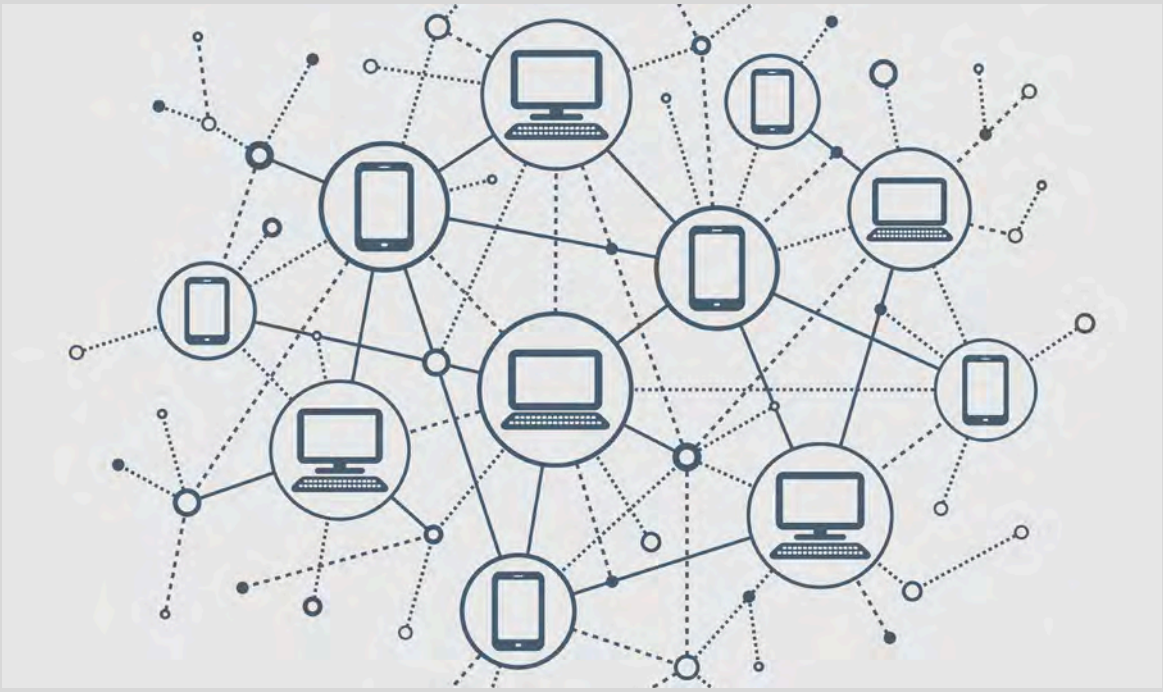
UNIVERSITAT ROVIRA i VIRGILI

Co-Utile protocols for decentralized computing

Jesús A. Manjón Paniagua



DOCTORAL THESIS
2023

Jesús A. Manjón Paniagua

Co-Utile Protocols for Decentralized Computing

DOCTORAL THESIS

Supervised by Prof. Dr. Josep Domingo-Ferrer

Co-Supervised by Dr. Alberto Blanco-Justicia

Department of Computer Engineering and Mathematics



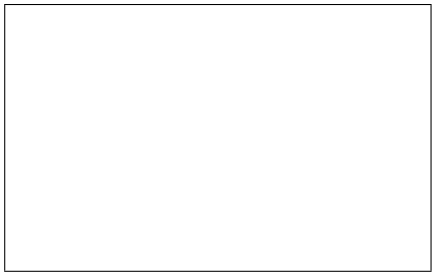
UNIVERSITAT ROVIRA i VIRGILI

July 2023

WE STATE that the present study, entitled "Co-Utile Protocols for Decentralized Computing", presented by Jesús A. Manjón Paniagua for the award of the degree of Doctor, has been carried out under our supervision at the Department of Computer Engineering and Mathematics of this university

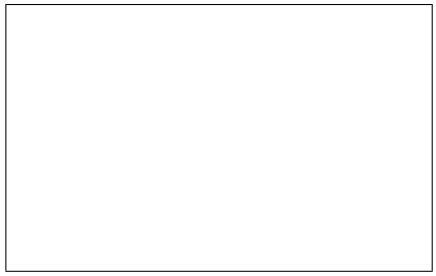
Tarragona, September 15, 2023

Doctoral Thesis Supervisor



Prof. Dr. Josep Domingo-Ferrer

Doctoral Thesis Supervisor



Dr. Alberto Blanco-Justicia

Acknowledgements

I would particularly like to thank my advisors Prof. Josep Domingo-Ferrer and Dr. Alberto Blanco for their guidance and support during the development of this thesis. I would also like to thank Prof. David Sánchez for his collaboration in the writing of some of the papers that support the content of this thesis.

I am also grateful to all the people who have been part of the CRISES research group since I arrived here in 2006. From full professors to PhD students, I have learned something valuable from each of them. Special thanks go to Dr. Jordi Castellà-Roca and Dr. Alexandre Viejo who have been here since the beginning.

Last but not least, I would like to thank all the people who were there when I needed them: my friends from the AEiG Alverna, the TJNBA group and, especially, my wife, my sons, my parents, my brother and my mother-in-law. Thank you for all your support and love.

This work was partially supported by the projects CONSENT (RTI2018-095094-B-C21) and CURLING (PID2021-123637NB-I00).

Contents

List of Figures **xiii**

List of Tables **xvii**

Abstract **xix**

Resum **xxi**

Resumen **xxiii**

1 Introduction **1**

1.1 History of decentralized computing 1

1.2 Motivation 3

1.3 Aims of the research 5

1.4 Contributions of the research 6

1.5 Thesis outline 7

2 Background **9**

2.1 Peer-to-peer networks 9

2.1.1 Incentives and trust in peer-to-peer networks 10

2.2 Co-utility 11
2.2.1 Self-enforcing protocols 11
2.2.2 Co-utile protocols 13
2.2.3 Co-utile reputation management 14
2.3 Decentralized computing and machine learning 15
2.3.1 Privacy and security attacks on federated learning 16
2.4 Multiparty computation 19

3 Secure and Privacy-Preserving FL via Co-Utility **21**
3.1 Introduction 21
3.2 Contributions 21
3.3 A co-utile framework for privacy-preserving and secure FL . . . 22
3.3.1 Players and security model 22
3.3.2 Requirements 24
3.3.3 Co-utile decentralized reputation 25
3.3.4 Downstream: from update generator to model manager . 27
3.3.5 Upstream: from model manager to update generator . . . 29
3.4 Co-utility analysis 32
3.4.1 Co-utility for the model manager 32
3.4.2 Co-utility for the update generator 33
3.4.3 Co-utility for the update forwardees 35
3.4.4 Co-utility for the accountability managers 36
3.5 Privacy and security 36
3.5.1 Privacy 37
3.5.2 Security 39

3.5.3 Computation and communications overhead 40
3.6 Experimental results 43
3.6.1 Test scenario 1 44
3.6.2 Test scenario 2 45
3.7 Summary 50

4 Secure, Accurate and Privacy-Aware FDML via Co-Utility 51
4.1 Introduction 51
4.2 Contributions 52
4.3 Co-utile FDML based on tit-for-tat 53
4.3.1 Model dissemination 59
4.3.2 Update exchange 61
4.3.3 Update forwarding 62
4.3.4 Punishing bad updates 63
4.3.5 Punishing duplicate updates 66
4.4 Co-utility analysis 67
4.4.1 Model dissemination 67
4.4.2 Update computation 69
4.4.3 The privacy tit-for-tat 69
4.4.4 The learning tit-for-tat 70
4.4.5 Model updating 70
4.4.6 Punishment 71
4.5 Privacy and security 72
4.5.1 Privacy 72
4.5.2 Security 73

4.6 Experimental results 74
4.6.1 Experimental setting 75
4.6.2 Baseline scenario 78
4.6.3 Protection against attacks 80
4.6.4 Overhead incurred by the protocol 89
4.7 Conclusions and future work 91

5 General-Purpose Co-Utile MPC **93**
5.1 Introduction 93
5.2 Contributions 94
5.3 A co-utile framework for multi-party computation 95
5.3.1 Players and security model 95
5.3.2 Requirements 97
5.3.3 General-purpose MPC in the honest-but-curious model . 97
5.3.4 General-purpose MPC in the rational model 98
5.4 Co-utility analysis 108
5.4.1 Co-utility for clients 109
5.4.2 Co-utility for forwardees 110
5.4.3 Co-utility for workers 111
5.4.4 Co-utility for accountability managers 112
5.5 Privacy and correctness 113
5.5.1 Privacy 113
5.5.2 Correctness 116
5.6 Example applications 116

5.6.1 Finding differences with the previous and following parties in a ranking or an auction 116
5.6.2 Electronic voting 116
5.7 Experimental results 117
5.7.1 Evaluation of the co-utile framework 118
5.7.2 Comparison with the baseline framework 120
5.8 Conclusions and future work 121

6 Conclusions and Future Work **123**
6.1 Conclusions 123
6.2 Publications 125
6.3 Future work 125

Bibliography **127**

List of Figures

2.1 Sequential version of the Battle of the Sexes game 13
2.2 Self-enforcing protocols in the Battle of Sexes game 14

3.1 FL: Scenario 1. Goodness probability vs reputation for each peer . 45
3.2 FL: Scenario 1. Generating peer's goodness probability vs submitting peer's reputation, for all updates 46
3.3 FL: Scenario 2. Goodness probability vs reputation for each peer . 47
3.4 FL: Scenario 2. Evolution of the average and the standard deviation of the reputations of good peers and bad peers as a function of the epoch 47
3.5 FL: Scenario 2. Generating peer's goodness probability vs submitting peer's reputation, for all updates 48
3.6 FL: Scenario 2. Ratio of bad updates discarded by the model manager as a function of the training epoch 49
3.7 FL: Scenario 2. Evolution of the reputation of newcomers 49

4.1 FDFL: Sequence diagram showing the interactions among peers in an example execution of Protocol 1 60

4.2 FDFL: Sequence diagram showing the interactions among peers in an example execution of the PUNISH_BAD subprotocol 65
4.3 FDFL: Sequence diagram showing the interactions among peers in an example execution of the PUNISH_DUPLICATE subprotocol . . . 68
4.4 FDFL: Baseline scenario. Ratio of useful (good and non-duplicate) updates received vs requests sent at each epoch 79
4.5 FDFL: Flow of updates 80
4.6 FDFL: Scenario with evil peers. Ratio of useful updates received vs requests sent at each epoch 81
4.7 FDFL: Scenario with evil peers. Ratio of useful updates received vs requests sent at each epoch when the effectiveness of the model manager's detection mechanism for bad updates is degraded 82
4.8 FDFL: Scenario with evil peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of evil peers 83
4.9 FDFL: Scenario with selfish peers. Ratio of useful updates received vs requests sent at each epoch 84
4.10 FDFL: Scenario with selfish peers. Computed updates against received updates for selfish and honest peers 85
4.11 FDFL: Scenario with selfish peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of selfish peers 86
4.12 FDFL: Scenario with duplicators. Ratio of useful updates received vs requests sent at each epoch 87

4.13 FDFL: Scenario with duplicator peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of duplicator peers 88
4.14 FDFL: Scenario with whitewashers. Ratio of useful updates received vs requests sent at each epoch 89
4.15 FDFL: Scenario with evil peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of whitewasher peers 90
4.16 FDFL: Scenario with an honest newcomer. Ratio of useful updates received vs requests sent at each epoch 91
4.17 FDFL: Overhead of the tit-for-tat protocol and three protocols based on decentralized global reputation 92

5.1 MPC: Evolution of the reputations of every kind of node 119
5.2 MPC: Rate of correct outputs as a function of the reputation of clients 120
5.3 MPC: Rate of correct outputs in the baseline and the co-utile frameworks for different types of clients, as a function of the proportion of malicious peers 121

List of Tables

3.1 Notation in Chapter 3 23

4.1 Notation in Chapter 4 54

4.2 Ratio of useful updates received by a malicious peer vs an honest peer for different values of δ 77

4.3 Ratio of useful updates received by malicious peers vs honest peers for different values of soft punishment and reward fractions of δ . . 78

Abstract

Decentralized computing is a growing paradigm that distributes computing tasks and decision making across a network. It brings forth numerous advantages, including resilience, scalability, privacy, and reduced dependence on central authorities. The rise of blockchain technology and other decentralized solutions, such as peer-to-peer networks, distributed ledger technologies, and decentralized file systems, has fueled the expansion of the field.

