseudo-code of our implementation, which has to be individually run by each user. Refer to Table A.2 for the description of its symbols.

6.4 Results and next steps

This section exposes the results we had with the Android prototype described in the previous section. Under the semi-decentralized setting, our prototype relies on a message and publication server for all communications. Among the good points, our prototype filters all exchanges so that no malicious users can ask for data that they are not expected to receive. We also took advantage of the semi-decentralized setting to reduce the complexity of the ZKRdP from $\Theta(|U|^2)$ to $\Theta(|U|)$. At the same time, we gave insights on how users can securely administrate their community by themselves in order to defend against a cheating server. And finally, among new contributions to GOPA, we implemented the noise sampling technique described in subsection 4.2.2. Besides, Figure 6.5 illustrates a successful execution of our basic Android application.

Among the bad points, we never tested our prototype on networks of more than 4 users, so we do not know if it scales well. This was notably due to the computation power needed to emulate more Android devices. In fact, it seems necessary to build a P2P network simulator

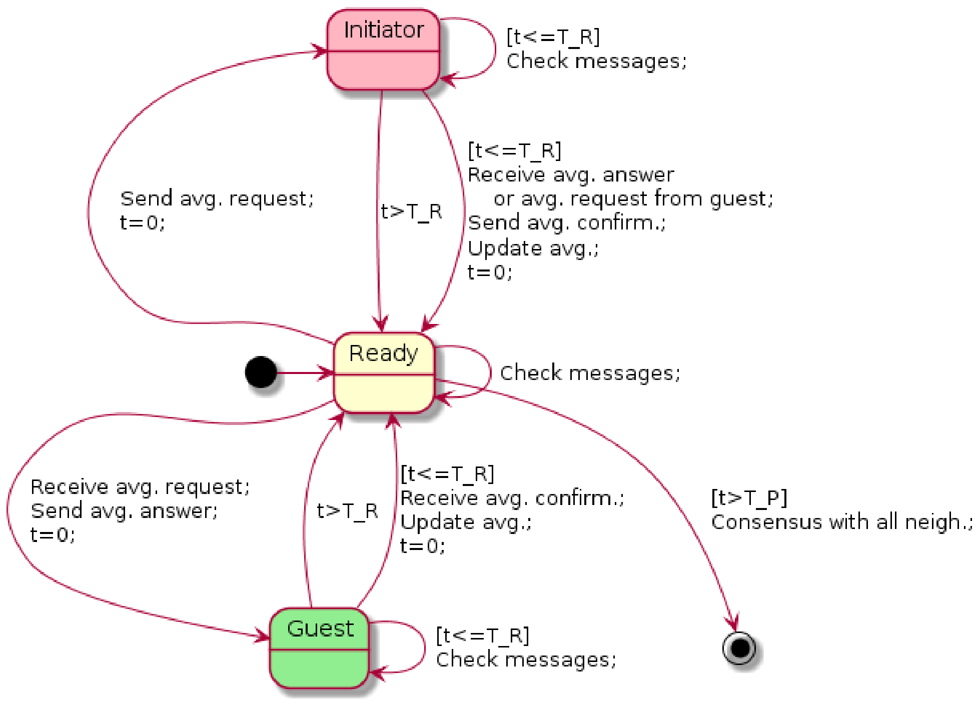


Figure 6.4. User state diagram in the averaging phase. `t` refers to a time counter which increments itself. Prefix `T` refers to timeout intervals.

(that replaces Android devices by simple objects) even to test GOPA on a network of a few dozens of users. Not only would it relax the constraint of computation power, but also we could simulate the specific delays of different devices. Testing and debugging would also be easier on a simulator than on (emulated) Android devices. Otherwise, by combining the power of multiple machines, we should be able to emulate about twenty Android devices. But we did not organize to do so as we were first occupied with the implementation itself.

As a matter of fact, even in our tests at very small scales, we faced problems related to interlocking processes. Indeed, our implementation is such that users must ask the server for their messages in order to receive them. But at the same time, we implemented mechanisms using reply timeouts in order to avoid that users indefinitely wait for an answer from of a disconnected peer. Consequently, it happens sometimes that users consult their messages and disengage from pairwise processes at rhythms such that they can always begin exchanges but never complete one. Thus, they block themselves in infinite messaging loops. Solving this problem is of major importance in future implementation.

A second high priority task is the implementation of Algorithm 4, because currently, each user reveals all his noises in plaintext during the ZKRdP.

And thirdly, the sequence of messages used for gossip averaging lacks robustness. Indeed, for any $u, v \in U$, if u sends an average confirmation to v but v does not receive it, u will have updated his average but v will have not, thus causing an unintended data injection of $\frac{1}{2|U|}(a_v[t] - a_u[t])$. This problem could be overcome through the implementation of our framework against data injection (*cf.* subsection 4.3.3).

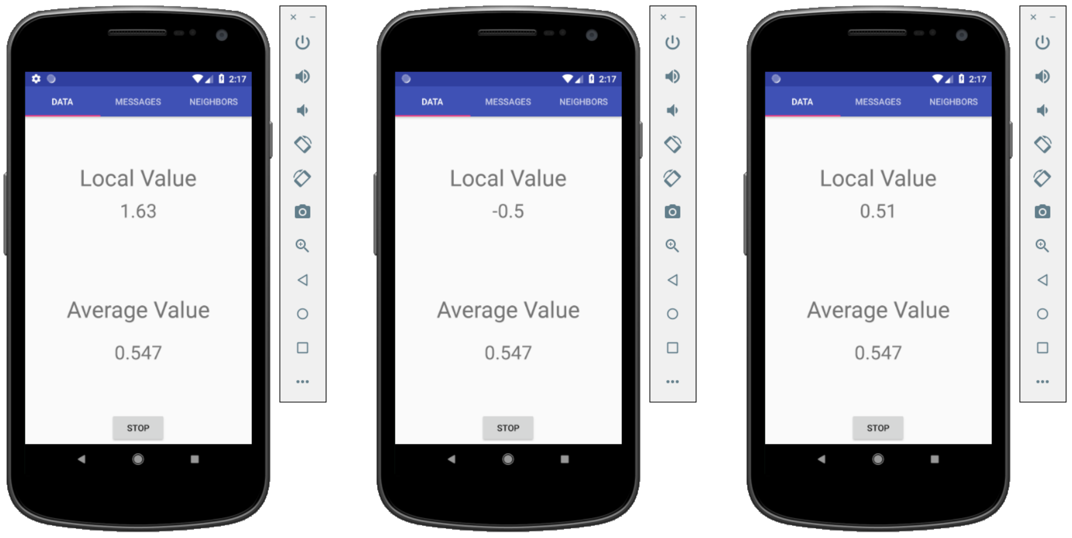


Figure 6.5. Example of the Android application at the end of a gossip averaging phase of three users. As expected the estimate of each user equals the average of their private ('local') values.

Chapter 7

Conclusion

This Master internship contributed to the theoretical improvement of the robustness of GOPA to churns and malicious adversaries. Particularly, our objectives were to guarantee privacy and correctness of averaging. In this study, we specified how to manage churns in the randomization and the averaging phases so that GOPA can maintain correctness of averaging almost in real time. This led us to a robust user-centric version of the protocol that manages both phases in parallel. Meanwhile, we developed many solutions against malicious adversaries trying to compromise privacy and correctness at various steps of the randomization phase. Additionally, we provided a ZKP for systematically detecting data injections during gossip averaging. These contributions are quite innovative in comparison with the reviewed literature. Aside from theoretical aspects, we developed a prototype of GOPA's original version which works on perfect networks, without churns or malicious adversaries. In the continuous improvement of GOPA, the MAGNET team foresee an implementation of our theoretical contributions. Besides, many problems could not be answered in the allocated time, but they may be treated as part of a next internship or PhD studies. Furthermore, GOPA as a whole may be adapted for the computation of other U-statistics such as the variance.

Through the last six months, I immersed myself deep into research at INRIA Lille Nord Europe. The experience was very different from my previous internships in companies. It asked me a lot of autonomy and self-confidence. It refined my English reading and writing skills. And it even recruited competencies that I learnt during the first years of my engineering studies. Although my efforts were mainly theoretical, based upon probabilistic and logical reasonings, the development of an Android prototype involved engineering work. Furthermore, this research made me discover totally new domains of science, including decentralized protocols, homomorphic encryption and other cryptographic primitives. Finally, this experience was instructive by preparing me for the PhD study that I will soon start.

References

[1] M. Ambrosin, P. Braca, M. Conti, and R. Lazzeretti, “ODIN: Obfuscation-based privacy-preserving consensus algorithm for Decentralized Information fusion in smart device Networks”, *ACM Trans. Internet Technol.*, vol. 18, no. 1, 6:1–6:22, Oct. 2017, ISSN: 1533-5399. DOI: `10.1145/3137573`.

[2] J. Bethencourt, *Paillier Library*, `http://acsc.cs.utexas.edu/libpaillier/` [v0.8], C library, Jul. 2006.

[3] A. Bouti and J. Keller, “Towards practical homomorphic encryption in cloud computing”, in *2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, Jun. 2015, pp. 67–74. DOI: `10.1109/NCCA.2015.20`.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, “Randomized gossip algorithms”, *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, Jun. 2006, ISSN: 0018-9448. DOI: `10.1109/TIT.2006.874516`.

[5] J.-Y. Chen, G. Pandurangan, and D. Xu, “Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 9, pp. 987–1000, Sep. 2006, ISSN: 1045-9219. DOI: `10.1109/TPDS.2006.128`.

[6] M. Cornejo, *Fork of lipbaillier*, `https://github.com/mcornejo/libpaillier` [Accessed commit 27b424e], GitHub repository, Feb. 2017.

[7] M. Dahl, M. Cornejo, and M. Poumeyrol, *Benchmarking Paillier encryption and an open source rust library*, `https://medium.com/snips-ai/benchmarking-paillier-encryption-15631a0b5ad8` [Accessed May 16, 2018], Snips, Feb. 2017.

[8] H. Daitch, *2017 Data Breaches — The Worst So Far*, `https://www.identityforce.com/blog/2017-data-breaches` [Accessed July 20, 2018], Identity Force, Dec. 2017.

[9] I. B. Damgård and M. J. Jurik, “A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system”, *BRICS Report Series*, vol. 7, no. 45, Dec. 2000, ISSN: 0909-0878. DOI: `10.7146/brics.v7i45.20212`.

[10] P. Dellenbach, A. Bellet, and J. Ramon, “Hiding in the Crowd: A Massively Distributed Algorithm for Private Averaging with Malicious Adversaries”, *ArXiv e-prints*, Mar. 2018. arXiv: `1803.09984 [cs.LG]`.

[11] P. Dellenbach, J. Ramon, and A. Bellet, “A decentralized and robust protocol for private averaging over highly distributed data”, in *NIPS 2016 Workshop on Private Multi-Party Machine Learning*, Barcelona, Spain, Dec. 2016.

[12] N. A. developers, *A Python 3 library for Partially Homomorphic Encryption*, `https://pypi.org/project/phe/` [v1.4.0], PyPI package, CSIRO’s Data61, Apr. 2018.

[13] A. D. G. Dimakis, A. D. Sarwate, and M. J. Wainwright, “Geographic gossip: Efficient averaging for sensor networks”, *IEEE Transactions on Signal Processing*, vol. 56, no. 3, pp. 1205–1216, Mar. 2008, ISSN: 1053-587X. DOI: `10.1109/TSP.2007.908946`.

[14] T. Dimitriou and M. K. Awad, “Secure and scalable aggregation in the smart grid resilient against malicious entities”, *Ad Hoc Networks*, vol. 50, pp. 58–67, 2016, ISSN: 1570-8705. DOI: `10.1016/j.adhoc.2016.06.014`.

[15] W. N. Gansterer, G. Niederbrucker, H. Straková, and S. Schulze Grotthoff, “Scalable and fault tolerant orthogonalization based on randomized distributed data aggregation”, *Journal of Computational Science*, vol. 4, no. 6, pp. 480–488, 2013, Scalable Algorithms for Large-Scale Systems Workshop (ScalA2011), Supercomputing 2011, ISSN: 1877-7503. DOI: `10.1016/j.jocs.2013.01.006`.

[16] R. Gentz, H. T. Wai, A. Scaglione, and A. Leshem, "Detection of data injection attacks in decentralized learning", in *2015 49th Asilomar Conference on Signals, Systems and Computers*, Oct. 2015, pp. 350–354. DOI: 10.1109/ACSSC.2015.7421145.

[17] C. Georgiou, S. Gilbert, and D. R. Kowalski, "Meeting the deadline: On the complexity of fault-tolerant continuous gossip", *Distributed Computing*, vol. 24, no. 5, pp. 223–244, Dec. 2011, ISSN: 1432-0452. DOI: 10.1007/s00446-011-0144-6.

[18] N. Gupta, J. Katz, and N. Chopra, "Privacy in distributed average consensus", *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9515–9520, 2017, 20th IFAC World Congress, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2017.08.1608.

[19] J. He, P. Cheng, L. Shi, and J. Chen, "SATS: Secure average-consensus-based time synchronization in wireless sensor networks", *IEEE Transactions on Signal Processing*, vol. 61, no. 24, pp. 6387–6400, Dec. 2013, ISSN: 1053-587X. DOI: 10.1109/TSP.2013.2286102.

[20] M. Jelasity, W. Kowalczyk, and M. van Steen, "Newscast computing", Vrije Universiteit, Department of Computer Science, Amsterdam, Internal report IR-CS-006, Nov. 2003.

[21] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks", *ACM Trans. Comput. Syst.*, vol. 23, no. 3, pp. 219–252, Aug. 2005, ISSN: 0734-2071. DOI: 10.1145/1082469.1082470.

[22] M. Jelasity and M. van Steen, "Large-scale newscast computing on the internet", Vrije Universiteit, Department of Computer Science, Amsterdam, Internal report IR-503, Oct. 2002.

[23] G. P. Jesi, A. Montresor, and M. van Steen, "Secure peer sampling", *Computer Networks*, vol. 54, no. 12, pp. 2086–2098, 2010, P2P Technologies for Emerging Wide-Area Collaborative Services and Applications, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.03.020.

[24] P. Jesus, C. Baquero, and P. S. Almeida, "Flow updating: Fault-tolerant aggregation for dynamic networks", *Journal of Parallel and Distributed Computing*, vol. 78, pp. 53–64, 2015, ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2015.02.003.

[25] M. J. Jurik, "Extensions to the Paillier cryptosystem with applications to cryptological protocols", PhD thesis, University of Aarhus, Department of Computer Science, Ny Munkegade, building 540, DK–8000 Aarhus C, Denmark, Aug. 2003.

[26] B. Kailkhura, S. Brahma, and P. K. Varshney, "Data falsification attacks on consensus-based detection systems", *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 1, pp. 145–158, Mar. 2017. DOI: 10.1109/TSIPN.2016.2607119.

[27] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information", in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, Oct. 2003, pp. 482–491. DOI: 10.1109/SFCS.2003.1238221.

[28] W. Kowalczyk and N. Vlassis, "Newscast em", in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds., MIT Press, 2005, pp. 713–720.

[29] S. Kumar and K. Dutta, "Intrusion detection in mobile ad hoc networks: Techniques, systems, and future challenges", *Security and Communication Networks*, vol. 9, no. 14, pp. 2484–2556, 2016, SCN-14-0221.R3, ISSN: 1939-0122. DOI: 10.1002/sec.1484.

[30] U. Kumar, T. Borgohain, and S. Sanyal, "Comparative analysis of cryptography libraries in IoT", *International Journal of Computer Applications*, vol. 118, no. 10, pp. 5–10, May 2015, ISSN: 0975-8887. DOI: 10.5120/20779-3338.

[31] H. Kunert, *A Java implementation of paillier cryptosystem*, https://github.com/kunerd/jpaillier [Accessed commit 1a01eee], GitHub repository, Nov. 2017.

[32] R. Lazzeretti, S. Horn, P. Braca, and P. Willett, "Secure multi-party consensus gossip algorithms", in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 7406–7410. DOI: 10.1109/ICASSP.2014.6855039.

[33] Q. Liu, X. Ren, and Y. Mo, "Secure and privacy preserving average consensus", in *2017 11th Asian Control Conference (ASCC)*, Dec. 2017, pp. 274–279. DOI: 10.1109/ASCC.2017.8287179.

[34] N. E. Manitara and C. N. Hadjicostis, "Distributed stopping for average consensus in directed graphs via a randomized event-triggered strategy", in *6th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, May 2014, pp. 483–486. DOI: 10.1109/ISCCSP.2014.6877918.

[35] N. E. Manitara and C. N. Hadjicostis, "Distributed stopping for average consensus in undirected graphs via event-triggered strategies", *Automatica*, vol. 70, pp. 121–127, 2016, ISSN: 0005-1098. DOI: 10.1016/j.automatica.2016.03.030.

[36] T. Mensink, W. Zajdel, and B. Krose, "Distributed em learning for appearance based multi-camera tracking", in *2007 First ACM/IEEE International Conference on Distributed Smart Cameras*, Sep. 2007, pp. 178–185. DOI: 10.1109/ICDSC.2007.4357522.

[37] Y. Mo and R. M. Murray, "Privacy preserving average consensus", *IEEE Transactions on Automatic Control*, vol. 62, no. 2, pp. 753–765, Feb. 2017, ISSN: 0018-9286. DOI: 10.1109/TAC.2016.2564339.

[38] M. Mousazadeh and B. T. Ladani, "Gossip-based data aggregation in hostile environments", *Computer Communications*, vol. 62, pp. 1–12, 2015, ISSN: 0140-3664. DOI: 10.1016/j.comcom.2015.02.002.

[39] M. Mousazadeh and B. Tork Ladani, "Randomized gossip algorithms under attack", *International Journal of Information Security*, vol. 13, no. 4, pp. 391–402, Aug. 2014, ISSN: 1615-5270. DOI: 10.1007/s10207-013-0221-x.

[40] J. Muszyński, S. Varrette, and P. Bouvry, "Reducing efficiency of connectivity-splitting attack on newscast via limited gossip", in *Applications of Evolutionary Computation*, G. Squillero and P. Burelli, Eds., Cham: Springer International Publishing, 2016, pp. 299–314, ISBN: 978-3-319-31204-0.

[41] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes", in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238, ISBN: 978-3-540-48910-8.

[42] A. Poenaru, R. Istrate, and F. Pop, "AFT: Adaptive and fault-tolerant peer-to-peer overlay—a user-centric solution for data sharing", *Future Generation Computer Systems*, vol. 80, pp. 583–595, 2018, ISSN: 0167-739X. DOI: 10.1016/j.future.2016.05.022.

[43] M. Ruan, M. Ahmad, and Y. Wang, "Secure and Privacy-Preserving Average Consensus", *ArXiv e-prints*, Mar. 2017. arXiv: 1703.09364.

[44] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Stochastic and deterministic fault detection for randomized gossip algorithms", *Automatica*, vol. 78, pp. 46–60, 2017, ISSN: 0005-1098. DOI: 10.1016/j.automatica.2016.12.011.

[45] M. Toulouse, H. Le, C. V. Phung, and D. Hock, "Defense strategies against byzantine attacks in a consensus-based network intrusion detection system", *Informatica*, vol. 41, no. 2, pp. 193–207, 2017.

[46] S. Voulgaris, M. Jelasity, and M. van Steen, "A robust and scalable peer-to-peer gossiping protocol", in *Agents and Peer-to-Peer Computing*, G. Moro, C. Sartori, and M. P. Singh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 47–58, ISBN: 978-3-540-25840-7. DOI: 10.1007/978-3-540-25840-7_6.

[47] G. Wang, Z. Wang, and J. Wu, "A local average broadcast gossip algorithm for fast global consensus over graphs", *Journal of Parallel and Distributed Computing*, vol. 109, pp. 301–309, 2017, ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2017.05.008.

[48] S. Wu and M. G. Rabbat, "Broadcast gossip algorithms for consensus on strongly connected digraphs", *IEEE Transactions on Signal Processing*, vol. 61, no. 16, pp. 3959–3971, Aug. 2013, ISSN: 1053-587X. DOI: 10.1109/TSP.2013.2264056.

[49] V. Yadav and M. V. Salapaka, "Distributed protocol for determining when averaging consensus is reached", in *45th Annual Allerton Conference on Communication, Control and Computing*, vol. 2, Sep. 2007, pp. 715–720.