For example, decentralized computing can provide a solution to the tension that arises between the General Data Protection Regulation (GDPR) and big data analytics, which comes from the conflicting goals of privacy protection and data-driven insights. The GDPR aims to protect the privacy rights of individuals and ensure the responsible handling of personal data. It imposes strict requirements on organizations that collect, process, and analyze personal data, including the principles of lawfulness, transparency, purpose limitation, data minimization, and data subjects' rights. Big data analytics, on the other hand, involves processing and analyzing large volumes of diverse data to extract valuable insights and patterns. It often requires collecting and processing large amounts of personal data to uncover correlations, trends, and predictive models. The use of big data analytics has the potential to deliver significant societal benefits, such as improving healthcare, enhancing cybersecurity, and optimizing business operations.

Decentralized protocols such as Federated Learning or Fully Decentralized Ma-

chine Learning offer a solution to this dilemma by eliminating the need to send data from client devices to global servers. Instead, the raw data on edge devices are used to train the models locally, thereby increasing data privacy.

This thesis focuses on one of the most important issues in decentralized protocols: how to ensure that all agents involved perform as expected. Nodes that deliberately deviate may do so to attack the system or simply to take advantage of it without contributing.

Based on the notion of co-utility, we have designed several ethics-by-design frameworks to solve the conflict among privacy and some security properties in three different decentralized and privacy-preserving computing scenarios: i) Federated learning; ii) Fully decentralized learning; and iii) Multi-party computation. These types of protocols facilitate collaboration among multiple parties or entities to achieve a common goal and operate under certain trust assumptions.

Resum

La computació descentralitzada és un paradigma creixent que distribueix les tasques informàtiques i la presa de decisions mitjançant una xarxa. Aporta nombrosos avantatges, com resistència, escalabilitat, privadesa i menys dependència d'entitats centrals. L'expansió de la tecnologia de les cadenes de blocs (*blockchain*) i d'altres solucions descentralitzades, com les xarxes entre iguals, les bases de dades distribuïdes o els sistemes de fitxers descentralitzats, ha impulsat l'expansió d'aquest camp.

Per exemple, la computació descentralitzada pot aportar una solució a la tensió que sorgeix entre el Reglament General de Protecció de Dades (RGPD) i l'anàlisi de dades massives, a causa dels objectius contraposats de protecció de la privadesa, d'una banda, i d'extracció del coneixement de les dades, de l'altra. El RGPD pretén protegir els drets de privadesa de les persones i garantir el tractament responsable de les dades personals. Imposa requisits estrictes a les organitzacions que recopilen, processen i analitzen dades personals, inclosos els principis de licitud, transparència, limitació de la finalitat, minimització de dades i drets dels interessats. L'anàlisi de dades massives, per la seva banda, implica el tractament i l'anàlisi de grans volums de dades diverses per extreure'n informació i patrons valuosos. Sovint requereix recopilar i processar grans quantitats de dades personals per descobrir correlacions, tendències i models predictius. L'ús de l'anàlisi de dades massives pot reportar beneficis socials importants, com ara la millora de l'assistència sanitària, la millora de la ciberseguretat i l'optimització de les

operacions empresarials.

Els protocols descentralitzats com l'aprenentatge federat (*Federated Learning*) o l'aprenentatge automàtic totalment descentralitzat (*Fully Decentralized Machine Learning*) ofereixen una solució a aquest dilema en eliminar la necessitat d'enviar dades des dels dispositius client cap als servidors globals. En comptes d'això, les dades dels dispositius perifèrics es fan servir per entrenar els models localment, cosa que n'augmenta la privadesa.

Aquesta tesi se centra en una de les qüestions més importants en aquest tipus de protocols: com garantir que tots els agents implicats actuïn com s'espera. Els nodes que es desvien deliberadament podrien fer-ho per atacar el sistema o simplement per aprofitar-se'n sense contribuir-hi.

Basant-nos en la noció de coutilitat, hem dissenyat diversos protocols per resoldre el conflicte entre la privadesa i algunes propietats de seguretat en tres escenaris diferents de computació descentralitzada i preservadora de la privadesa: i) aprenentatge federat; ii) aprenentatge totalment descentralitzat; i iii) computació multipart. Aquests tipus de protocols faciliten la col·laboració entre múltiples parts o entitats per assolir un objectiu comú i operen sota certs supòsits de confiança.

Resumen

La computación descentralizada es un paradigma en auge que distribuye las tareas informáticas y la toma de decisiones a través de una red. Aporta numerosas ventajas, como resistencia, escalabilidad, privacidad y menor dependencia de entidades centrales. El auge de la tecnología de las cadenas de bloques (*blockchain*) y de otras soluciones descentralizadas, como las redes entre pares, las bases de datos distribuidas o los sistemas de archivos descentralizados, ha impulsado la expansión de este campo.

Por ejemplo, la computación descentralizada puede aportar una solución a la tensión que surge entre el Reglamento General de Protección de Datos (RGPD) y el análisis de datos masivos, debido a los objetivos contrapuestos de protección de la privacidad, por un lado, y extracción del conocimiento de los datos, por el otro. El RGPD pretende proteger los derechos de privacidad de las personas y garantizar el tratamiento responsable de los datos personales. Impone requisitos estrictos a las organizaciones que recopilan, procesan y analizan datos personales, incluidos los principios de licitud, transparencia, limitación de la finalidad, minimización de datos y derechos de los interesados. El análisis de datos masivos, por su parte, implica el tratamiento y análisis de grandes volúmenes de datos diversos para extraer ideas y patrones valiosos. A menudo requiere recopilar y procesar grandes cantidades de datos personales para descubrir correlaciones, tendencias y modelos predictivos. El uso del análisis de datos masivos puede reportar importantes beneficios sociales, como la mejora de la asistencia sanitaria, la mejora de la

ciberseguridad y la optimización de las operaciones empresariales.

Los protocolos descentralizados como aprendizaje federado (*Federated Learning*) o aprendizaje automático totalmente descentralizado (*Fully Decentralized Machine Learning*) ofrecen una solución a este dilema al eliminar la necesidad de intercambiar datos desde los dispositivos cliente a los servidores globales. En su lugar, los datos de los dispositivos periféricos se utilizan para entrenar los modelos localmente, lo que aumenta la privacidad.

Esta tesis se centra en una de las cuestiones más importantes en este tipo de protocolos: cómo garantizar que todos los agentes implicados actúen como se espera. Los nodos que se desvían deliberadamente podrían hacerlo para atacar al sistema o simplemente para aprovecharse de él sin contribuir.

Basándonos en la noción de co-utilidad, hemos diseñado varios protocolos para resolver el conflicto entre la privacidad y algunas propiedades de seguridad en tres escenarios diferentes de computación descentralizada y preservadora de la privacidad: i) aprendizaje federado; ii) aprendizaje totalmente descentralizado; y iii) computación multiparte. Estos tipos de protocolos facilitan la colaboración entre múltiples partes o entidades para alcanzar un objetivo común y operan bajo ciertos supuestos de confianza.

Chapter 1

Introduction

1.1 History of decentralized computing

The history of computing is characterized by a succession of centralization and decentralization periods. At the dawn of computing, no one imagined a global market larger than a thousand devices. Mainframes were purchased by governments and big companies to execute complex managerial and operational tasks. These computers had the size of rooms or small buildings and, at the very beginning, only single users could access them at a time, using punched cards or paper tapes. Due to those constraints, both computation and access to computing resources were extremely centralized. As new computer systems evolved, electronic devices for entering data into the mainframes appeared. These dumb terminals (essentially a keyboard and a monitor) allowed multiple users to work with mainframes concurrently but were still totally dependent on the mainframes' computing resources.

In the 80s, the appearance of personal computers led to the first taste of decentralization in computing. Instead of dumb terminals, each user got their own computer at home, with their own CPU and their own memory. However, most home computers were not connected to each other, and could only communicate by

sharing physical memory devices such as tapes or floppy disks. Thus, applications were more independent, split, or disconnected rather than actually decentralized.

It was not until the early 2000s that decentralization certainly arose. With the widespread deployment of the Internet, peer-to-peer (P2P) systems for file transfer and cooperative computation appeared. P2P file-sharing projects like Napster or BitTorrent or distributed computing projects like the Great Internet Mersenne Prime Search (GIMPS) or SETI@Home showed the great potential of decentralized applications.

However, the evolution of computing and the reduction of costs in storage space hampered or reversed this path of decentralization. People and (a few) organizations took advantage of cheap computation and storage resources to build large, centralized services on which very popular applications were built: social media sites such as Facebook, LinkedIn, Twitter, or Instagram, file-sharing services such as Dropbox, Google Drive, or One Drive, and instant messaging services such as WhatsApp, Telegram, Messenger, etc. The rise of massively distributed (but not decentralized) data centers and systems and the availability and general access to them made it easier than ever to create applications that can serve millions of users. Most of the content we create and the data we generate today is again centralized in resources owned by just a few companies.

This scenario, while most certainly convenient for users, is also quite lucrative for these (few) companies. Access and control over user data have allowed the birth of a new kind of economy, based on access and analysis of personal data to profile users to, on the one hand, provide personalized services and second, and more important for these companies, to serve targeted ads that maximize their economic returns. In time, it has been demonstrated with cases such as the Facebook–Cambridge Analytica data usage scandal[1] that placing trust in these centralized services might not be in the best interest of users.

In response to this scenario and its associated technical and ethical issues, a pushback to computational and storage resources to the edge of the networks, involving decentralized architectures, has gained momentum lately[2]. In the field of information and communication technologies, a system is called decentralized when multiple, different, and independent authorities control different system components so that no authority has full control of the whole system and no full

[1] `https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal`
[2] `https://redecentralize.org/`

trust is required in any of the authorities. They are not just network topologies but systems with a purpose, which execute protocols to achieve some specific goal. These protocols are run by passing messages between nodes, with other nodes often serving as intermediaries to relay the messages.

One of the most, if not the most, promising and disruptive technology is blockchain, a distributed ledger that maintains a continuously growing list of linked ordered records, called "blocks", and with distributed consensus protocols in place to maintain the integrity and consistency of the database. Blockchains, in addition to cryptocurrency transactions, support mechanisms such as smart contracts that can be used to build a new ecosystem of distributed applications (DApps) and services. Among those are decentralized identity management, user content storage, networking, computing services, etc., which combined may bring forth new communication paradigms such as Web3. Although this ecosystem is still very young and no standards have been defined yet, DApps and Web3 have the potential to allow users to recover control of their data and to empower them against powerful corporations.

1.2 Motivation

As we have briefly discussed above, we are currently in the middle of a technological revolution. On one side, we witness the emergence of a new generation of increasingly sophisticated technologies and Internet services allowing new forms of communication. On the other side, every day more and more devices are being connected to the Internet, not only smartphones, laptops, or powerful computers but also new Internet of Things (IoT) devices such as cars, smart home devices, and industrial or e-health equipment. The development of faster and more reliable networks, especially with the extensive roll-out of 5G, accelerates the pace with which IoT deployment occurs. This massive ecosystem produces colossal amounts of data every day[3] and most of them are collected, processed, stored, and analyzed in huge data centers with a serious impact on the environment. From the very beginning, companies have examined their own big data to extract meaningful insights[4] such as hidden patterns, unknown correlations, market trends, and customer preferences, leading to a profound transformation of the traditional

[3] `https://financesonline.com/how-much-data-is-created-every-day/`

[4] `https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html`

business model and our society. Moreover, most of the new Internet services capitalized on the collection of user private data, which has become a new currency of the digital era, exchanging and merging them for secondary uses in a global and enormous data market.