[50] C. Zhao, J. He, and J. Chen, "Resilient consensus with mobile detectors against malicious attacks", *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 1, pp. 60–69, Mar. 2018. DOI: 10.1109/TSIPN.2017.2742859.

Lexicon

Acronyms

AFT Adaptive Fault-Tolerant
ANR National Research Agency

CDF cumulative distribution function
CNRS National Center for Scientific Research
CPER State-Region Planning Contract
CRIStAL Research Center on Computer Science, Signal Processing and Automatic Control of Lille

DataInG Data Intelligence Group
DoS denial of service
DPA Data Protection Act
DRG Distributed Random Grouping

EM expectation-maximization

FIPS Federal Information Processing Standard

GDPR General Data Protection Regulation
GMP GNU Multi-Precision
GOPA GOssip Private Averaging

HBC honest-but-curious
HE homomorphic encryption

i.i.d. independent and identically distributed
IDS intrusion detection system
IEEE 754 IEEE Standard for Floating-Point Arithmetic
INRIA National Research Institute of Computer Science and Automatic Control

MAGNET MAchine learning in information NETworks
MITM man-in-the-middle
MRCA Mobile Resilient Consensus Algorithm

NDK Native Development Kit
NLP Natural Language Processing

P2P peer-to-peer
PAMELA Personalized and decentrAlized MachinE Learning under constrAints

PMF probability mass function

r.v. random variable

SVO Set Values-Observer

UMR Mixed Research Unit

w.p. with probability
w.r.t. with respect to

ZKACP Zero-Knowledge Average Consensus Proof
ZKP zero-knowledge proof
ZKRdP Zero-Knowledge Randomization Proof

List of Symbols

x_u user u's private value
$\delta_{u,v}$ user u's noise exchanged with user v
Δ_u user u's total noise
$\tilde{x}_u$ user u's noisy value
$\bar{x}$ average of all users' private values
$a_u[t]$ user u's average estimate of $\bar{x}$ at time t
$\mathcal{N}_u$ user u's neighbors with whom he exchanged noise
d_u number of noise exchanges performed by user u
$\mathcal{E}_u(y)$ Paillier encryption of y with user u's public key
$\mathcal{E}_u^P(y)$ published ciphertext expected to equal $\mathcal{E}_u(y)$
γ_0 direct privacy level

$\mathcal{N}(\mu, \sigma^2)$ Normal distribution of mean μ and variance σ^2
$\mathcal{B}(p)$ Bernouilli distribution of success probability p
$\mathcal{B}(n,p)$ Binomial distribution describing the probability of k successes in n Bernouilli trials of success probability p
$\mathcal{PB}(\mathbf{p})$ Poisson binomial distribution describing the probability of k successes in n Bernouilli trials of success probabilities $\mathbf{p} \in [0,1]^n$
$\mathcal{H}(N,K,n)$ Hypergeometric distribution describing the probability of k successes in n simultaneous draws from N objects of which K represent a success

$\lceil \cdot \rceil$ ceil function
$\lfloor \cdot \rfloor$ floor function
$\neg$ logical negation, 'not'
$\wedge$ logical conjunction, 'and'
$\vee$ logical disjunction, 'or'
$\top$ true
$\perp$ false
$\vdash$ proves

Appendix A

Pseudo-code for prototyping randomization and gossip averaging

To improve legibility of the algorithms of this appendix, refer beforhand to the primitive classes and functions written just below. Then, see the next tables for the symbols of each algorithm.

Classes

Message

A message is an object that allows user-to-user, user-to-server and server-to-user communication. In our conception, every message passes through the server. It has for attributes:

id — the unique ID of the message (set by the server)
sender — the sender of the message
recipient — the recipient of the message
replyTo — (optional) the ID of the message to which this message replies
sendDate — the send date of the message, the moment at which the server receives it for mailing (set by the server)
data — the content of the message

User's methods

These are the methods that a user can use when running its instance of GOPA.

function GETNEWMESSAGES(): List<Message>
Get the list of new messages in increasing order of send date (older messages comes first).

function SEND(*recipient*: User, *type*: Any, *data*: Any): Message
Send a message to a peer. Parameter *type* can be any constant defined in the sequel. Parameter *data* can be anything: a string, a number, a set of values, *etc.* As we rely on a mailbox-like system, method SEND sends a message to the server, which instantiates it as required (adds an ID and a send date), puts it in the receiver's mailbox and also returns it to the sender.

function REPLY(m: Message, *type*: Any, *data*: Any): Message
Send another message as a reply to message m and return the message instantiated by the server.

Miscellaneous

function DRAW(P: Any, k: PositiveInteger): Any
Draw k samples from set or probability distribution P. If omitted, $k = 1$.

Table A.1. Table of symbols of Algorithm 8

Name	Type	Description
u	User	The user itself (analogous to `this` in Java).
$state_v$	State	The neighbor v's state in the related noise exchange. `READY` if he is ready to exchange noise, `INITIATOR` if he initiated a noise exchange, `GUEST` if he was invited in on, or `VALID` if the exchange was completed.
δ_v	Real	The noise refering to user v (analogous to $\delta_{u,v}$).
s_v	Real	The noise sample sent to user v (analogous to $s_{v,u}$).
s'_v	Real	The noise sample sent by user v (analogous to $s_{v,u}$).
r_v	Integer	The random number used to compute an encrypted number sent to v.
c'_v	Integer	The encrypted noise sample sent by v.
r'_v	Integer	The random number used by v to compute c'_v.
$lastSent_v$	Message	The last message sent to v.
T_R	Duration	The maximum waiting time for a reply message.

Table A.2. Table of symbols of Algorithm 9

Name	Type	Description
u	User	The user itself (analogous to `this` in Java).
$state$	State	The u's state. `READY` if he is ready to exchange average, `INITIATOR` if he initiated an average exchange, `GUEST` if he was invited in one.
M	List	The list of messages to process. We will use the same formalism as that of the Java List interface to operate over it.
$vAvg$	User	The user whom u is engaged in an exchange with.
i	Integer	The number of average exchanges performed by u.
$a[i]$	Real	The u's average estimate at iteration i.
i_v	Integer	The number (minus 1) of average exchanges made with user v.
$a_v[i_v]$	Real	The v's average estimate at exchange i_v.
$lastSent$	Message	In most cases, the last message sent to a neighbor.
T_R	Duration	The maximum waiting time for a reply.
T_P	Duration	The length of the probatory period before stating consensus.

Algorithm 8 Asynchronous randomization from user u's first person view

```
Input: $(K_v^{pub})_{v \in U}$ the public keys of all users
Ensure: $u$ has exchanged noise with all his neighbors
 1: procedure RANDOMIZE($k$: PositiveInteger)
 2:     for all $v \in U \setminus \{u\}$ do
 3:         $state_v \leftarrow$ READY
 4:     $\mathcal{N}_u \leftarrow$ DRAW($U \setminus \{u\}, k$) /* Build first instance of neighborhood */
 5:     while $\exists v \in \mathcal{N}_u, state_v \neq$ VALID do
 6:         PROCESSNOISEMESSAGES()
 7:         REQUESTNEIGHBORSFORNOISE()
 8:         WAIT($\dots$) /* Optionnal */
 9: procedure PROCESSNOISEMESSAGES()
10:     for all $m \in$ GETNEWMESSAGES() do
11:         $v \leftarrow m$.sender
12:         if $m$.type = NOISE_REQUEST then
13:             if $v \notin \mathcal{N}_u$ then
14:                 $\mathcal{N}_u \leftarrow \mathcal{N}_u \cup \{v\}$
15:             if ($state_v$ = READY) $\vee$ (($state_v$ = GUEST) $\wedge$ ($m$.sendDate > $lastSent_v$.sendDate)) then
16:                 $state_v \leftarrow$ INITIATOR
17:             if $state_v$ = INITIATOR then
18:                 $c'_v \leftarrow m$.data; $s_v \leftarrow$ DRAW($\mathcal{U}(0,1)$)
19:                 $lastSent_v \leftarrow$ REPLY($m$, NOISE_SAMPLE, ($s_v$, NULL)) /* Send $r_v$ = NULL to say that no
    ciphertext $\mathcal{E}_u(s_v)$ was sent before */
20:         else if ($m$.type = NOISE_SAMPLE) $\wedge$ ($state_v \in$ {INITIATOR, GUEST}) $\wedge$ ($m$.replyTo =
    $lastSent_v$.id) then
21:             $(s'_v, r'_v) \leftarrow m$.data
22:             if $state_v$ = GUEST then
23:                 $lastSent_v \leftarrow$ SEND($v$, NOISE_SAMPLE, $(s_v, r_v)$)
24:             else if $c'_v = \mathcal{E}_u$(FLOATTOINTBITS($s'_v$); $r'_v$) then
25:                 REPLY($m$, NOISE_CONFIRM, $s'_v$)
26:                 COMPUTEFINALNOISEWITH($v$)
27:             else
28:                 Blacklist $v$, report its fraud, etc.
29:         else if ($m$.type = NOISE_CONFIRM) $\wedge$ ($state_v$ = GUEST) $\wedge$ ($m$.replyTo = $lastSent_v$.id) then
30:             COMPUTEFINALNOISEWITH($v$)
31: procedure REQUESTNEIGHBORSFORNOISE()
32:     for all $v \in \mathcal{N}_u$ do
33:         if $state_v$ = READY then
34:             $s_v \leftarrow$ DRAW($\mathcal{U}(0,1)$); $r_v \leftarrow$ DRAW($\mathbb{Z}^*_{n_u}$)
35:             $lastSent_v \leftarrow$ SEND($v$, NOISE_REQUEST, $\mathcal{E}_u$(FLOATTOINTBITS($s_v$); $r_v$))
36:             $state_v \leftarrow$ GUEST
37:         else if ($state_v \neq$ VALID) $\wedge$ (CURRENTTIME() $-$ $lastSent_v$.sendDate > $T_R$) then /* No answer
    in due time */
38:             $state_v \leftarrow$ READY
39:             /* If $v$ is assumed not reliable, $u$ may also replace him with another user */
40: function FLOATTOINTBITS($f$: Float): PositiveInteger
41:     $(b_1, \dots, b_n) \leftarrow$ bits of $f$
42:     return $\sum_{i=1}^{n} b_i \cdot 2^{i-1}$
43: procedure COMPUTEFINALNOISEWITH($v$: User)
44:     $\delta_v \leftarrow \mathrm{sgn}(s_v - s'_v)\, F_\delta^{-1}((s_v + s'_v) - \lfloor s_v + s'_v \rfloor)$
45:     $state_v \leftarrow$ VALID
46:     PUBLISHFINALNOISEWITH($v$, $\mathcal{E}_u(\delta_v)$) /* For the verification phase */
```