The aforementioned behaviors imply evident threats to users' privacy. On May 2018, the European General Data Protection Regulation (GDPR) came into force and quickly became adopted as a *de facto* global privacy standard by Internet companies. It limits the collection, processing, and sharing of personally identifiable information (PII) and guarantees specific privacy rights to data subjects (physical or legal entities to which the personal data belongs) ensuring that personal data "can only be gathered legally, under strict conditions, for a legitimate purpose". Furthermore, data controllers (the entity that determines the purposes and means by which personal data is processed) must implement privacy-by-design techniques to protect PII against intruders.

Hence, there is currently an intense tension between two opposing interests: on the one side, the GDPR and its goal to provide greater protection and rights to individuals. On the other side, the extremely valuable analysis of the data and the promotion of data sharing between organizations and public authorities to advance research and hence the welfare of humankind in many fields.

Regarding big data analysis, for example, the traditional approach is to have a centralized entity that holds big datasets gathered and merged from different sources which can be used, for example, to train machine learning models. This method leads to high efficiency and accuracy but puts the privacy of users at risk. Users are willing to benefit from accurate and better predictions but may not want to share their own raw data. Hence, some privacy-preserving techniques are needed in order to protect the user's data while training a model. Some (rather limited and computationally expensive) techniques allow training ML models with encrypted datasets so users can encrypt their sensitive data before uploading them to the servers where the analyses are carried out. Other techniques rely on the use of data perturbation methods and assume that the users can provide modified values for sensitive attributes so as to keep the real value private. These techniques are largely based on statistical disclosure control techniques. An alternative approach to cope with this dilemma is to never send the sensitive data directly to the server and leverage decentralized computing to build a common, robust machine learning model.

The main benefits of computing in decentralized scenarios come from the lack of a central authority. First, by removing the central node, we have a system with no real single point of failure. Second, the scalability and flexibility improve because the computational resources are spread among all the nodes and more computing power can be added at any moment. Finally, without a central authority, no node in the network needs to collect and manage the data of the rest of the nodes. The trust that a node needs to place in third parties can be reduced or even removed.

However, decentralized computing is not free of problems. Due to the lack of a central coordinator, decentralized systems require more complex management of routing, naming, and consistency. These systems may prevent some conventional attacks but introduce new threats: as data go through several nodes between the sender and the receiver, data integrity might be compromised and the analysis of traffic and/or metadata might jeopardize privacy. Nevertheless, one of the major hurdles for decentralization to become widespread is how to guarantee that all agents involved perform as expected. Nodes that intentionally deviate may do so to attack the system (malicious behavior) or just to benefit from the system without contributing to it (selfish behavior).

The most usual approach when designing protocols for decentralized computing is to assume a behavior that is semi-honest (agents will honestly follow the protocol rules, although perhaps they will try to learn more than they should about other agents) or malicious (agents may arbitrarily deviate from the prescribed rules). As a result, although decentralized computing can in principle lead to more ethical solutions than centralized computing, correct behavior from all agents cannot be taken for granted, because they are autonomous and their motivations might not be aligned with the general purpose. In this case, artificial incentives should be added to compensate for the influence of negative utilities and to compel agents to behave as they should.

1.3 Aims of the research

The main objective of this thesis is to design ethics-by-design protocols for decentralized computing in the rational model, that court rational agents into adhering to them, in order to solve the conflict between some ethical values and security properties. These features are captured by the notion of co-utility[1], a design

paradigm to build self-enforcing protocols that make mutual help between rational agents the best strategy.

Three different decentralized and privacy-preserving computing scenarios are considered, where protocols enable collaboration between multiple parties or entities to achieve a common goal under specific trust assumptions:

- **Federated learning** (FL, [2]) is a machine learning technique where each federated device shares its local model parameters with a model manager instead of sharing the whole dataset used to train it. By design, federated learning is more privacy-friendly than the traditional approach because the private data used for training the models are not uploaded to the model manager. However, information on the user's private data may still be leaked by the model update returned by the user. A major security problem of FL is that it is vulnerable to Byzantine attacks, aimed at preventing the model from converging, and to poisoning attacks, aimed at causing convergence to a wrong model.
- **Fully decentralized machine learning** (FDML, [3]) is the extreme form of federated learning. Unlike FL, in this scenario each peer is its own model manager and uses other peers as workers that compute model updates based on their respective private data, exchanging messages without any central coordination.
- **Multiparty computation** (MPC, [4]) protocols allow several parties to compute a joint function in such a way that each party's inputs and outputs remain private to that party. Example applications include secure statistical analysis, financial oversight, electronic voting, secure machine learning, auctions, biomedical computations (in particular those involving highly sensitive data, such as genetic information), etc.

1.4 Contributions of the research

Our contributions are ethics-by-design protocols to solve the conflict between some ethical values and security properties in the above three scenarios.

In the first contribution, we build a co-utile federated learning framework that offers both privacy to the participating peers and security against Byzantine and

poisoning attacks. The framework consists of several protocols designed in such a way that no rational party is interested in acting maliciously. This makes our protocols robust against security attacks. The protocols also provide strong privacy to the participating peers via unlinkability between peers and their updates, and without requiring the aggregation of model updates. In this way, peer updates reach the model manager individually, while being, at the same time, perfectly accurate. This provides an optimum balance between security, privacy, and learning accuracy.

Our second contribution presents a co-utile approach to fully decentralized machine learning that allows perfectly accurate individual updates to be returned by peers to the model manager in a privacy-preserving manner. Rational (that is, self-interested) peers can be expected to follow the protocol we propose without deviating, thanks to a tit-for-tat mechanism that renders the protocol naturally co-utile. This makes our proposal sustainable and robust against rational security attacks.

Finally, in the third contribution, we propose an MPC protocol that is agnostic to the specific computation to be performed (that is, any function can be computed). Our method assumes a P2P community is available, makes a minimalistic use of cryptography, and leverages co-utile computation outsourcing using anonymous channels and a decentralized reputation mechanism.

1.5 Thesis outline

The remainder of this thesis is organized as follows. In Chapter 2, we discuss the necessary background on peer-to-peer networks, co-utility, machine learning, and multiparty computation. In Chapter 3, we present our first contribution, a co-utile framework for federated learning. The fully decentralized machine learning scenario is addressed in Chapter 4. In Chapter 5, we then propose a general-purpose co-utile MPC protocol. Finally, concluding remarks and some guidelines for future research are given in Chapter 6.

Chapter 2

Background

In order to make the understanding of the upcoming chapters easier, this chapter provides some required background information: Section 2.1 provides an overview of peer-to-peer networks, including Distributed Hash Tables, and an introduction to incentive and reputation systems, focusing on the EigenTrust algorithm. Section 2.2 gives some background on co-utility. Section 2.3 presents the decentralized paradigms for machine learning and some related security and privacy attacks. Finally, Section 2.4 introduces the concept of Multiparty Computation (MPC) and the evolution of MPC protocols.

2.1 Peer-to-peer networks

A peer-to-peer network is a computer network that enables peers to share network resources, computational power, and data storage, without relying on a central authority. Unlike traditional client-server networks, where servers only provide content, and clients only consume content, in P2P networks each peer is equally privileged, both a client and a server. Peers act autonomously responding to a common communication and consensus protocol. In this way, the members of the network can exchange information directly and without intermediaries. P2P networks are the natural architecture for decentralized computing protocols.

Peer-to-peer networks generally implement some form of virtual overlay network on top of the physical network topology, where the nodes in the overlay form a

subset of the nodes in the physical network. Overlays are used for indexing and peer discovery, and make the P2P system independent from the physical network topology. Based on how the nodes are linked to each other within the overlay network, and how resources are indexed and located, we can classify networks as *unstructured* or *structured*. In unstructured networks, connections among peers are formed arbitrarily without a particular structure. In order to discover other peers in the network, these query data using some techniques such as flooding or random walking. However, in structured networks, peers organize themselves in specific network topologies and maintain information about the resources that their neighbors possess. Some of these networks impose constraints both on node (peer) topology and on data placement to enable efficient discovery of data. The most common indexing system is the Distributed Hash Table (DHT). Similar to a hash table, a DHT provides a lookup service with (key, value) pairs that are stored in the DHT. Any participating peer can efficiently retrieve the value associated with a given unique key. Different DHT-based systems such as Chord [5], Pastry [6], or CAN [7] differ in their routing strategies and their organization schemes for the data objects and keys. For example, Chord uses a variant of consistent hashing [8] to assign keys to nodes. In the basic variant of a DHT, when a peer joins or leaves a P2P network, the lookup tables stored by each of the peers have to be recomputed to assign pieces of data to the new peers, which is an expensive operation. In contrast, consistent hashing is designed to let peers enter and leave the network with minimal overhead. When a peer leaves the network, only the data that were mapped to that peer need to be reassigned. We use a similar mechanism in our second contribution to disseminate the models in the fully decentralized machine learning scenario.

2.1.1 Incentives and trust in peer-to-peer networks

Lack of cooperation (free riding) is one of the key problems in P2P systems. The anonymous and open nature of these networks often results in many peers selfishly benefiting from the network resources while not contributing any of their own resources. Peers have natural disincentives to cooperate because cooperation consumes their own resources and may degrade their own performance. As a result, users attempting to maximize their own utilities effectively lower the overall utility of the network and, eventually, make the system collapse.

The main theoretical framework to study these possibly conflicting interactions in the literature is game theory [9], which often models P2P systems as a Pris-

oners' Dilemma [10]. Two classes of incentive schemes have been considered: soft schemes, which use a reputation system to model the peers' behavior allowing peers to determine their decisions depending on the reputations of the other peers; and hard schemes, which use monetary payments to compensate peers contributing to the P2P network [11]. These methods also provide the P2P network with a level of robustness against malicious nodes.

Reputation systems have been used from the very beginning mainly in the context of packet forwarding [12] and file sharing, one of the main applications of P2P networks in the early days. Many reputation systems were proposed for these kinds of P2P networks [13], being EigenTrust [14] one of the most cited and adapted systems.

2.2 Co-utility

Domingo-Ferrer et al. proposed the notion of co-utility in [1, 15, 16], as a type of interaction between rational agents in which the best option for each agent to reach her own goal is to help other agents reach theirs.

2.2.1 Self-enforcing protocols

A protocol specifies a precise set of rules that govern the interaction between agents performing a certain task; that is, it details the expected behavior of each agent involved in the interaction for the task to be successfully completed. For protocols to be effective, they must be adhered to. This is not problematic when the participating agents cannot deviate by design, but it becomes an issue when the agents are free to choose between following the protocol or not. Although free agents cannot be forced to follow a protocol, rational free agents can be persuaded to do it if the protocol is properly designed. These protocols are called *self-enforcing protocols*.

With rational agents in mind, game theory [17] is the natural way to formalize the concept of protocol because this theory models interactions between self-interested agents that act strategically. In [15], a game is used to model all the possible interactions among agents in the underlying scenario. In particular, the game includes also those interactions among agents that are not desired. Then, a protocol is regarded as a prescription of a specific behavior in the underlying scenario, that is, a sequence of desired interactions.

Domingo-Ferrer et al. focused on sequential games with perfect information because this is quite a common and basic type of interaction between agents. We say that a game is sequential if, at the time of choosing a move, previous moves made by other agents are known (at least to some extent), and a sequential game is said to be a perfect information game if the agent about to make her move has complete knowledge on the previous moves made by the other agents.

A perfect information game G can be represented in extensive form as a tree where:

- Internal nodes represent decision-making points and are labeled with the name of the agent making the move;
- Leaf nodes are labeled with the utility each of the agents gets if the leaf is reached;
- Outgoing edges in a node represent the actions available to the agent making the decision.

A protocol Π on a game G is either a path from the root to a leaf or a subtree from the root to several leaves. Π is self-enforcing if, at each successive node of the protocol path, an agent selects a move that yields an equilibrium of the remaining subgame, that is, of the portion of the game that remains to be played. Nash equilibrium in game theory is a situation in which each player, after taking into consideration the opponent's strategy, has no incentive to deviate from her own because she could not increase her expected utility and would not have anything to gain.

For example, let us find Nash equilibria in the sequential version of the well-known Battle of the Sexes (BoS) game introduced by R. Duncan Luce and Howard Raiffa in [18] and represented by Figure 2.1. Such equilibria are usually found by means of backward induction [17]. The backward induction algorithm assumes that, at each node, the agent making the move selects the action that gives her the best outcome. The algorithm starts evaluating the decision nodes that are parents of terminal nodes. Once the moves have been selected, the algorithm proceeds backwards by evaluating the choice of the predecessor nodes. In the BoS game, M is the agent making the last choice. In the left branch, the best option for M is O, which leads her to utility (3, 2). In the right branch, the best option

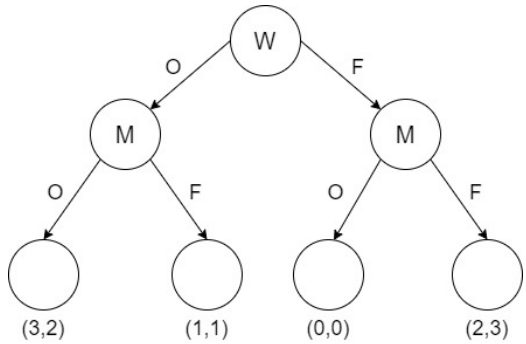


Figure 2.1: Sequential version of the Battle of the Sexes game. The woman (W) chooses between going to the opera (O) or to a football match (F); then the man (M), does the same. The utility is how much they enjoy themselves: both would like to go together, but W prefers opera and M football.

for M is F, which leads her to utility (2, 3). Because W knows that M will seek to maximize his own utility, W can simplify the original tree to get a tree with a single decision node (hers) in which choosing 0 leads to utility (3,2) and choosing F leads to utility (2,3). Thus W should select O in the first place, which maximizes her own utility, and M should then also select O. As (O, O) is a Nash equilibrium of the BoS game, and (O) is a Nash equilibrium of the game that remains after the first action, protocol $P = (O, O)$ is self-enforcing. But it is not the only one as you can see in Figure 2.2. Protocol (F, F) is also self-enforcing because no agent can improve her utility by deviating from the protocol provided that the other agent sticks to it. Of course, since in the sequential version W chooses the move first, she is most likely to favor (O, O) over (F, F).

2.2.2 Co-utile protocols

We say that a protocol is co-utile if it results in a mutually beneficial collaboration between the participating agents. More specifically, a protocol P is co-utile if and only if the three following conditions hold:

- P is self-enforcing;
- the utility derived by each agent participating in Π is strictly greater than the utility the agent would derive from not participating;

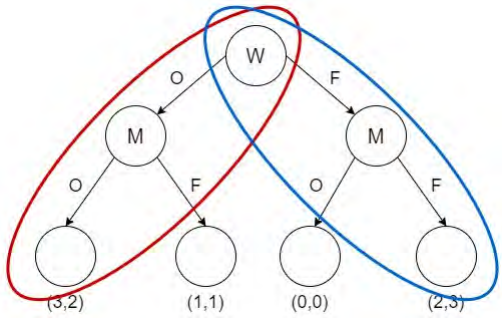


Figure 2.2: Self-enforcing protocols in the Battle of Sexes game

- there is no alternative protocol Π' giving greater utilities to all agents and strictly greater utility to at least one agent.

The first condition ensures that if participants engage in the protocol, none of them can increase their utility by deviating. The second condition is needed to ensure that engaging in the protocol is attractive for everyone. The third condition ensures that there is no alternative protocol whereby participants could get a better outcome; hence, Π is Pareto-optimal.

Co-utility may arise naturally, but, unfortunately, in most real-life cases, "negative" incentives (like costs, privacy loss, fear of strangers, etc.) may override positive incentives for the agents to follow the rules. Hence, artificial positive incentives may need to be added to compensate for negative incentives and thereby spark co-utility.

2.2.3 Co-utile reputation management

As we mentioned in section 2.1.1, P2P systems do not have a centralized node that acts as an authority and that monitors, rewards, and punishes the peers according to their behavior. Therefore, a reputation scheme can be introduced in order to "guide" the interaction between peers. These reputations would help to corner malicious peers and would also work as an artificial positive incentive to achieve co-utility.

It turns out, though, that managing the reputations of peers in a distributed way constitutes itself a protocol that requires collaboration (*e.g.*, to compute, update and spread reputations); hence, it is crucial that reputation management

is designed to be co-utile itself so that the collaboration it implies is rationally sustainable.

In [16], authors adapt and extend the well-known Eigentrust reputation calculation mechanism so that: (i) it can be applied to a variety of scenarios and heterogeneous reputations needs and, (ii) as a result, it is itself co-utile, and hence selfish peers are interested in following it.

EigenTrust [14] was originally designed to filter out inauthentic content in peer-to-peer file-sharing networks. Its basic idea is to compute a global reputation for each peer based on aggregating the local opinions of the peers that have interacted with such peer. If we represent the local opinions by a matrix whose component (i, j) contains the opinion of peer i on peer j, the distributed calculation mechanism computes global reputation values that approximate the left principal eigenvector of this matrix.

The resulting adapted protocol is itself strictly co-utile, robust against several attacks, and compatible with peer anonymity. In this way, a virtuous circle can be achieved by which the reputation mechanism is self-enforcing, and, in turn, enables co-utility in protocols that would not be co-utile without reputation. This provides a key tool that we have applied in our second and third contributions. Some adapted concepts have also been used in our first contribution.

2.3 Decentralized computing and machine learning

As we discussed in Section 1, decentralized computing has notable advantages over centralized approaches in terms of scalability, flexibility, security, and privacy. This is particularly interesting for machine learning. The standard methods require the collection and aggregation of massive volumes of data by a server from several edge devices like smartphones or personal computers, but new paradigms are emerging lately.

Federated learning (FL, [19]) is a distributed approach to machine learning. A model manager initializes a global model, such as a neural network. The model manager then sends the model to multiple edge devices acting as workers. Each worker has a private data set (that she does not share with the model manager) on which she trains the global model; then she returns this updated model (or the difference between the global model and the updated model trained on the

peer's private data) to the model manager. Based on all received model updates, the model manager updates the global model, and this process is repeated several times (called epochs) until the global model converges.

Fully decentralized machine learning (FDML) is the extreme form of decentralized machine learning [20, 21, 22]. Unlike in FL, where there is a distinction between workers and the model manager, in FDML all nodes are equally privileged and train their own model. In their role as model managers, each peer periodically sends their current model to other peers, who return model updates based on the private data they hold.

2.3.1 Privacy and security attacks on federated learning

In this section, we will discuss the main privacy and security attacks on federated learning. For recent and exhaustive surveys, see [23, 24].

Privacy attacks

Machine learning models can be vulnerable to privacy attacks that aim at inferring information about the training data. Among privacy attacks we find model inversion attacks [25], which exploit information leaked by the machine learning model about the training data to (partially) reconstruct the potentially private data used for training; membership inference attacks, which attempt to determine whether some particular data points are part of the training data [26]; and attribute inference attacks, which attempt to estimate confidential attributes of individuals whose data is present in the training data [27]. Some of the above attacks are applicable both to conventional centralized machine learning and federated learning.

Privacy attacks on federated learning can exploit the updates sent by peers to infer information on those peers' private data in a white-box setting (where the attacker has access to the model parameters). For example, Hitaj *et al.* [28] present a powerful data inference attack against federated deep learning that relies on GANs (Generative Adversarial Networks). This attack assumes an attacker that can see and use internal parameters of the learned model. The attacker participates as an honest peer in the collaborative learning protocol, but she tries to extract information about a class of data she does not own. To that end, the attacker builds a GAN locally and crafts gradient updates before returning them in order to influence other participating peers to leak more information on

their data. If the attacker is the model manager rather than a peer, she can do more: the model manager can isolate the shared model trained by the victim peer. The victim peer's update trained on the victim's data is used to train the model manager's GAN, that can eventually re-create the victim's data. As explained in [28], not even differential privacy, used as proposed in [29], can protect against the proposed GAN attack.

A common requirement of all data inference attacks in federated learning is that the attacker must be able to link the successive updates submitted by a certain peer.

Security attacks

Security attacks on federated learning aim at disrupting model convergence and thereby the learning process. They can be subdivided into Byzantine and poisoning attacks.

In a general sense, a Byzantine fault, from the Byzantine General's Problem, refers to the problem of reaching consensus in a distributed system [30]. In the context of machine learning, *Byzantine attacks* consist of malicious peers submitting defective updates in order to prevent convergence of the global model [31].

Subtler than Byzantine attacks are *model poisoning attacks*. Rather than preventing convergence, the latter aim at causing federated learning to converge towards a false global model, normally one that misclassifies a specific set of inputs. For example, if the goal is to collaboratively learn a movie recommendation model, a poisoned model might be learned that always recommends a certain movie to all subscribers.

In [32] it is shown that a single, non-colluding malicious peer is enough to mount a poisoning attack. Yet, security attacks can also be mounted by collusions of peers or by a single peer masquerading as several peers (Sybil attack).

Countermeasures against Byzantine or poisoning attacks require accessing the exact values of the individual updates, in order to assess their goodness. This is why some techniques that are good to protect the privacy of peers, such as secure aggregation of peer updates via homomorphic encryption [33], may impair the model manager's ability to thwart security attacks.

There are many approaches in the literature that a model manager can use to decide whether a received update is good or bad. They fall into three main classes:

- *Detection via model metrics [34].* The model manager updates her current model m_i with each single received update u and compares the updated model m_i^u with m_i in terms of accuracy and possibly other performance metrics. If m_i^u offers poorer performance than m_i, then u is probably bad and m_i should be kept rather than m_i^u. This detection approach requires a validation data set on which the two models can be compared and it takes significant computation for each received update.

- *Detection via update statistics.* In this approach, the model manager observes the statistics of the magnitude of the received updates [35]. As the model converges, the update magnitudes tend to zero. Even before convergence, any update that lies far from the other updates is suspicious. Hence, a way to detect bad updates is to use distances [32, 31]: given a batch of updates, an update u is classified as bad if it is much more distant than the other updates from the centroid θ of the batch. One possible way to quantify what "much more distant" means is as follows: if the distance between u and θ is greater than the third quartile (or greater than a small multiple of the third quartile, say 1.5 times) of the set of distances between updates in the batch and θ, then u is classified as a bad update. Note that this distance-based procedure can only be run after a peer has received a batch of updates. This can be accommodated if P_i waits until the end of the epoch before evaluating her updates.

- *Neutralization of bad updates via special aggregation.* This approach consists in ignoring bad updates rather than seeking to explicitly detect them. It combines the above intuitions of distance-based detection and majority voting. The authors of [31] propose to aggregate the received updates using an aggregation function called Krum that is resilient against up to f bad updates. Basically, the Krum aggregation of a set of updates is the most central update, where the centrality of each update u is computed by taking into account all the other updates in the set *except* the f most distant updates from u. Alternative aggregation functions that are also resistant against outlying updates are the coordinate-wise median and the coordinate-wise trimmed mean, proposed in [35]. The former returns the median for every dimension of the received updates, whereas the latter returns the mean for every dimension after removing a fraction of the smallest values and a fraction of the largest values.

2.4 Multiparty computation

Multiparty computation (MPC) consists in several parties engaging in joint computation in such a way that each party's input and output remain private to that party. As phrased in [36], MPC can be viewed as a cryptographic method for providing the functionality of a trusted party —whose role would be to accept private inputs from a set of participants, compute a function and return the outputs to the corresponding participants— without the need for mutual trust among participants.