Algorithm 9 Asynchronous gossip averaging from user u's first person view

```
 1: function AVERAGEGOSSIP($T_R$, $T_P$: Duration): Real
Require: $T_R < T_P$
 2:     $state \leftarrow$ READY; $M \leftarrow [\,]$; $i \leftarrow 0$; $a[0] \leftarrow \tilde{x}_u$; $vAvg \leftarrow$ NULL
 3:     for all $v \in \mathcal{N}_u$ do
 4:         $i_v \leftarrow 0$; $a_v[0] \leftarrow \infty$
 5:     while $\neg$AVERAGECONSENSUS() do
 6:         $M \leftarrow$ FILTERAVERAGEMESSAGES($M$)
 7:         if $M$.SIZE() $> 0$ then
 8:             for all $m \in M$ do
 9:                 if PROCESSAVERAGEMESSAGE($m$) then
10:                     $M$.REMOVE($m$)
11:                     break
12:         else if $state$ = READY then
13:             REQUESTAVERAGE()
14:         if CURRENTTIME() $-$ $lastSent$.sendDate $> T_R$ then /* Reply timeout is reached */
15:             $state \leftarrow$ READY
16:     return $a[i]$
17: function AVERAGECONSENSUS(): Boolean /* neighborhood consensus only */
18:     return (CURRENTTIME() $-$ $lastSent$.sendDate $> T_P$) $\wedge$ ($\forall v \in \mathcal{N}_u \setminus \{vAvg\}, a[i] = a_v[i_v]$)
19: function FILTERAVERAGEMESSAGES($M$: List<Message>): List<Message>
20:     for all $m \in$ GETNEWMESSAGES do
21:         if $m$.type = AVERAGE_REQUEST then /* Keep only the last request from $m$.sender */
22:             for all $m' \in M$ do
23:                 if ($m'$.type = AVERAGE_REQUEST) $\wedge$ ($m'$.sender = $m$.sender) then
24:                     $M$.REMOVE($m'$)
25:             $M$.ADD($m$)
26:         else if ($m$.type $\in$ {AVERAGE_RESPONSE, AVERAGE_CONFIRM}) $\wedge$ ($m$.sender = $vAvg$) then
27:             $M$.INSERT(0, $m$) /* Responses and confirmations are high-priority */
28:     return $M$
```

Continued below…

... continuation of Algorithm 9

```
29: function processAverageMessage($m$: Message): Boolean
30:     if $m$.type = AVERAGE_REQUEST then
31:         $a_{vAvg}[i_{vAvg}] \leftarrow m$.data
32:         if $state$ = READY then
33:             $vAvg \leftarrow m$.sender
34:             $lastSent \leftarrow$ reply($m$, AVERAGE_RESPONSE, $a[i]$)
35:             $state \leftarrow$ GUEST
36:             return $\top$
37:         else if ($state$ = INITIATOR) $\wedge$ ($m$.sender = $vAvg$) $\wedge$ ($lastSent$.type = AVERAGE_REQUEST) then
    /* Average requests crossed each others */
38:             reply($m$, AVERAGE_CONFIRM, $m$.data)
39:             return $\top$
40:     else if ($m$.type = AVERAGE_RESPONSE) $\wedge$ ($state$ = INITIATOR) $\wedge$ ($m$.sender = $vAvg$) $\wedge$ ($m$.replyTo = $lastSent$.id) $\wedge$ ($lastSent$.type = AVERAGE_REQUEST) then
41:         $lastSent \leftarrow$ reply($m$, AVERAGE_CONFIRM, $m$.data)
42:         averageWith($vAvg$)
43:         return $\top$
44:     else if ($m$.type = AVERAGE_CONFIRM) $\wedge$ ($m$.sender = $vAvg$) $\wedge$ ($m$.replyTo = $lastSent$.id) $\wedge$ ($state \in$ {INITIATOR, GUEST}) then
45:         averageWith($vAvg$)
46:         return $\top$
47:     else
48:         return $\bot$
49: procedure requestAverage()
50:     $v \leftarrow$ draw($\{v' \in \mathcal{N}_u : a[i] \neq a_{v'}[i_{v'}]\}$)
51:     $lastSent \leftarrow$ send($v$, AVERAGE_REQUEST, $a[i]$)
52:     $state \leftarrow$ INITIATOR
53: procedure averageWith($v$: User)
54:     $a(i+1) \leftarrow \frac{1}{2}\left(a(i) + a_v(i_v)\right)$
55:     $i \leftarrow i + 1$; $i_v \leftarrow i_v + 1$
56:     $state \leftarrow$ READY; $vAvg \leftarrow$ NULL
```

Appendix B

Reference paper on GOPA[1]

Hiding in the Crowd: A Massively Distributed Algorithm for Private Averaging with Malicious Adversaries

Pierre Dellenbach[1], Aurélien Bellet*[1], and Jan Ramon[1]

[1]INRIA

Abstract

The amount of personal data collected in our everyday interactions with connected devices offers great opportunities for innovative services fueled by machine learning, as well as raises serious concerns for the privacy of individuals. In this paper, we propose a massively distributed protocol for a large set of users to privately compute averages over their joint data, which can then be used to learn predictive models. Our protocol can find a solution of arbitrary accuracy, does not rely on a third party and preserves the privacy of users throughout the execution in both the honest-but-curious and malicious adversary models. Specifically, we prove that the information observed by the adversary (the set of maliciours users) does not significantly reduce the uncertainty in its prediction of private values compared to its prior belief. The level of privacy protection depends on a quantity related to the Laplacian matrix of the network graph and generally improves with the size of the graph. Furthermore, we design a verification procedure which offers protection against malicious users joining the service with the goal of manipulating the outcome of the algorithm.

1 Introduction

Through browsing the web, engaging in online social networks and interacting with connected devices, we are producing ever growing amounts of sensitive personal data. This has fueled the massive development of innovative personalized services which extract value from users' data using machine learning techniques. In today's dominant approach, users hand over their personal data to the service provider, who stores everything on centralized or tightly coupled systems hosted in data centers. Unfortunately, this poses important risks regarding the privacy of users. To mitigate these risks, some approaches have been proposed to learn from datasets owned by several parties who do not want to disclose their data. However, they typically suffer from some drawbacks: (partially) homomorphic encryption schemes (Paillier, 1999; Graepel et al., 2012; Aslett et al., 2015) require the existence of a trusted third party, secure multi-party computation techniques (Yao, 1982; Lindell and Pinkas, 2009) are generally intractable when the number of parties is large, and exchanging noisy sketches of the data through (local) differential privacy (Dwork, 2006; Duchi et al., 2012) only provides approximate solutions which are quite inaccurate in the highly distributed setting considered here. Furthermore, many of these techniques are not robust to the presence of malicious parties who may try to manipulate the outcome of the algorithm.

In this paper, our goal is to design a massively distributed protocol to collaboratively compute averages over the data of thousands to millions of users (some of them honest-but-curious and some corrupted by a malicious party), with arbitrary accuracy and in a way that preserves their privacy. For machine learning algorithms whose sufficient statistics are averages (e.g., kernel-based algorithms in primal space and decision trees), this could be used as a primitive to privately learn more complex models. The approach we propose is fully decentralized: users keep their own data locally and exchange information asynchronously over a

*Corresponding author: `first.last@inria.fr`

[1]without supplementary material

peer-to-peer network (represented as a connected graph), without relying on any third party. Our algorithm (called GOPA: GOssip for Private Averaging) draws inspiration from a randomized gossip protocol for averaging (Boyd et al., 2006), augmented with a first phase involving pairwise exchanges of noise terms so as to mask the private values of users without affecting the global average. We first analyze the correctness of the algorithm, showing that the addition of noise has a mild effect on the convergence rate. We then study the privacy guarantees of GOPA in a Bayesian framework, where the adversary has some prior belief about the private values. Specifically, we give an exact expression for the posterior variance of the adversary after he has observed all the information output by the protocol, and show that the variance loss is negligible compared to the prior. This is equivalent to showing that the uncertainty in the adversary's predictions of the private values has not been significantly reduced. Interestingly, the proportion of preserved variance depends on the variance of the noise used to mask the values but also on an interpretable quantity related to the Laplacian matrix of the network graph. To the best of our knowledge, we are the first to draw a link between privacy and a graph smoothing operator popular in semi-supervised learning (Zhu and Ghahramani, 2002; Zhou et al., 2003), multi-task learning Evgeniou and Pontil (2004) and signal processing (Shuman et al., 2013). We show how this result motivates the use of a random graph model to construct the network graph so as to guarantee strong privacy even under rather large proportions of malicious users, as long as the number of users is big enough. The practical behavior of GOPA is illustrated on some numerical simulations. Finally, we further enhance our protocol with a verification procedure where users are asked to publish some values in encrypted form, so that cheaters trying to manipulate the output of the algorithm can be detected with high probability while preserving the aforementioned privacy guarantees.

The rest of this paper is organized as follows. Section 2 describes the problem setting, including our adversary and privacy models. Section 3 presents some background on (private) decentralized averaging and partially homomorphic encryption, along with related work and baseline approaches. Section 4 introduces the GOPA algorithm and studies its convergence qs well as privacy guarantees. Section 5 describes our verification procedure to detect cheaters. Finally, Section 6 displays some numerical simulations. Proofs can be found in the supplementary material.

2 Preliminaries

We consider a set $U = \{1, \ldots, n\}$ of $n \geq 3$ users. Each user $u \in U$ holds a personal value $X_u \in \mathbb{R}$, which can be thought of as the output of some function applied to the personal data of u (e.g., a feature vector describing u). The users want to collaboratively compute the average value $X^{avg} = \frac{1}{n} \sum_{u=1}^{n} X_u$ while keeping their personal value private. Such a protocol could serve as a building block for privately running machine learning algorithms which interact with the data only through averages, such as linear regression models (ordinary least-squares, ridge regression), decision trees and gradient descent for empirical risk minimization problems. We denote by X the vector $X = [X_1, \ldots, X_n]^\top \in \mathbb{R}^n$.