MPC can be used to solve a variety of problems that require using the data of participants without compromising their privacy. Example applications include secure statistical analysis, financial oversight, electronic voting, secure machine learning, auctions, biomedical computations (in particular those involving highly sensitive data, such as genetic information), etc. See [37, 38, 36] for more details and related references on MPC applications.

The most central security properties required in MPC are as follows:

- *Privacy.* No party should learn anything more than its prescribed output. In particular, nothing should be learned about the other parties' inputs except what can be derived from the output itself.
- *Correctness.* Each party should receive its prescribed output and this output should be correct.

There are different security models to characterize the assumptions on the adversarial behavior of parties in MPC. In the *honest-but-curious* model (also called *semi-honest*), parties are assumed to correctly follow the protocol specification, but they may try to learn other parties' inputs or outputs. In the *malicious* model, parties can arbitrarily deviate from the protocol specification. As pointed out in [37], the honest-but-curious model is very weak and it underestimates the power of realistic adversaries in most scenarios. The malicious model, in contrast, is very strong but only a small subset of MPC protocols in the literature can cope with it [39].

Since the 1980s, when the seminal MPC protocols [4, 40, 41, 42] were proposed, and until very few years ago, practical MPC protocols existed only for specific computations. The appearance of general-purpose compilers enabling MPC for

arbitrary functions is a recent breakthrough; see [36] for a state of knowledge on such compilers. Most MPC compilers translate ordinary high-level programming code expressing the computation to be performed into a Boolean or an arithmetic circuit. Once at the circuit level, they use several cryptographic primitives, such as secret sharing, oblivious transfer, garbled circuits, and others to generate the MPC protocol for the target computation (see [37] for descriptions of such primitives).

As noted in [36], using state-of-the-art compilers requires substantial programming effort and skill on the user's side, due *inter alia* to the inherent constraints of the circuit representation. In particular, loops and recursive calls in the original computation code must be unrolled, which forces programmers to (often manually) define loop bounds and hardly use recursion.

Chapter 3

Secure and Privacy-Preserving Federated Learning via Co-Utility

3.1 Introduction

Federated learning [19, 33], is a decentralized machine learning technique that allows training a model with the collaboration of multiple peer devices holding private local data sets that include class labels. This approach favors privacy because the peers do not need to upload their private data to a centralized server. It is also naturally scalable, because the computational load is split among the peers, which may be edge devices such as idle smartphones, and thus widely available. Unfortunately, as we discussed in Section 2.3.1, the decentralized nature of federated learning makes it vulnerable to attacks against privacy and security.

3.2 Contributions

In this chapter we build a federated learning framework that offers both privacy to the participating peers and security against Byzantine and poisoning attacks. Our framework consists of several protocols designed in such a way that no rational party is interested in acting maliciously. This makes our protocols robust against

security attacks. Our protocols also provide strong privacy to the participating peers via unlinkable anonymity and without requiring the aggregation of model updates. In this way, peer updates reach the model manager individually, while being, at the same time, perfectly accurate. This provides an optimum balance between security, privacy and learning accuracy.

To be rationally sustainable, our protocols are based on the co-utility property presented in Section 2.2. We also use reputation as a utility to reward well-behaved peers and punish potential attackers. In order to properly integrate reputations in the federated learning scenario, our reputation management is decentralized and itself co-utile.

We report empirical results that show the effectiveness of our protocols at mitigating security attacks and at motivating rational peers to refrain from deviating.

Section 3.3 introduces a co-utile protocol suite for privacy-preserving and secure federated learning. Section 3.4 shows that the proposed protocol suite achieves co-utility (and hence is rationally sustainable), and Section 3.5 does the same regarding privacy and security. Experimental results are presented in Section 3.6. Finally, Section 3.7 summarizes conclusions.

3.3 A co-utile framework for privacy-preserving and secure federated learning

The foundations of our proposed protocol suite are: i) the notion of co-utility applied to protocol design and ii) the use of reputations (computed themselves in a decentralized and co-utile manner) to motivate all rational players to behave honestly. In Section 2.2 we gave some background on co-utility. Also, for convenience Table 3.1 summarizes the notation used in the rest of this chapter.

In the next sections we describe a framework based on co-utility that ensures that peers can keep their private data sets confidential and, at the same time, makes them rationally interested in returning honest updates to the model manager.

3.3.1 Players and security model

The players in our framework and their security properties are as follows:

- *Model manager.* The model manager M is a player who wants to train a machine learning model on the private data of the peers in a peer-to-peer

Table 3.1: Notation in this Chapter

Notation	Concept
M	Model manager
AM	Accountability manager
P	Peer
m	Number of AMs per peer
δ	Reputation reward/punishment
U	Federated learning update
N_U	Random nonce encrypted with update U
g_i	Peer P_i's reputation
$PK_M(\cdot)$	Public key encryption under M's public key
$S_P(\cdot)$	Digital signature under P's private key
$H(\cdot)$	Cryptographic one-way hash function
α	Flexibility parameter (Note 1)
p	Forwarding probability
p_0	Probability of discarding an update from a peer with zero reputation
b	Size of a batch of non-discarded updates
C	Centroid of a batch of updates
T	Reputation threshold s.t. if a peer's reputation is at least T, her updates are never discarded

(P2P) network. Her interest is to obtain a good quality model, but she might be curious to learn as much as possible on the peers' private data sets. Hence, M can be viewed as *rational-but-curious*: rational to adhere to her prescribed function, but curious on the peers' private data.

- *Peers.* They are participants in the network who compute model updates based on their local private data sets. Peers want to preserve their private data confidential. We assume that *a majority of peers are rational-but-curious*: like M, they are interested in obtaining a good quality model, but they also want to influence the model based on their own respective data; further, they might be curious to learn as much as possible on the other peers' private data sets. On the other hand, there may be a minority of *malicious peers* that wish to impair the learning process, because they do not have the same utility function and/or do not respond to the same incentives as the rest of peers.

- *Accountability managers.* Accountability managers (AMs) are randomly chosen peers that manage the reputations of other peers. Being peers themselves, most accountability managers are rational-but-curious, but a minority may be malicious.

3.3.2 Requirements

The assumption that peers are rational rather than honest calls for incentives to make honest behavior attractive to them. We will use reputation as an incentive to reward or punish peers. In order for this to be effective, the following requirements need to be fulfilled:

- *Reward.* If a peer contributes a good update, her reputation must increase.

- *Punishment.* If a peer contributes a bad update, her reputation must decrease.

- *Unlinkable anonymity.* Peers contributing good updates must stay not only anonymous, but their successive updates must be unlinkable.

- *Reputation utility.* Having high reputation must be attractive for peers. Specifically, it must be easier for peers with higher reputation to contribute their updates while preserving their privacy. Thus reputation translates to influence without privacy loss.

Unlinkability is our approach to thwarting the privacy attacks sketched in Section 2.3.1 while perfectly retaining the accuracy of the updates. On the other hand, reward, punishment and reputation utility are our tools to protect against the security attacks described in Section 2.3.1. This will become clear in this section and in Sections 3.4 and 3.5 below.

3.3.3 Co-utile decentralized reputation

Whereas we assume that a majority of peers want to learn a good model, we still need to incentivize rational peers to abstain from free-riding: if they find greater utility in deviating from the federated learning protocol, they might seriously impair the overall quality of the learned model. Also, we need a way to stigmatize/recognize malicious peers in order to mitigate their attacks. To meet the above purposes, we will use reputation management. In this section we present a reputation management system that does not require direct interaction between peers and has the following interesting properties: pseudonymity of peers, decentralization, resistance to tampering with reputations, proper management of new peers (to discourage whitewashing bad reputations as new identities and creating fake peers in Sybil attacks) and low overhead.

One of the fundamental changes of this reputation protocol with respect to the EigenTrust based protocol presented in [16] is that peers do not maintain local opinions of the other peers, since they do not see the rest of the updates and they do not know how other peers have behaved. Only the model manager is able to know whether an update was good or bad. Therefore, the distinction between local and global reputation is meaningless.

Our reputation protocol maintains a public reputation for each peer P that is the result of updating P's previous reputation according to the behavior of P reported by the model manager M. Next we explain how the above interesting properties are satisfied:

- *Pseudonymity of peers.* Only the pseudonym of peers is known, rather than their real identity. Furthermore, updates that are sent over the network cannot be linked to the peers that generated them.
- *Decentralization.* The reputation of every peer P is redundantly managed by a number m of peers that act as accountability managers for P. Typically, m is an odd number at least 3 and the (pseudonymous) identities of P's

accountability managers are (pseudo)randomly determined by hashing the peer's pseudonym P. In this way, P cannot choose her m accountability managers, which makes the latter more likely to perform their duty honestly.

- *Tamper resistance.* Since M does not know the identity of peers nor is able to link the updates to peers, M cannot leverage her position to promote or slander any particular peer M likes or dislikes. As a consequence, M's rational behavior is to exclusively base her reports on the quality of the received model updates. Regarding tampering by accountability managers, it is thwarted by their redundancy (see the previous item on decentralization).
- *Proper management of new peers.* Reputations take values in the range $[0, 1]$. New peers start with reputation 0, which makes whitewashing and also Sybil attacks unattractive.

Let us describe the dynamics of reputation. Call *epoch* the period between two successive changes of the global model by M. During an epoch, peers generate and send model updates based on their private data, with the aim of influencing the next global model change. Depending on their actions, peers can earn or lose reputation. Generating a good update increases the generator's reputation by a certain quantum $\delta/2$ fixed by the model manager; furthermore, helping a good update reach the model manager in a way unlinkable to the generator brings a $\delta/2$ reputation increase to one of the helping peers. Thus, every good update results in a total δ reputation increase. On the other hand, generating a bad update decreases the generator's reputation by δ. Thus, the overall reward for a good update equals the punishment for a bad update.

Some peer reputations may become negative and some may become greater than 1 as an epoch progresses. At the epoch's end, reputations are re-normalized into the range $[0, 1]$ as follows. First, accountability managers reset any negative reputation to 0. Then, if there are reputations above 1, all reputations are divided by the largest reputation. To that end, when a peer's reputation becomes larger than 1, the peer's accountability managers broadcast that reputation, which allows all accountability managers to compute the maximum reputation reached in that epoch and thereby normalize all reputations into the interval $[0, 1]$.

Normalization has the beneficial effect of deterring free-riding: even if a peer has attained high reputation, she will lose it gradually if she stops participating. Indeed, any peer's reputation will decrease due to normalization unless she continues

to generate good updates or helps routing them. This addresses the second condition of the co-utility definition: the utility derived from participating must be greater than the utility derived from not participating. Fulfillment of the other two conditions for co-utility will be justified in Section 3.4 below.

3.3.4 Downstream: from update generator to model manager

We call downstream operation the submission of model updates from the peers to the model manager M. In order to preserve privacy and encourage security, we propose Protocol 1. In Section 3.4, we will show that it is co-utile.

The idea of Protocol 1 is that a peer, say P_1, does not directly send her update to M. Rather, P_1 asks another peer, say P_2, to do so. P_2 randomly decides whether to submit P_1's update to M or forward it to another peer, say P_3, who stands the same choice as P_2. Forwarding continues until a peer is found that submits the update to M.

Protocol 1 (Update submission)**.** *1. Let P_1 be a peer that generates an update U. Then P_1 encrypts U along with a random nonce N_U under the model manager's public key, to obtain $PK_M(U, N_U)$ (we assume the message U, N_U to have a certain format that allows distinguishing it from gibberish at decryption). In this way, only M will be able to recover the update U. The generator P_1* never *submits her own update to the manager M; rather, P_1 builds an update message that takes the form*

$$\begin{aligned} U_{1\to 2} = \langle & P_1, P_2, \\ & PK_M(U, N_U), \\ & H(H(H(U, N_U))), \\ & S_{P_1}(PK_M(U, N_U), H(H(H(U, N_U))), P_2)\rangle \end{aligned} \tag{3.1}$$

and forwards it to $P_2 = \textsc{Select}(g_1)$, where function Select*() is explained below. In Expression (3.1), H is a one-way hash function and S_{P_1} is P_1's signature.*

2. If P_1's reputation g_1 is such that $g_1 < \min(g_2, T) - \alpha$, where g_2 is P_2's reputation, T is a parameter such that updates submitted by peers with reputation T or above are never discarded, and α is a flexibility parameter discussed in Note 1, then P_2 discards the received update.