Instead of relying on a central server or any third party to exchange information, users communicate over a peer-to-peer network represented by a connected undirected graph $G = (U, E)$, where $(u, v) \in E$ indicates that users u and v are neighbors in G and can exchange messages directly. For a given user u, we denote by $N(u) = \{v : (u, v) \in E\}$ the set of its neighbors. We denote by A the adjacency matrix of G, by $d = (d_1, \ldots, d_n)$ the degree vector ($d_u = \sum_v A_{uv}$) and by $L = \text{diag}(d) - A$ its Laplacian matrix.

2.1 Adversary Models

We consider two commonly adopted adversary models for users, which were formalized by Goldreich (1998) and are used in the design of many secure protocols. An *honest-but-curious* (*honest* for short) user will follow the protocol specification, but can use all the information received during the execution to infer information about other users. In contrast, a *malicious user* may deviate from the protocol execution by sending incorrect values at any point (but we assume that they follow the required communication policy; if not, this can be easily detected). Malicious users can collude, and thus will be seen as a single malicious party who has access to all information collected by malicious users. We only restrict the power of attackers by requiring that honest users communicate through secure channels, which means that malicious users only observe information during communications they take part in.

Each user in the network is either honest or malicious, and honest users do not know whether other nodes are honest or malicious. We denote by $U^H \subseteq U$ the set of honest users and by $f = 1 - |U^H|/n$ the proportion of malicious users in the network. We also denote by $G^H = (U^H, E^H)$ the subgraph of G induced by U^H, so that $E^H = \{(u, v) \in E : u, v \in U^H\}$. Throughout the paper, we will rely on the following natural assumption on G^H (we will discuss how to generate G such that it holds in practice in Section 4.4).

Assumption 1. *The graph of honest users G^H is connected.*

This implies that there exists a path between any two honest users in the full graph G which does not go through a malicious node. In the rest of the paper, we will use the term *adversary* to refer to the set of malicious users (privacy with respect to a honest user can be obtained as a special case).

2.2 Privacy Model

Recall that our goal is to design a protocol which deterministically outputs the *exact* average (which we argue does not reveal much information about individual values in the large-scale setting we consider). This requirement automatically rules out Differential Privacy (Dwork, 2006) as the latter implies that the output of the protocol has to be randomized.

We take a Bayesian, semantic view of privacy promoted by several recent papers (Kasiviswanathan and Smith, 2014; Li et al., 2013; Kifer and Machanavajjhala, 2014; He et al., 2014). We consider a family of prior distributions which represent the potential background knowledge that the adversary may have about the private values of honest users (since the adversary controls the malicious users, we consider he has exact knowledge of their values). We will denote by $P(X_u)$ the prior belief of the adversary about the private value of some honest user $u \in U^H$. Given all the information $\mathcal{I}$ gathered by the adversary during the execution of the protocol, the privacy notion we consider is that the ratio of prior and posterior variance of the private value X_u is lower bounded by $1 - \epsilon$ for some $\epsilon \in [0, 1]$:

$$\frac{var(X_u \mid \mathcal{I})}{var(X_u)} \geq 1 - \epsilon \tag{1}$$

The case $\epsilon = 1$ describes the extreme case where observing $\mathcal{I}$ removed all uncertainty about X_u. On the other hand, when $\epsilon = 0$, all variance was preserved.

It is important to note that (1) with ϵ close to 0 does not guarantee that the adversary learns almost nothing from the output $\mathcal{I}$: in particular, the posterior expectation $\mathbb{E}[X_u \mid \mathcal{I}]$ can largely differ from the prior expectation $\mathbb{E}[X_u]$, especially if the observed global average was very unlikely under the prior belief. This is related to the "no free lunch" theorem in data privacy (Kifer and Machanavajjhala, 2011), see also the discussions in Kasiviswanathan and Smith (2014); Li et al. (2013). What (1) does guarantee, however, is that the squared error that the adversary expects to make by predicting the value of some private X_u after observing $\mathcal{I}$ is almost the same as it was before observing $\mathcal{I}$. In other words, the *uncertainty* in its prediction has not been significantly reduced by the participation to the protocol. This is formalized by the following remark.

Remark 1 (Expected squared error). *Assume that* (1) *is satisfied. Let $Y_u = \mathbb{E}[X_u]$ be the best prediction (in terms of expected square error) that the adversary can make given its prior belief. After observing the output $\mathcal{I}$, the adversary can make a new best guess $Y_u^{\mathcal{I}} = \mathbb{E}[X_u \mid \mathcal{I}]$. We have:*

$$\frac{\mathbb{E}[(Y_u^{\mathcal{I}} - X_u)^2]}{\mathbb{E}[(Y_u - X_u)^2]} = \frac{var(X_u \mid \mathcal{I})}{var(X_u)} = 1 - \epsilon.$$

Our results will be valid for Gaussian prior distributions of the form $X_u \sim \mathcal{N}(0, \sigma_X^2)$, for all $\sigma_X^2 > 0$.[1] We assume for simplicity that the prior variance is the same for all u, but our analysis straightforwardly extends

[1]We use Gaussian distributions for technical reasons. We expect similar results to hold for other families of distributions which behave nicely under conditioning and linear transformations, such as the exponential family.

Algorithm 1 Randomized gossip (Boyd et al., 2006)

```
1: Input: graph $G = (U, E)$, initial values $(X_u(0))_{u \in U}$
2: for $t \in \{1, 2, ...\}$ do
3:     Draw $(u, v)$ uniformly at random from $E$
4:     Set $X_u(t), X_v(t) \leftarrow \frac{1}{2}(X_u(t-1) + X_v(t-1))$
5: end for
```

to the more general case where $X_u \sim \mathcal{N}(0, \sigma^2_{X_u})$. We use centered Gaussians without loss of generality, since (1) depends only on the variance.

Remark 2 (Privacy axioms)**.** *One can show that our notion of privacy* (1) *satisfies the axioms that any reliable privacy definition should satisfy according to Kifer and Lin (2012), namely "transformation invariance" and "convexity".*

3 Background and Related Work

3.1 Decentralized Averaging

The problem of computing the average value X^{avg} of a set of users in a fully decentralized network without a central coordinator has been extensively studied (see for instance Tsitsiklis, 1984; Kempe et al., 2003; Boyd et al., 2006). In most existing approaches, users iteratively compute the weighted average between their value and those of (a subset of) their neighbors in the network. We focus here on the randomized gossip algorithm proposed by Boyd et al. (2006) as it is simple, asynchronous and exhibits fast convergence. Each user has a clock ticking at the times of a rate 1 Poisson process. When the clock of an user u ticks, it chooses a random neighbor v and they average their current value. As discussed in Boyd et al. (2006), one can equivalently consider a single clock ticking at the times of a rate n Poisson process and a random edge $(u, v) \in E$ drawn at each iteration. The procedure is shown in Algorithm 1, and one can show that all users converge to X^{avg} at a geometric rate.

Although there is no central server collecting all users' data, the above gossip algorithm is not private since users must share their inputs directly with others. A way to ensure privacy is that each user locally perturbs its own input before starting the algorithm so as to satisfy local differential privacy (Duchi et al., 2012; Kairouz et al., 2016).

Baseline 1. *Assume $|X_u| \leq B_X$ for some finite $B_X \in \mathbb{R}$. Each user u computes a perturbed value $\widetilde{X}_u = X_u + \eta_u$, where η_u is a noise value with Laplacian distribution $P(\eta_u) = \exp(-|\eta_u|/b)/2b$ with $b = 2B/\epsilon$. This guarantees that $\widetilde{X} = [\widetilde{X}_1, \ldots, \widetilde{X}_n]^T$ is an ϵ-differentially private approximation of X (see e.g., Dwork, 2008), hence running Algorithm 1 on $\widetilde{X}$ is also ϵ-differentially private.*

Unfortunately, this protocol converges to an approximate average $\hat{X}^{avg} = \sum_{u=1}^n (X_u + \eta_u)/n$, and the associated RMSE $\sqrt{\mathbb{E}[(X^{avg} - \hat{X}^{avg})^2]} = b\sqrt{2/n}$ can be high even for a large number of users. For instance, for $n = 10^4$ users, $\epsilon = 0.1$ and $B = 0.5$, the RMSE is approximately 0.14.

Other attempts have been made at designing privacy-preserving decentralized averaging protocols under various privacy and attack models, and additional assumptions on the network topology. For an adversary able to observe all communications in the network, Huang et al. (2012) proposed an ϵ-differentially private protocol where users add exponentially decaying Laplacian noise to their value. The protocol converges with probability p to a solution with an error radius of order $O(1/\epsilon\sqrt{(1-p)n})$. Manitara and Hadjicostis (2013) instead proposed that each user iteratively adds a finite sequence of noise terms which sum to zero, so that convergence to the exact average can be achieved. The adversary consists of a set of curious nodes which follow the protocol but can share information between themselves. Under some assumption on the position of the curious nodes in the network, their results prevent *perfect* recovery of private values but do not bound the estimation accuracy that the adversary can achieve. Mo and Murray (2014) proposed to add and subtract exponentially decaying Gaussian noise, also ensuring convergence to the exact average. They consider only honest-but-curious nodes and rely on a strong topological assumption, namely that there is no node whose neighborhood includes another node and its entire neighborhood. Their privacy guarantees are in the form of lower bounds on the covariance matrix of the maximum likelihood estimate of the private inputs that any node can make, which are difficult to interpret and very loose in practice. Finally, Hanzely et al. (2017) introduced variants of Algorithm 1 which intuitively leak less information about the private inputs, but do not necessarily converge to the exact average. Importantly, they do prove any formal privacy guarantee.

In the above approaches, privacy is typically achieved by asking each user to independently add decaying local noise to its private value, which results in accuracy loss and/or weak privacy guarantees. In contrast, the protocol we propose in this paper relies on *sharing* zero-sum noise terms across users so as to "dilute" the knowledge of private values into the crowd. This will allow a flexible trade-off between privacy and convergence rate even when a node has a large proportion of malicious neighbors, while ensuring the convergence to the exact average. Furthermore and unlike all the above approaches, we provide a verification procedure to prevent malicious nodes from manipulating the output of the algorithm. This verification procedure relies on partially homomorphic encryption, which we briefly present below.

3.2 Partially Homomorphic Encryption

A standard technique for secure computation is to rely on a partially homomorphic encryption scheme to cipher the values while allowing certain operations to be carried out directly on the cypher text (without decrypting).