Otherwise, P_2 makes a random choice: with probability $1-p$, she submits $U_{2\to M}$ to M and with probability p she forwards $U_{2\to 3}$ to another peer $P_3 =$ SELECT(g_2).

3. *If P_2's reputation g_2 is below $\min(g_3, T) - \alpha$ then P_3 discards the received update. Otherwise, P_3 makes a random decision as to submit or forward. If it is forward, P_3 will use the* SELECT*() function and there may be more peers involved: P_4, P_5, etc.*

4. *Eventually M receives an update $U_{i\to M}$ from a peer P_i. Upon this, M does:*

   (a) *Directly discard the update with probability $p_0(1 - \min(g_i/T, 1))$, where p_0 is a parameter indicating the probability of discarding an update submitted by a peer with 0 reputation, and g_i is P_i's reputation.*

   (b) *If the update has not been discarded, decrypt $PK_M(U, N_U)$, obtain U, check that the nonce N_U was not received before (to make sure U is not a replay of a previously received update) and check the hash $H(H(H(U, N_U)))$.*

   (c) *Wait until a batch of b non-discarded updates has been received in order to be able to decide whether U is good or bad (see Section 2.3.1 on how to detect bad updates).*

   (d) *Change the model with the good updates in the batch and publish the updated model.*

   (e) *Publish the value $\delta = 1/b$.*

   (f) *For every good non-discarded update U, publish $H(H(H(U, N_U)))$.*

   (g) *For every bad non-discarded update U, call* PUNISH*(P_i) where P_i is the peer having submitted U and* PUNISH*() is Protocol 2 in Section 3.3.5.*

Function SELECT(g_i) is used by a peer P_i to select a forwardee. There are several ways in which this can be accomplished. However, the rational choice is for P_i to select a forwardee P_j with a sufficient reputation so that M does not reject the update should P_j submit it directly to M. Hence, if P_i's reputation is $g_i \geq T - \alpha$, P_i can randomly pick any of the peers whose reputation is T or above, because none of those peers risks update discarding. However, if $g_i < T - \alpha$, P_i chooses the peer with the maximum reputation that does not exceed $g_i + \alpha$, because no peer with reputation above that value will accept to forward P_i's update.

Note 1 (On the flexibility parameter α)**.** In Protocol 1 a peer accepts to forward updates from peers that have at least her own reputation minus a flexibility amount α. Using a small value $\alpha > 0$ introduces some flexibility and helps new peers (that start with 0 reputation) to earn reputation as generators or first forwardees of good updates. Large values of α are not acceptable from the rational point of view: high-reputation peers have little to gain by accepting updates from peers who are much below them in reputation, because the latter are likelier to convey bad updates or to fail to reward the first forwardee in case of good updates.

Note 2 (On loops, multiple paths and other misbehaviors)**.** Nothing is gained by any peer if loops arise accidentally or intentionally in Protocol 1. As it will be seen below (Protocol 3 and Note 4) only the first peer chosen by the update generator is rewarded. Hence, forwarding twice or more times the same message brings no additional benefit. On the other hand, a generator P might send the same good update through several paths to increase the reputation of several first peers. However, by promoting more peers than necessary, P may experience a decrease of her own reputation, because reputations are normalized when any peer reaches a reputation above 1 (see Section 3.3.3). Finally, update generators could systematically choose themselves as first forwardees of good updates to collect additional reward; but if they do so, they weaken their privacy.

Note 3 (Key generation)**.** In Protocol 1, peers sign the messages they send. To that end, each peer needs a public-private key pair. At least the two following alternative key generation procedures are conceivable: i) identity-based signatures, in which the peer's pseudonym is her public key and the peer's private key is generated by a trusted third-party [43]; ii) blockchain-style key generation [44], in which the peer generates her own key pair without the intervention of any trusted third-party or certification authority, and then obtains her pseudonym P_i (her *address* in the blockchain network) as a function of her public key.

3.3.5 Upstream: from model manager to update generator

By upstream operation we denote the punishment of bad updates and the reward of good updates. Let us start with Protocol 2 that seeks to penalize the generator of a bad update by retracing the reverse path from M to the generator. The peer P_i who submits an update found to be bad by the manager can escape punishment if P_i can show to her accountability managers that she received the bad update from a previous peer, say P_{i-1}.

Protocol 2 (PUNISH(P_i))**.**
Every accountability manager AM of P_i's does:

1. *Ask P_i whether P_i can prove she did not generate U.*
2. *If P_i can show to AM a message*

$$S_{P_{i-1}}(PK_M(U, N_U), H(H(H(U, N_U))), P_i)$$

then

(a) Do not punish P_i (the peer's reputation is left intact);
(b) Call PUNISH*(P_{i-1}).*

Otherwise, punish P_i by decreasing her reputation by δ.

The punishment protocol must be initiated by M, because the model manager is the only party that can detect bad updates and that is interested in punishing them. However, the punishment is actually executed by the guilty peer's accountability managers. Hence, M cannot track which peer is actually punished for that bad update, which prevents M from identifying the generator of an update by (falsely) claiming that the update is bad.

Unlike the punishment protocol, the rewarding protocol is initiated by the peer who submitted a good update, because that peer is the one interested in the reward. As we will later justify, the first peer (and only the first peer) who is asked by the generator to submit or forward a good update is also entitled to a reward. We will call that peer the "first forwardee".

Protocol 3 (REWARD(U))**.**
1. *When M publishes $H(H(H(U, N_U)))$ for a good update, then the update generator, say P_1, sends to the first forwardee, say P_2, $S_{P_1}(H(H(U, N_U)), P_2)$.*
2. *P_2 checks that the hash of $H(H(U, N_U))$ matches $H(H(H(U, N_U)))$ published by M. If it is so, P_2 returns a receipt $S_{P_2}(H(H(U, N_U)), P_1)$ to the generator P_1.*
3. *P_1 proves to her accountability managers that she is the generator by showing $H(U, N_U)$ to them and proves that she has acknowledged her first forwardee by showing the receipt $S_{P_2}(H(H(U, N_U)), P_1)$.*

4. *Every accountability manager AM of P_1's checks P_2's receipt and checks that the* double *hash of $H(U, N_U)$ received from P_1 matches $H(H(H(U, N_U)))$ published by M. If* both *checks are fine, AM increases P_1's reputation by $\delta/2$.*

5. *P_2 sends $S_{P_1}(H(H(U, N_U)), P_2)$ to her accountability managers to claim her reward.*

6. *Every accountability manager AM of P_2's checks that the hash of $H(H(U, N_U))$ matches $H(H(H(U, N_U)))$ published by M. If it is so, AM increases P_2's reputation by $\delta/2$.*

Note 4 (On rewarding the first forwardee only)**.** In Protocol 3 only the first forwardee is rewarded, rather than all forwardees. The reason is that we want the total budget to reward a good update to be fixed and equal to the budget δ used to punish a bad update. We also want the reward share for the generator of a good update to be fixed, say $\delta/2$, and independent of the number of hops the update travels before reaching M. Hence, if we chose to reward all forwardees, the fixed reward share $\delta/2$ for forwardees ought to be distributed among them. Therefore, every forwardee would be better off by submitting the update to M rather than forwarding it to another forwardee who would take part of the reward. As a consequence, there would be only one forwardee, who would know that the previous peer is the generator of U. This would break privacy. Rewarding only the first forwardee avoids this problem and is a sufficient incentive, because any forwardee can hope to be the first (due to the protocol design, a forwardee does not know whether she receives an update from the generator or from another forwardee) and thus has a reason to collaborate.

Note 5 (On peer dropout)**.** Accidental (due to power or network failure) or intentional peer dropout does not affect the learning process: on the one hand, once an update has been generated/forwarded, the generator/forwarder can disappear; on the other hand, the next forwardee is chosen among the peers who are online. Reputation management is also resistant to dropout of accountability managers, because there are m of them for each peer; m just needs to be increased if dropout is very likely. Punishment is not affected: even though a peer drops out, he will be punished with a reputation decrease all the same. However, rewarding may be problematic in the very specific case that either the update generator P_1 or the first forwardee P_2 drop out before rewarding is complete: the one of the two that remains online may not receive her/his reward.

3.4 Co-utility analysis

In this section, we demonstrate that the framework formed by Protocols 1, 2 and 3 is co-utile, that is, that those protocols will be adhered to by the players defined in Section 3.3.1.

To argue co-utility for Protocols 1, 2 and 3, we must show that following them is a better option for M and the peers than deviating.

3.4.1 Co-utility for the model manager

The model manager's goal is to train a model based on the peers' private data sets. For that reason, M is interested in encouraging good updates and punishing bad updates. On the other hand, M's role is limited to Step 4 of Protocol 1. Let us examine in detail the actions of M in that step and whether M could gain by deviating from them or skipping them:

1. In Step 4a, M directly discards an update with a probability that is inversely proportional to the reputation of the submitting peer. Discarding is only based on reputation, without examining whether the update is an outlier. M is interested to perform this step at least for two reasons: first, it reduces M's computational overhead, and second, it allows M to make reputation attractive for peers (only high-reputation peers, those with reputation at least T, are sure of getting their updates examined). At the same time, if M wants to keep the peer community alive, M should allow a nonzero probability $1 - p_0$ of examining an update submitted by a new peer (that has 0 reputation). Also, setting up a threshold T above which updates are examined for sure is a way for M of not losing too many good updates.

2. Step 4b consists of decrypting the update, checking its freshness and checking that the hash is correct. Obviously, M is interested in carrying out these steps. Without the updates, M cannot train the model.

3. Step 4c is about deciding whether an update is good or bad. M clearly needs to make this decision, in order to use good updates to improve the model and punish bad updates to discourage them.

4. Step 4d is about changing the model using the good updates. This is exactly M's main goal.

5. Step 4e publishes δ that determines the amount whereby reputations must be increased/decreased by the accountability managers. M is interested in publishing δ to facilitate a correct reputation management that keeps peers incentivized. In fact, if the number b of updates per batch is fixed, then δ is also fixed and does not need to be published at each protocol execution.

6. Step 4f publishes information that peers can use to claim rewards for good updates. If M deviates and does not publish this information, then peers cannot claim rewards. This would discourage peers from submitting good updates and would be against M's interests.

7. Step 4g launches the punishment procedure for each bad update. If M did not perform this step, bad updates would go unpunished, which would fail to discourage them.

3.4.2 Co-utility for the update generator

In Protocol 1, the update generator only works in Step 1. Let us analyze the actions in this step:

1. *Update generation and encryption.* The generator, say P_1, generates an update and encrypts it together with a random nonce so that only M can decrypt the update and check its freshness:

   (a) The intrinsic motivation for P_1 to generate an update is to have an influence on the model being learned: a rational peer wants to help obtain an accurate model that is socially beneficial in some sense, whereas a malicious peer wants to poison the learned model.

   (b) The motivation for P_1 to generate a *good* update U is to keep her reputation high. A high reputation brings more influence on the model learning. Specifically, a high g_1 allows P_1 to find P_2 such that $g_1 \geq g_2 - \alpha$, which means that P_2 does not discard P_1's update, and with g_2 high enough for P_1 to be confident that P_2 can be entrusted with relaying U towards M with little or no probability of U being discarded by M without examination (see description of the SELECT() function in Section 3.3.4). If U eventually reaches M, this brings P_1 influence and further reputation increase, which means more influence in the future.

(c) The motivation for P_1 to encrypt U under M's public key is to prevent anyone else from claiming the reward for that update, should U be good. The motivation for P_1 to sign the forwarded message is that the forwardee P_2 will not accept an unsigned message, because P_2 will need that signed message to escape punishment in case U is bad.