We use the popular Paillier cryptosystem (Paillier, 1999), which is additive. Formally, one generates a public (encryption) key $K^{pub} = (N, g)$, where $N = pq$ for $p \neq q$ two independent large primes and $g \in \mathbb{Z}^*_{N^2}$, and a secret (decryption) key $K^{priv} = \operatorname{lcm}(p-1, q-1)$. A message $m \in \mathbb{Z}^*_N$ can then be encrypted into a cypher text with

$$\varepsilon(m) = g^m \cdot r^N \bmod N^2, \tag{2}$$

where r is drawn randomly from $\mathbb{Z}^*_N$. With knowledge of the secret key, one can recover m from $\varepsilon(m)$ based on the fact that $\frac{\varepsilon(m)^\lambda - 1}{g^\lambda - 1} \equiv m \bmod N$. Denote the decryption operation by ε^{-1}. Paillier satisfy the following homomorphic property for any $m_1, m_2 \in \mathbb{Z}^*_N$:

$$\varepsilon^{-1}(\varepsilon(m_1) \cdot \varepsilon(m_2) \bmod N^2) = m_1 + m_2 \bmod N, \tag{3}$$

hence $\varepsilon(m_1) \cdot \varepsilon(m_2)$ is a valid encryption of $m_1 + m_2$. For the purpose of this paper, we will consider the Paillier encryption scheme as perfectly secure (i.e., the computational complexity needed to break the encryption is beyond reach of any party). We can use the Paillier scheme to design a second simple baseline for private averaging.

Baseline 2. *Consider $X_1, \dots, X_n \in \mathbb{Z}^*_N$ and assume that users trust two central honest-but-curious entities: the server and the third party. The server generates a Paillier encryption scheme (K^{pub}, K^{priv}) and broadcasts K^{pub} to the set of users. Each user u computes $\varepsilon(X_u)$ and sends it to the third party. Following* (3)*, the third party then computes $\prod_{u=1}^n \varepsilon(X_u)$ and sends it to the server, which can obtain X^{avg} by decrypting the message (and dividing by n).*

If the server and the third party are indeed honest, nobody observes any useful information except the outcome X^{avg}. But if they are not honest, various breaches can occur. For instance, if the third party is malicious, it can send an incorrect output to the server. The third party could also send the encrypted values of some users to the server, which can decrypt them using the private key K^{priv}. Similarly, if the server is malicious, it can send K^{priv} to the third party which can then decrypt all of the users' private values.

In contrast, we will design a protocol which eliminates the need for such central trusted entities and instead distributes the trust across many users in the network (Section 4). We will however rely on homomorphic encryption to detect potential malicious users (Section 5).

Algorithm 2 Randomization phase of GOPA

```
1: Input: graph $G = (U, E)$, private values $(X_u)_{u \in U}$, distribution $\mu : \mathbb{R} \to [0, 1]$
2: for all neighbor pairs $(u, v) \in E$ s.t. $u < v$ do
3:     $u$ and $v$ jointly draw a random number $\delta \in \mathbb{R}$ from $\mu$
4:     $\delta_{u,v} \leftarrow \delta$, $\delta_{v,u} \leftarrow -\delta$
5: end for
6: for all users $u \in U$ do
7:     $\Delta_u \leftarrow \sum_{v \in N(u)} \delta_{u,v}$, $\widetilde{X}_u \leftarrow X_u + \Delta_u$
8: end for
9: Output: noisy values $(\widetilde{X}_u)_{u \in U}$
```

4 GOPA: Private Gossip Averaging Protocol

4.1 Protocol Description

We describe our GOPA protocol (GOssip for Private Averaging), which works in two phases. In the first phase (*randomization phase*), users mask their private value by adding noise terms that are correlated with their neighbors, so that the global average remains unchanged. In the second phase (*averaging phase*), users average their noisy values. For simplicity of explanation, we abstract away the communication mechanisms, i.e., contacting a neighbor and exchanging noise are considered as atomic operations.

Randomization phase. Algorithm 2 describes the first phase of GOPA, during which all neighboring nodes $(u, v) \in E$ contact each other to exchange noise values. Specifically, they jointly draw a random real number δ from a probability distribution μ (a parameter of the algorithm that the community agrees on), that u will add to its private value and v will subtract. Following the common saying "don't put all your eggs in one basket", each user thereby distributes the noise masking his private value across several other users (his direct neighbors but also beyond by transitivity), which will provide some robustness to malicious parties. The idea is reminiscent of one-time pads (see for instance Bonawitz et al., 2017, Section 3 therein), but our subsequent analysis will highlight the key role of the network topology and show that we can tune the magnitude of the noise so as to trade-off between privacy on the one hand, and convergence speed as well as the impact of user drop out on the other hand. The result of this randomization phase is a set of noisy values $\widetilde{X} = [\widetilde{X}_1, \dots, \widetilde{X}_n]^\top$ for each user, with the same average value X^{avg} as the private values. Note that each user u exchanges noise exactly once with each of his neighbors, hence the noisy value $\widetilde{X}_u$ consists of d_u noise terms.

Averaging phase. In the second phase, the users start from their noisy values $\widetilde{X}$ obtained in the randomization phase and simply run the standard randomized gossip averaging algorithm (Algorithm 1).

4.2 Correctness and Convergence of GOPA

In this section, we study the correctness and convergence rate of GOPA. Let $\widetilde{X}(t) = [\widetilde{X}_1(t), \dots, \widetilde{X}_n(t)]^\top$ be the values after $t \geq 0$ iterations of the averaging phase (Algorithm 1) initialized with the noisy values $\widetilde{X}$ generated by the randomization phrase (Algorithm 2). Following the seminal work of Boyd et al. (2006), we will measure the convergence rate in terms of the τ-*averaging time.* Given $\tau \in (0, 1)$, the τ-averaging time is the number of iterations $t(\tau)$ needed to guarantee that for any $t \geq t(\tau)$:

$$\Pr(\|\widetilde{X}(t) - X^{avg}\mathbf{1}\| / \|X\| \geq \tau) \leq \tau. \tag{4}$$

Note that the error in (4) is taken relatively to the original set of values X to account for the impact of the addition of noise on the convergence rate. We have the following result for the case where all users are honest (we will lift this requirement in Section 5).

Proposition 1 (Correctness and convergence rate)**.** *When all users are honest, the sequence of iterates* $(\widetilde{X}(t))_{t\geq 0}$ *generated by* GOPA *satisfies* $\lim_{t\to+\infty}\mathbb{E}[\widetilde{X}(t)] = X^{avg}\mathbf{1}$*. Furthermore, the* τ*-averaging time of Algorithm 1 is:*

$$t(\tau) = 3\log\Big(\frac{2B_\delta(d_{max}+3)}{\tau B_X}\Big)\Big/\log(1/C_G), \tag{5}$$

where $C_G = 1-\lambda_2(L)/|E| \in (0,1)$ *with* $\lambda_2(L)$ *the second smallest eigenvalue of the Laplacian matrix L (Boyd et al., 2006; Colin et al., 2015),* $d_{max} = \max_u d_u$ *is the maximum degree and* B_X, B_δ *are upper bounds for the absolute value of private values and noise terms respectively.*[2]

Proposition 1 allows to quantify the *worst-case* impact of the randomization phase on the convergence of the averaging phase. The τ-averaging time of GOPA is only increased by a constant additive factor compared to the non-private averaging phase. Importantly, this additive factor has a mild (logarithmic) dependence on B and d_{max}. This behavior will be confirmed by numerical experiments in Section 6.

4.3 Privacy Guarantees

We now study the privacy guarantees of GOPA. We consider that the knowledge $\mathcal{I}$ acquired by the adversary (colluding malicious users) during the execution of the protocol contains the following: (i) the noisy values $\widetilde{X}$ of all users at the end of the randomization phase, (ii) the full network graph (and hence which pairs of honest users exchanged noise), and (iii) for any communication involving a malicious party, the noise value used during this communication. The only unknowns are the private values of honest users, and noise values exchanged between them. Note that (i) implies that the adversary learns the network average.

Remark 3. *Since we assume that the noisy values* $\widetilde{X} = [\widetilde{X}_1,\dots,\widetilde{X}_n]^\top$ *are known to the adversary, our privacy guarantees will hold even if these values are publicly released. Note however that computing the average with Algorithm 1 provides additional protection as the adversary will observe only a subset of the noisy values of honest nodes (those who communicate with a malicious user at their first iteration). In fact, inferring whether a received communication is the first made by that user is already challenging due to the asynchronous nature of the algorithm.*

Our main result exactly quantifies how much variance is left in any private value X_u after the adversary has observed all information in $\mathcal{I}$.

Theorem 1 (Privacy guarantees)**.** *Assume that the prior belief on the private values* $X_1,\dots,X_n$ *is Gaussian, namely* $X_u \sim \mathcal{N}(0,\sigma_X^2)$ *for all honest users* $u \in U^H$*, and that the noise variables are also drawn from a Gaussian distribution* $\mu = \mathcal{N}(0,\sigma_\delta^2)$*. Denote by* $e_u \in \mathbb{R}^n$ *the indicator vector of the u-th coordinate and let* $M = (I + \frac{\sigma_\delta^2}{\sigma_X^2}L^H)^{-1} \in \mathbb{R}^{n\times n}$*, where* L^H *is the Laplacian matrix of the graph* G^H*. Then we have for all honest user* $u^H \in U$*:*

$$\frac{var(X_u \mid \mathcal{I})}{var(X_u)} = 1 - e_u^\top M e_u. \tag{6}$$

We now provide an detailed interpretation of the above result. First note that the proportion of preserved variance only depends on the interactions between honest users, making the guarantees robust to any adversarial values that malicious users may send. Furthermore, notice that matrix M can be seen as a smoothing operator over the graph G^H. Indeed, given some $y \in \mathbb{R}^n$, $My \in \mathbb{R}^n$ is the solution to the following optimization problem:

$$\min_{s\in\mathbb{R}^n} \|s-y\|_2^2 + \alpha s^\top L^H s, \tag{7}$$

where $\alpha = \sigma_\delta^2/\sigma_X^2 \geq 0$. This problem is known as *graph smoothing* (also graph regularization, or graph filtering) and has been used in the context of semi-supervised learning (see Zhu and Ghahramani, 2002; Zhou et al., 2003), multi-task learning (Evgeniou and Pontil, 2004) and signal processing on graphs (Shuman et al.,

[2]We use a bounded noise assumption for simplicity. The argument easily extends to boundedness with high probability.