2. *Update forwarding.* In terms of privacy, it is bad for P_1 to submit her generated update directly to M, as it could leak information on her private data set. It is still bad if P_1 directly submits with probability $1-p$ and forwards with probability p, like in the Crowds system [45]. If we used the Crowds algorithm, from the point of view of M the most likely submitter of an update would be the update generator: U would be submitted by P_1 with probability $1-p$, whereas it would be submitted by the i-th forwardee with probability $(1-p)p^i < 1-p$. Hence, P_1 is interested in looking for a forwardee P_2 who takes care of her update, rather than submitting her update herself. Specifically, P_1 wants a forwardee P_2 such that: a) P_2 will accept to forward P_1's update; b) P_2 does not risk update discarding ($g_2 \geq T$) or risks it with the smallest possible probability (see the description of the SELECT() function in Section 3.3.4). Further, if P_1 can choose among several possible P_2 with $g_2 \geq T$, P_1's best option is to pick P_2 randomly for the sake of unlinkability of successive updates to each other. Here we see a second benefit of a high reputation for P_1: the higher g_1, the more peers with reputation at least T P_1 can choose from and the higher is unlinkability.

In Protocol 2, the update generator P_1 has a role only if her update is bad. The generator's role in this case is a passive and inescapable one: when P_1 is asked by her accountability managers to show that P_1 received the bad update from someone else, P_1 cannot show it and is punished.

In Protocol 3, the generator P_1 of a good update is clearly interested in running Step 1 of the protocol to claim a reward. In Step 1, P_1 is forced to give the first forwardee P_2 the necessary information $H(H(U, N_U))$ so that P_2 can claim his reward. The reason is that, without P_2's receipt, P_1 cannot claim her own reward at Step 3 (this latter step is also self-enforcing if P_1 wants her reward).

P_1 could certainly decide to favor a false first forwardee P_2' of her choice, rather than the real first forwardee P_2. This would still work well for P_1, because P_2' would return a signed receipt for the same reasons that P_2 would do it. However,

if P_1 wants to favor P_2', it entails less risk (of being discovered) for P_1 to use P_2' as a *real* first forwardee. Thus, there is no rational incentive to favor false first forwardees.

3.4.3 Co-utility for the update forwardees

In Protocol 1, the forwardees $P_2, P_3, \ldots$ work in Steps 2 and 3, which are analogous to each other. Let us examine the actions expected from a forwardee:

1. *Update acceptance or discarding.* The incentive for a forwardee P_i to accept to deal with an update U is to be rewarded in case U is good and P_i is the first forwardee (note that P_i does not know whether she is the first, but hopes to be). Thus, if P_i receives the update from a previous peer P_{i-1} with high reputation, P_i's rational decision is to accept that update: there are chances that U is good, which will bring reward if P_i turns out to be the first forwardee. In contrast, if U comes from a peer P_{i-1} with low reputation, it is less likely that the update is good, so P_i's rational decision is to discard U to avoid working and spending bandwidth for nothing.

2. *Update submission or forwarding.* It takes about the same effort for a forwardee P_i to submit an update to M or to forward it to some other peer P_{i+1}. Hence, it is rational for P_i to make the decision randomly according to the prescribed probabilities ($1 - p$ for submission and p for forwarding). In case of forwarding, P_i's rational procedure is like the generator's: look for a forwardee with reputation at least T if $g_i \geq T - \alpha$ or the maximum possible reputation that does not exceed $g_i + \alpha$ otherwise (as per the SELECT() function. Also, no matter whether forwarding or submitting, P_i has to replace the previous signature of the update by her own signature: neither the model manager nor any forwardee will accept from P_i a message that is not signed by P_i, because they will need the signed message in case U turns out to be bad and punishment is launched.

In Protocol 2, if P_i did not generate a bad update U, P_i will rationally do her best to avoid punishment (reputation decrease) by showing a message signed by whoever sent U to her.

In Protocol 3, P_2's best option is to return the receipt at Step 2, because P_1 could otherwise blacklist P_2 and never make P_2 a first forwardee in future epochs. Finally, P_2 is obviously interested in claiming her reward in Step 5.

3.4.4 Co-utility for the accountability managers

The accountability managers are a keystone in Protocols 1, 2 and 3. In our security model (Section 3.3.1) a majority of them is assumed to be rational and to be interested in obtaining a well-trained model. Hence, a majority of the m accountability managers pseudorandomly assigned to each peer can be expected to behave honestly, which in turn means that the reputation of every peer can be expected to be honestly managed.

In Protocol 1, there is no direct intervention of accountability managers. It suffices that they honestly maintain and supply the reputations g_i of all involved peers P_i as described in Section 3.3.3.

As to Protocol 2, it is launched at the request of M in the last step of Protocol 1. In Protocol 2, the accountability managers have the lead role. Most of each peer's accountability managers can be assumed rational and therefore they can be assumed to discharge their role as described in the protocol.

Finally, in Protocol 3, the accountability managers of the generator reward the latter in Step 4. Then in Step 6 the first forwardee is rewarded by her accountability managers. Again, since for each peer a majority of accountability managers can be assumed rational, we can expect them to honestly perform those two steps as described in Protocol 3.

Note 6 (Non-collusion scenario)**.** In fact, given that the accountability managers assigned to a peer are randomly chosen, it is reasonable to assume that in general they do not know each other and hence they do not collude. In the non-collusion scenario, not even a majority of honest accountability managers is needed. If malicious accountability managers do not collude, each of them is likely to report different reputation results. Hence, as long as *two* of the peer's accountability managers act rationally and follow the protocol, their correct result is likely to be the most frequent one and thus to prevail.

3.5 Privacy and security

In this section, we will show that the protocols satisfy the requirements of Section 3.3.2, and thereby preserve the confidentiality of the users' private data and protect the learned model from Byzantine and poisoning security attacks.

3.5.1 Privacy

As mentioned in Section 2.3.1, ensuring the unlinkability of updates goes a long way towards guaranteeing that the private data sets of peers stay confidential. We can state the following proposition.

Proposition 1. *If the forwarding probability is $p > 0$ and there is no collusion between the model manager M and peers, the private data set of each peer remains confidential versus the model manager and the other peers. Confidentiality is based on update encryption and unlinkability, and unlinkability increases with p and the generator's reputation.*

Proof. The privacy guarantee is based on unlinkability and update encryption.

Let us first consider linkability by M. By the design of Protocol 1, M knows that the submitter of an update U is never the update generator. At best, M knows that the probability that U was submitted by the i-th forwardee is $(1-p)p^{i-1}$, and hence that the most likely submitter is the first forwardee. However:

- The larger p, the greater the uncertainty about the number of hops before the update is submitted, and hence the harder for M to link a received update to its generator.

- The next forwardee is selected using the SELECT() function, described in Section 3.3.4. If $g_{gen} \geq T-\alpha$, then P_{gen} chooses the first forwardee randomly among the set of peers with reputation at least T, and this set depends on the current reputations and varies over time; hence, as long as there are several peers with reputation T or above, the fact that two updates were submitted by the same peer does not tell M that both updates were generated by the same peer. If $g_{gen} < T - \alpha$, then P_{gen} chooses as a first forwardee the peer with the maximum reputation that does not exceed $g_i + \alpha$: if reputations do not change between two successive updates, P_{gen} would choose the same first forwardee for both updates; yet, M cannot be sure that the submitter of both updates is really the first forwardee, and hence M cannot be sure that both updates were generated by the same P_{gen}. Hence, in no case can two different updates by the same generator be unequivocally linked, even if the probability of correctly linking them is lower when $g_{gen} \geq T - \alpha$.

On the other hand, *neither the reward nor the punish protocols allow M to learn who generated a good or a bad update.* Thus, M can neither link the updates he receives nor unequivocally learn who generated a certain update U. Therefore, M cannot obtain any information on the private data set of any specific peer P.

Consider now linkability by a peer P_i:

- If P_i is a forwardee for two different updates from P_{i-1} and $p > 0$, P_i does not know whether P_{i-1} generated any of the updates or is merely forwarding them. P_i's uncertainty about P_{i-1} being the generator is Shannon's entropy $H(p)$, which grows with p for $p \leq 0.5$; for $p > 0.5$, what grows with p is P_i's certainty that P_{i-1} is *not* the generator. In summary, P_i can only guess right that P_{i-1} is the generator if p is very small: in this case, forwarding hops after the first mandatory hop from generator to first forwardee are very unlikely.

- The only exception is when P_i is the first forwardee for two good updates from the same generator (because in this case he receives a message from the generator in Step 1 of Protocol 3). However, in this case P_i can only link the *encrypted* version of updates (that is, $PK_M(U, N_U)$ and $PK_M(U', N_{U'})$), but has no access to the clear updates U, U'. Hence, P_i gets no information on P_{i-1}'s private data set.

- If P_i is an accountability manager of a generator P_j, P_i can link all *encrypted* good updates originated by P_j. However, since those updates are not in the clear, P_i gets no information on P_j's private data set.

□

Note that assuming there are no collusions is plausible because peers are pseudonymous: normally people collude only with those they know.

A successful collusion must include one or more first forwardees (who know the pseudonyms of the update generators) and M (who can decrypt the updates). In this way, M can attribute updates and perhaps link those corresponding to the same generator; then M can infer whatever information on the generator's private data set is leaked by the generator's updates.

However, to allow update linkage, a collusion requires a malicious model manager and a significant proportion of malicious peers, whereas in our security model (Section 3.3.1) we assume M and a majority of peers to be rational-but-curious. A collusion of M with a substantial number of peers is hard to keep in secret, and if it becomes known that M is malicious, peers will be unwilling to help M to train the global model. Therefore, M's rational behavior is to abstain from collusion.

3.5.2 Security

Guaranteeing security means thwarting Byzantine and poisoning attacks (Section 2.3.1), which consist of submitting bad model updates. In the aforementioned section we presented the approaches that have been proposed in the federated learning literature for the model manager to defend against bad updates. In brief:

- *Detection via model metrics.* An update is labeled as bad if incorporating it to the model degrades the model accuracy. This approach requires a validation data set on which the model with the update and the model without the update can be compared. Also, the computation needed to make a decision on each received update is significant.
- *Detection via update statistics.* An update is labeled as bad if it is an outlier with respect to the other updates.
- *Neutralization via aggregation.* Updates are aggregated using operators that are insensitive to outliers, such as the median, the coordinate-wise median, or Krum aggregation. In this way, updates too different from the rest have little or no influence on the learning process.

In our protocol, we want to explicitly detect bad updates in order to avoid interaction with the malicious peers generating them. Hence, we discard methods in the third class (neutralization).

Any detection method in the two other classes can be used with our approach, including new methods that may appear in the future. Yet, detection based on model metrics is quite costly and requires validation data. For this reason, in the experimental work we have instantiated our implementation with a method based on update statistics, more specifically a distance-based method in line with [31, 32]. Given a batch of updates, this method labels as bad an update U if U is much

more distant than the rest of updates in the batch from the batch centroid C. One possible way to quantify what "much more distant" means is to check whether the distance between U and C is greater than the third quartile (or greater than a small multiple of the third quartile, say 1.5 times) of the set of distances between updates in the batch and C.

Protocols 1, 2 and 3 are designed to incentivize the submission of good updates. Thus, we can state the following proposition.

Proposition 2. *Provided that the model manager can detect bad updates, the rational behavior for generators and forwardees in Protocol 1 is to submit good updates.*

Proof. See discussion on co-utility for generators and forwardees in Section 3.4. □

As to collusions of irrationally malicious peers, they can only disrupt the learning process if they are sufficiently large so that the majority of updates received by M are bad ones *and* coordinated in the same direction. Note that uncoordinated bad updates are likely to cancel each other to some extent. Such large collusions seem hard to mount for the reasons explained in the previous section.

3.5.3 Computation and communications overhead

Let us compare the computation and communications overhead of the proposed method against alternatives based on homomorphic encryption (HE), which offer a comparable level of privacy (but cannot detect bad updates, as argued below).