2013), among others. The first term in (7) encourages solutions that are close to y while the second term is the Laplacian quadratic form $s^\top L^H s = \sum_{(u,v)\in E^H}(s_u - s_v)^2$ which enforces solutions that are smooth over the graph (the larger α, the more smoothing). One can show that M is a doubly stochastic matrix (Segarra et al., 2015), hence the vector My sums to the same quantity as the original vector y. In our context, we smooth the indicator vector e_u so $e_u^\top M e_u \in [0,1]$: the larger the noise variance σ_δ^2 and the more densely connected the graph of honest neighbors, the more "mass" is propagated from user u to other nodes, hence the closer to 0 the value $e_u^\top M e_u$ and in turn the more variance of the private value X_u is preserved. Note that we recover the expected behavior in the two extreme cases: when $\sigma_\delta^2 = 0$ we have $e_u^\top M e_u = 1$ and hence the variance ratio is 0, while when $\sigma_\delta^2 \to \infty$ we have $e_u^\top M e_u \to 1/|U_H|$ and hence $var(X_u \mid \mathcal{I})/var(X_u) = 1 - 1/|U_H|$. This irreducible variance loss accounts for the fact that the adversary learns from $\mathcal{I}$ the average over the set of honest users (by adding their noisy values and subtracting the noise terms he knows). Since we assume the number of users to be very large, this variance loss can be considered negligible. We emphasize that in contrast to the lower bounds on the variance of the maximum likelihood estimate obtained by Mo and Murray (2014), our variance computation (6) is exact, interpretable and holds under non-uniform prior beliefs of the adversary.

It is important to note that smoothing occurs over the entire graph G^H: by transitivity, all honest users in the network contribute to keeping the private values safe. In other words, Gopa turns the massively distributed setting into an advantage for privacy, as we illustrate numerically in Section 6. Note, still, that one can derive a simple (but often loose) lower bound on the preserved variance which depends only on the local neighborhood.

Proposition 2. *Let $u \in U^H$ and denote by $|N^H(u)|$ the number of honest neighbors of u. We have:*

$$var(X_u \mid \mathcal{I}) \geq \sigma_X^2 \cdot \frac{\sigma_\delta^2(|N^H(u)|+1)}{\sigma_X^2 + \sigma_\delta^2(|N^H(u)|+1)} \cdot \frac{|N^H(u)|}{|N^H(u)+1|}.$$

Proposition 2 shows that the more honest neighbors, the larger the preserved variance. In particular, if $\sigma_\delta^2 > 0$, we have $var(X_u \mid \mathcal{I}) \to \sigma_X^2$ as $|N^H(u)| \to \infty$.

We conclude this subsection with some remarks.

Remark 4 (Composition). *As can be seen from inspecting the proof of Theorem 1, $P(X_u \mid \mathcal{I})$ is a Gaussian distribution. This makes our analysis applicable to the setting where* Gopa *is run several times with the same private value X_u as input, for instance within an iterative machine learning algorithm. We can easily keep track of the preserved variance by recursively applying Theorem 1.*

Remark 5 (G^H not connected). *The results above still hold when Assumption 1 is not satisfied. In this case, the smoothing occurs separately within each connected component of G^H, and the irreducible variance loss is ruled by the size of the connected component of G^H that the user belongs to (instead of the total number of honest users $|U_H|$).*

Remark 6 (Drop out). *The use of centered Gaussian noise with bounded variance effectively limits the impact of some users dropping out during the randomization phase. In particular, any residual noise term has expected value 0, and can be bounded with high probability. Alternatively, one may ask each impacted user to remove the noise exchanged with users who have dropped (thereby ensuring exact convergence at the expense of reducing privacy guarantees).*

4.4 Robust Strategies for Network Construction

We have seen above that the convergence rate and most importantly the privacy guarantees of GOPA crucially depend on the network topology. In particular, it is crucial that the network graph $G = (U, E)$ is constructed in a robust manner to ensure good connectivity and to guarantee that all honest users have many honest neighbors with high probability. In the following, we assume users have an address list for the set of users and can select some of them as their neighbors. Note that the randomization phase can be conveniently executed when constructing the network.

Algorithm 3 Verification procedure for the randomization phase of GOPA

```
1: Input: each user $u$ has generated its own Paillier encryption scheme and has published:
2:     • Public key $K_u^{pub}$, $\varepsilon_u^P(X_u)$ (before the execution of Algorithm 2)
3:     • Noise values $\varepsilon_u^P(\delta_{u,v})$ (when exchanging with $v \in N(u)$ during Algorithm 2)
4:     • $\varepsilon_u^P(\widetilde{X}_u)$, $\varepsilon_u^P(\Delta_u)$ (at the end of Algorithm 2)
5: for all user $u \in U$ do
6:     $\varepsilon^{\Delta} \leftarrow \prod_{v \in N(u)} \varepsilon_u^P(\delta_{u,v})$, $\varepsilon^{\widetilde{X}} \leftarrow \varepsilon_u^P(X_u) \cdot \varepsilon_u^P(\Delta_u)$
7:     Verify that $\varepsilon_u^P(\Delta_u) = \varepsilon^{\Delta}$ and $\varepsilon_u^P(\widetilde{X}_u) = \varepsilon^{\widetilde{X}}$ (if not, add $u$ to cheater list)
8: end for
9: for all user $u \in U$ do
10:     Draw random subset $V_u$ of $N(u)$ with $|V_u| = \lceil (1-\beta) d_u \rceil$, ask $u$ to publish $\delta_{u,v}$ and $r_{\delta_{u,v}}$ for $v \in V_u$
11:     for all $v \in V_u$ do
12:         Ask $v$ to publish $r_{\delta_{v,u}}$, verify that $\varepsilon_v^P(\delta_{v,u}) = \varepsilon_v(-\delta_{u,v})$ and $\varepsilon_u^P(\delta_{u,v}) = \varepsilon_u(\delta_{u,v})$ (if not, add $u, v$ to
            cheater list)
13:     end for
14: end for
```

A simple choice of network topology is the complete graph, which is best in terms of privacy (since each honest user has all $|U_H| - 1$ other honest users as neighbors) and convergence rate (best connectivity). Yet this is not practical when n is very large: beyond the huge number of pairwise communications needed for the randomization phase, each user also needs to create and maintain $n-1$ secure connections, which is costly (Chan et al., 2003).

We propose instead a simple randomized procedure to construct a sparse network graph with the desired properties based on *random k-out random graphs* (Bollobás, 2001), also known as *random k-orientable graphs* (Fenner and Frieze, 1982). The idea is to make each (honest) user select k other users uniformly at random from the set of all users. Then, the edge $(u, v) \in E$ is created if u selected v or v selected u (or both). This procedure is the basis of a popular key predistribution scheme used to create secure peer-to-peer communication channels in distributed sensor networks (Chan et al., 2003). Fenner and Frieze (1982) show that for any $k \geq 2$, the probability that the graph is k-connected goes to 1 almost surely. This provides robustness against malicious users. Note also that the number of honest neighbors of a honest node follows a hypergeometric distribution and is tightly concentrated around its expected value $k(1-f)$, where f is the proportion of malicious users. It is worth noting that the probability that the graph is connected is actually very close to 1 even for small n and k. This is highlighted by the results of Yağan and Makowski (2013), who established a rather tight lower bound on the probability of connectivity of random k-out network graphs. For instance, the probability is guaranteed to be larger than 0.999 for $k = 2$ and $n = 50$. The algebraic connectivity $\lambda_2(L)$ is also large in practice for random k-out graphs, hence ensuring good convergence as per Proposition 1.

5 Verification Procedure

We have shown in Section 4.2 that GOPA converges to the appropriate average when users are assumed not to tamper with the protocol by sending incorrect values. In this section, we complement our algorithm with a verification procedure to detect malicious users who try to influence the output of the algorithm (we refer to them as *cheaters*). While it is impossible to force a user to give the "right" input to the algorithm (no one can force a person to answer honestly to a survey),[3] our goal is to make sure that given the input vector X, the protocol will either output X^{avg} or detect cheaters with high probability. Our approach is

[3]We can use range proofs (Camenisch et al., 2008) to check that each input value lies in an appropriate interval without revealing anything else about the value.

based on asking users to publish some (potentially encrypted) values during the execution of the protocol.[4] The published information should be publicly accessible so that anyone may verify the validity of the protocol (avoiding the need for a trusted verification entity), but should not threatens the privacy.

In order to allow public verification without compromising privacy, we will rely on the Paillier encryption scheme described in Section 3.2. For the purpose of this section, we assume that all quantities (private values and noise terms) are in $\mathbb{Z}_N$. This is not a strong restriction since with appropriate N and scaling one can represent real numbers from a given interval with any desired finite precision. Each user u generates its own Paillier scheme and publishes encrypted values using its encryption operation $\varepsilon_u(\cdot)$. These published cypher texts may not be truthful (when posted by a malicious user), hence we denote them by the superscript P to distinguish them from a valid encryption of a value (e.g., $\varepsilon_u^P(v)$ is the cypher text published by u for the quantity v).

Algorithm 3 describes our verification procedure for the randomization phase of Gopa (the averaging phase can be verified in a similar fashion). In a first step, we verify the coherence of the publications of each user u, namely that $\Delta_u = \sum_{v \in N(u)} \delta_{u,v}$ and $\widetilde{X}_u = X_u + \Delta_u$ are satisfied for the published cypher texts of these quantities. To allow reliable equality checks between cipher texts, some extra care is needed (see supplementary material for details). The second step is to verify that during a noise exchange, the user and his neighbor have indeed used opposite values as noise terms. However, each user has his own encryption scheme so one cannot meaningfully compare cypher texts published by two different users. To address this issue, we ask each user u to publish *in plain text* a random selected fraction $(1 - \beta) \in [0, 1]$ of his noise values $(\delta_{u,v})_{v \in N(u)}$ (the noise values to reveal are drawn publicly). The following result lower bounds the probability of catching a cheater.

Proposition 3 (Verification)**.** *Let C be the number of times a user cheated during the randomization phase. If we apply the verification procedure (Algorithm 3), then the probability that a cheater is detected is at least $1 - \beta^{2C}$.*

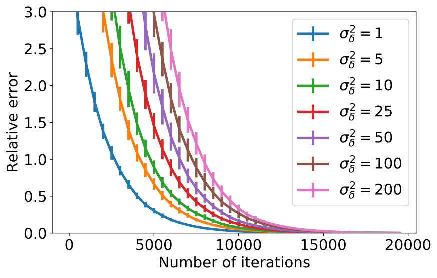
Proposition 3 shows that Algorithm 3 guards against large-scale cheating: the more cheating, the more likely at least one cheater gets caught. Small scale cheating is less likely to be detected but cannot affect much the final output of the algorithm (as values are bounded with high probability and the number of users is large). Of course, publishing a fraction of the noise values in plain text decreases the privacy guarantees: publishing $\delta_{u,v}$ in plain text for $u, v \in U^H$ corresponds to ignoring the associated edge of G^H in our privacy analysis. If the community agrees on β in advance, this effect can be easily compensated by constructing a network graph of larger degree, see Section 4.4.

6 Numerical Experiments

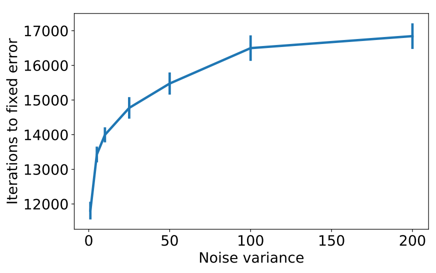
In this section, we run some simulations to illustrate the practical behavior of Gopa. In particular, we study two aspects: the impact of the noise variance on the convergence and on the proportion of preserved data variance, and the influence of network topology. In all experiments, the private values are drawn from the normal distribution $\mathcal{N}(0, 1)$, and the network is a random k-out graph (see Section 4.4).

Figure 1 illustrates the impact of the noise variance on the convergence of the averaging phase of Gopa, for $n = 1000$ users and $k = 10$. We see on Figure 1(a) that larger σ_δ^2 have a mild effect on the convergence rate. Figure 1(b) confirms that the number of iterations t needed to reach a fixed error $\|\widetilde{X}(t) - X^{avg}\mathbf{1}\| / \|X\|$ is logarithmic in σ_δ^2, as shown by our analysis (Proposition 1). Figure 2 shows the proportion of preserved data variance for several topologies and proportions of malicious nodes in the network. Recall that in random k-out graphs with fixed k, the average degree of each user is roughly equal to $2k$ and hence remains constant with n. The results clearly illustrate one of our key result: beyond the noise variance and the number of honest neighbors, the *total* number of honest users has a strong influence on how much variance is preserved. The more users, the more smoothing on the graph and hence the more privacy, without any additional cost

[4] We assume that users cannot change their published values after publication. This could be enforced by relying on blockchain structures as in Bitcoin transactions (Nakamoto, 2008).

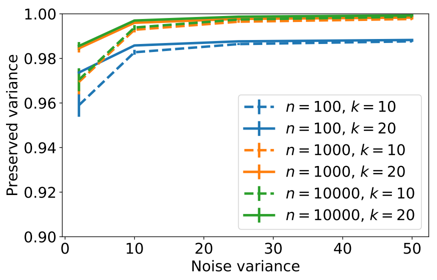


(a) Relative error w.r.t. # iter.

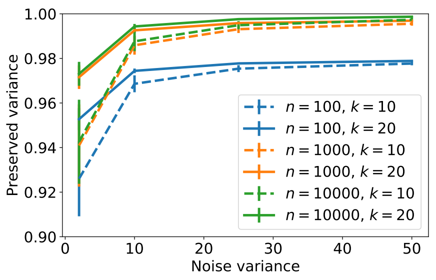


(b) # iter. to 10^{-2} error w.r.t. σ_δ^2

Figure 1: Impact of the noise variance on the convergence of Gopa for a random k-out graph with $n = 1000$ and $k = 10$, with mean and standard deviation over 10 random runs.



(a) 10% of malicious nodes



(b) 50% of malicious nodes

Figure 2: Preserved data variance (mean and standard deviation across users) w.r.t. noise variance for several random k-out topologies and two proportions of malicious nodes.

in terms of memory or computation for each individual user since the number of neighbors to interact with remains constant. This effect is even more striking when the proportion of malicious users is large, as in Figure 2(b). Remarkably, for n large enough, Gopa can effectively withstand attacks from many malicious users.

7 Conclusion

We proposed and analyzed a massively distributed protocol to privately compute averages over the values of many users, with robustness to malicious parties. Our novel privacy guarantees highlight the benefits of the large-scale setting, allowing users to effectively “hide in the crowd” by distributing the knowledge of their private value across many other parties. We believe this idea to be very promising and hope to extend its scope beyond the problem of averaging. In particular, we would like to study how our protocol may be used as a primitive to learn complex machine learning models in a privacy-preserving manner.

Acknowledgments This research was partially supported by grant ANR-16-CE23-0016-01 and by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020. The work was also partially supported by ERC-PoC SOM 713626.

References

Aslett, L. J. M., Esperança, P. M., and Holmes, C. C. (2015). Encrypted statistical machine learning: new privacy preserving methods. Technical report, arXiv:1508.06845.

Bollobás, B. (2001). *Random Graphs*. Cambridge University Press.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS*.

Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. (2006). Randomized Gossip Algorithms. *IEEE/ACM Transactions on Networking*, 14(SI):2508–2530.

Camenisch, J., Chaabouni, R., and Shelat, A. (2008). Efficient protocols for set membership and range proofs. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 234–252.

Chan, H., Perrig, A., and Song, D. X. (2003). Random Key Predistribution Schemes for Sensor Networks. In *S&P*.

Colin, I., Bellet, A., Salmon, J., and Clémençon, S. (2015). Extending Gossip Algorithms to Distributed Estimation of U-statistics. In *NIPS*.

Duchi, J. C., Jordan, M. I., and Wainwright, M. J. (2012). Privacy Aware Learning. In *NIPS*.

Dwork, C. (2006). Differential Privacy. In *ICALP*.

Dwork, C. (2008). Differential Privacy: A Survey of Results. In *TAMC*.

Evgeniou, T. and Pontil, M. (2004). Regularized multi-task learning. In *SIGKDD*.

Fenner, T. I. and Frieze, A. M. (1982). On the connectivity of random m-orientable graphs and digraphs. *Combinatorica*, 2(4):347–359.

Goldreich, O. (1998). Secure multi-party computation. *Manuscript. Preliminary version.*

Graepel, T., Lauter, K. E., and Naehrig, M. (2012). ML Confidential: Machine Learning on Encrypted Data. In *ICISC*.

Hanzely, F., Konečný, J., Loizou, N., Richtárik, P., and Grishchenko, D. (2017). Privacy Preserving Randomized Gossip Algorithms. Technical report, arXiv:1706.07636.

He, X., Machanavajjhala, A., and Ding, B. (2014). Blowfish privacy: tuning privacy-utility trade-offs using policies. In *SIGMOD*.

Huang, Z., Mitra, S., and Dullerud, G. (2012). Differentially private iterative synchronous consensus. In *ACM workshop on Privacy in the Electronic Society*.

Kairouz, P., Oh, S., and Viswanath, P. (2016). Extremal Mechanisms for Local Differential Privacy. *Journal of Machine Learning Research*, 17:1–51.

Kasiviswanathan, S. P. and Smith, A. (2014). On the 'Semantics' of Differential Privacy: A Bayesian Formulation. *Journal of Privacy and Confidentiality*, 6(1):1–16.

Kempe, D., Dobra, A., and Gehrke, J. (2003). Gossip-Based Computation of Aggregate Information. In *FOCS*.

Kifer, D. and Lin, B.-R. (2012). An Axiomatic View of Statistical Privacy and Utility. *Journal of Privacy and Confidentiality*, 4(1):5–46.

Kifer, D. and Machanavajjhala, A. (2011). No free lunch in data privacy. In *SIGKDD*.

Kifer, D. and Machanavajjhala, A. (2014). Pufferfish: A framework for mathematical privacy definitions. *ACM Transactions on Database Systems*, 39(1):3:1–3:36.

Li, N., Qardaji, W. H., Su, D., Wu, Y., and Yang, W. (2013). Membership privacy: a unifying framework for privacy definitions. In *CCS*.

Lindell, Y. and Pinkas, B. (2009). Secure Multiparty Computation for Privacy-Preserving Data Mining. *Journal of Privacy and Confidentiality*, 1(1):59–98.

Manitara, N. E. and Hadjicostis, C. N. (2013). Privacy-preserving asymptotic average consensus. In *ECC*.

Mo, Y. and Murray, R. M. (2014). Privacy preserving average consensus. In *CDC*.

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Available online at http://bitcoin.org/bitcoin.pdf.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*.

Segarra, S., Huang, W., and Ribeiro, A. (2015). Diffusion and Superposition Distances for Signals Supported on Networks. *IEEE Transactions on Signal and Information Processing over Networks*, 1(1):20–32.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98.

Tsitsiklis, J. N. (1984). *Problems in decentralized decision making and computation*. PhD thesis, Massachusetts Institute of Technology.

Yağan, O. and Makowski, A. M. (2013). On the Connectivity of Sensor Networks Under Random Pairwise Key Predistribution. *IEEE Transactions on Information Theory*, 59(9):5754–5762.

Yao, A. C. (1982). Protocols for secure computations. In *FOCS*.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2003). Learning with Local and Global Consistency. In *NIPS*.

Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, Carnegie Mellon University.

Abstract

GOPA is a privacy-preserving gossip algorithm for averaging that has applications in decentralized machine learning over P2P networks. This Master thesis aims to improve the robustness of the algorithm to churns and malicious adversaries. Thanks to the Paillier cryptosystem, commitment schemes and mathematical analysis, we have convincing theoretical results regarding both privacy aspects and correctness of averaging. Notable contributions are a framework for churn-tolerance and a privacy-preserving algorithm for detecting data injections in gossip averaging. In addition, we unify all our improvements around a robust user-centric version of GOPA. All our contributions have proved to be quite novel in comparison with the literature. We also have initiated the development of an Android prototype of GOPA so that our solutions can be experimented in the future. Finally, new theoretical and technical issues have been identified and may be treated as part of a next internship or PhD studies.

Keywords Gossip averaging; Decentralized protocols; Privacy; Churn tolerance; Byzantine attacks; Zero-knowledge proofs; Paillier homomorphic encryption