HE has been used in federated aggregation mechanisms to prevent the model manager and the rest of peers in the network from having access to the individual updates of peers. In HE-based mechanisms, peers first encrypt their respective updates using an additive HE scheme (*e.g.* Paillier, [46]). Several protocols have been proposed in the literature to aggregate HE updates and decrypt the aggregated HE update [47]. Let us focus on a protocol that minimizes the number of required messages and the amount of computation (which is the most challenging benchmark when comparing with our proposed method): (i) assume a sequence of peers is defined such that the first peer sends her HE update to the next peer, who aggregates it with her own HE update and so on; (ii) after the last peer has aggregated her HE update, she sends the encrypted update aggregation to the

manager, who can decrypt it to obtain the cleartext update aggregation. In this protocol, each peer sends only one message per update, just as in plain federated learning.

Whatever the protocol used, HE-based solutions offer privacy (no one other than the peer sees the peer's cleartext update), but they *do not allow* the model manager to detect bad updates, because the manager does not see the individual updates. In this respect, HE-based solutions are inferior to our proposed method, which offers privacy without preventing bad update detection.

Even so, let us compare HE-based systems and our system in terms of computational overhead. HE-based systems require the peers to encrypt, using a public-key HE scheme, each individual model parameter at each training epoch (an update contains values for all parameters). The authors of [48] report an encryption time of 3111.14 seconds for a model with 900,000 parameters (6.87 MB) using 3072-bit Paillier (a key size of 3072 bits in factorization-based public-key cryptosystems offers equivalent security to 128-bit symmetric key schemes [49]). Expensive modular operations (with 3072-bit long moduli in the case of Paillier) for each model parameter are required to aggregate the update of each peer.

In contrast, our approach requires each peer to compute an encryption of her update using a regular non-homomorphic public-key cryptosystem, three hashes and one digital signature. With the usual digital envelope approach, regular public-key encryption amounts to encrypting a symmetric (*e.g.* AES) session key under the manager's public key, and then encrypting the bulk of the update parameters using the much faster symmetric cryptosystem under the session key. The encryption time of AES on current smartphones using AES-128-GCM is around 0.29 seconds for a model of the same size as reported above[1], to be compared with the aforementioned 3111.14 seconds of HE. Finally, the model manager just needs to decrypt the received updates and aggregate them in cleartext as in plain federated learning mechanisms (this is much faster than homomorphic aggregation in ciphertext).

Regarding the communication overhead, we first refer to the message expansion incurred by HE-based mechanisms and our proposal. As stated above, HE-based mechanisms require peers to encrypt each model parameter using an additive HE scheme. Model parameters are usually 32-bit floating point values that, when encrypted using Paillier with sufficiently strong keys, become 3072-bit integers. This

[1] AES performance per CPU core `https://calomel.org/aesni_ssl_performance.html`

implies an increase in the message size of two orders of magnitude. The proposal in [48] substantially reduces the communication requirements, but it is still one order of magnitude above plain federated learning with cleartext updates. In our proposal and thanks to the digital envelope technique, updates are encrypted using a symmetric encryption scheme, which does not expand the plaintext models (save for potential paddings, which are negligible for messages of the size we are considering). Additionally, our messages include the session key encrypted under the model manager's public key, a triple hash of the model, and a signature. This additional information increases the size of the message by approximately 6.5 KB with standard key and hash sizes, which, if we consider the example given before, amounts to a 0.09% increase in the total size of the messages.

Finally, in the HE-based protocol considered the number of messages exchanged among participants does not increase with respect to plain federated learning, *i.e.* for each training epoch there is one broadcast of the global model from the model manager to the peers and one message from each peer containing her update. In contrast, our proposal includes a forwarding mechanism, which implies that for a forwarding probability p every encrypted model hops across an expected number of forwardees equal to

$$(1-p)\sum_{i=1}^{\infty} ip^{i-1} = \frac{1}{1-p}.$$

For example, if $p = 1/2$ there are 2 additional hopping messages with respect to plain federated learning. Additionally, if each peer has m accountability managers:

- $2m+1$ messages containing one hash of the update and one digital signature of a hash value are required by the reward protocol;
- $2m$ messages, of which m are short polling messages and m contain the signed encrypted update, are required in the punishment protocol when a peer wishes to avoid punishment.

All in all, our approach requires more messages per epoch than plain and HE-based federated learning. However, whereas the message expansion in our approach is almost negligible (as the bulk of encryption is symmetric key), the HE-based approach increases message length by one or two orders of magnitude with respect to plain federated learning. In particular, if we take say $m = 3$ and $p = 1/2$, the

overall communications overhead of our approach stays below that of HE-based federated learning.

In summary, our method achieves *much* less computation overhead and less communication overhead than HE-based methods. Add to this performance advantage the functionality advantage: our method offers both privacy for peers and detection of bad updates for the manager, whereas the latter feature is lacking in HE-based methods.

3.6 Experimental results

In this section we report the results of the experiments we conducted to test how the reputations of peers evolve over time depending on whether they submit good or bad updates.

First, let us explain the expected system behavior. If our protocols are well designed, a peer's reputation should highly correlate with the probability that she generates *good* updates. Furthermore, the reputation of the peer who *submits* an update to the model manager M should also highly correlate with the probability that the peer who *generated* that update generates good updates. Since the submitting peer's reputation is used by M to decide on processing or discarding an update, M will only process a fraction of the received updates. This reduces M's overhead related to detection and punishment of *bad* updates.

Now, let us go to the actual empirical results. We bounded the range of reputations between 0 and 1. Then we built a peer-to-peer network with 100 peers whose initial reputations were set to 0. We let the network evolve for 500 iterations (or global training epochs). At each epoch, the model manager received one update from each peer. Thus, the batch size was $b = 100$ and the reward/punishment quantum was $\delta = 1/b = 0.01$. We then experimented with two test scenarios, depending on the proportion of honest peers:

- *Scenario 1.* Every peer is assigned a random *goodness probability* $\pi_g \in_R [0, 1]$. With probability π_g the peer generates a good update and with probability $1 - \pi_g$ she generates a bad update. Reputation management is used by peers to decide on accepting or rejecting a forwarded update and to choose forwardees. That is, a peer P_j accepts a forwarded update only if the requesting peer's reputation is at least $g_j - \alpha$, where we set $\alpha = 0.03$. In turn, a peer P_i chooses a forwardee based on reputations as described when

explaining the function SELECT() in Section 3.3.4. Additionally, reputation management is also used by the model manager M to decide on processing or directly discarding an update submitted by a peer P_k. That is, M directly discards the update with probability $p_0(1 - \min(g_k/T, 1))$, where we set $p_0 = 0.5$ and $T = 0.5$.

- *Scenario 2.* 90% of peers always generate good updates whereas the remaining 10% have probability 0.2 of generating good updates and probability 0.8 of generating bad updates. Hence we can say that 90% of peers have goodness probability $\pi_g = 1$ and 10% of peers have goodness probability $\pi_g = 0.2$. Like in the previous scenario, reputation management is set up by taking $\alpha = 0.03$, $p_0 = 0.5$ and $T = 0.5$.

3.6.1 Test scenario 1

In large real federated learning networks with, say several thousands or hundreds of thousands of peers (*e.g.* smartphones), a small proportion of malicious peers (even smaller than in Scenario 2) is the most realistic assumption. Nevertheless, let us study an extreme scenario with even proportions of good and bad updates. This will allow us to demonstrate that the goodness probability of a peer correlates with her reputation and with the reputations of the peers submitting her updates.

Let us assign a random goodness probability in the interval $[0, 1]$ to each of the 100 peers. Thus, on average we can expect peers to generate good updates only half of the time. Reputations are computed after each of the 500 global training epochs and are used to decide, on the one hand, on update acceptance and forwarding (peers accept updates from and forward updates to other peers depending on the flexibility parameter $\alpha = 0.03$), and on the other hand, on update processing and discarding by the model manager (it directly discards updates with probability $p_0(1 - \min(g_i/T, 1))$, with $p_0 = 0.5$ and $T = 0.5$).

Figure 3.1 displays the goodness probability versus the reputation of every peer after the 500 global training epochs. The goodness probability is represented in the abscissae and the reputation in the ordinates. It can be seen that both the goodness probabilities and their corresponding reputations spread over the entire $[0, 1]$ range. Furthermore, the peers' goodness probabilities and their reputations are highly correlated (0.977).

Figure 3.2 displays, for every update during the 500 global training epochs (50,000 updates), the goodness probability of the update generating peer versus the rep-

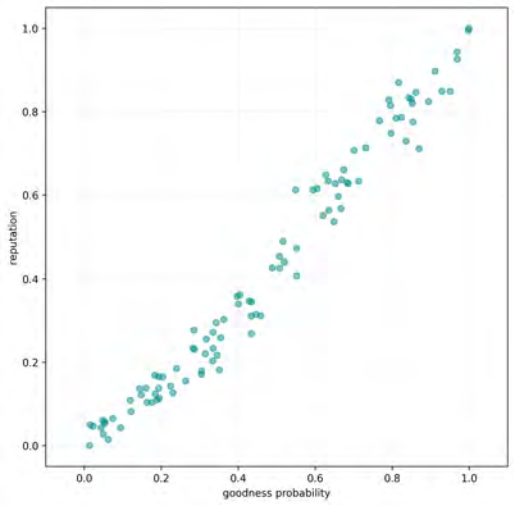


Figure 3.1: Scenario 1. Goodness probability vs reputation for each peer. Correlation: 0.977.

utation of the submitting peer. It can be seen that both values are also highly correlated (0.833). In fact, this correlation is even higher for peers with reputation below $T = 0.5$; for submitting peers with reputations $T = 0.5$ or above, the precise reputation of the submitter is not that relevant, because the model manager will process all updates submitted by peers with reputation T or above.

3.6.2 Test scenario 2

The previous scenario is highly unlikely in the real world. As said above, in large real federated learning networks a small proportion of malicious peers is the most realistic assumption.

In Scenario 2, a clear majority of 90% of peers are completely honest (goodness probability $\pi_g = 1$), whereas the remaining 10% have a goodness probability of only $\pi_g = 0.2$. Reputations are computed after each epoch and are used to decide, on the one hand, on update acceptance and forwarding, and on the other hand, on update processing and discarding by the model manager.

Figure 3.3 displays the goodness probability against the reputation of every peer after 500 global training epochs. Malicious peers (those with $\pi_g = 0.2$) are correctly assigned low reputations, because most of the updates they generate are

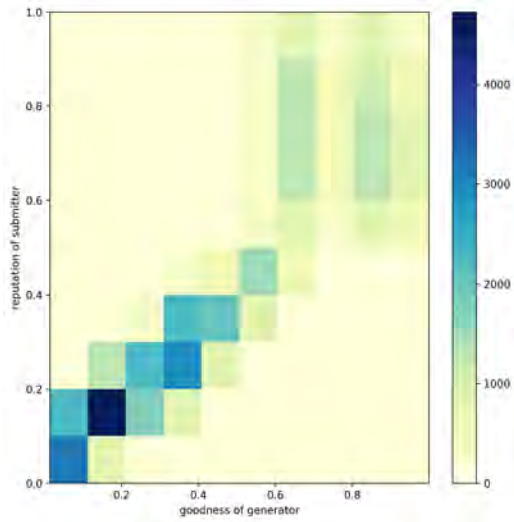


Figure 3.2: Scenario 1. Generating peer's goodness probability vs submitting peer's reputation, for all updates. The color scale indicates the number of peers in each 2-dimensional interval. Correlation: 0.838.

bad and they are punished when their updates reach the model. Besides that, it is hard for such peers to be selected as forwardees of good updates and thereby improve their reputation. On the other side, all honest users (those with $\pi_g = 1.0$) achieve high reputation values that correspond to their good behavior. Peers with a reputation $T = 0.5$ or above are part of a "community" whose members improve the reputations of each other, by forwarding or submitting their respective updates.

The evolution of the reputations of good peers (with $\pi_g = 1.0$) and bad peers ($\pi_g = 0.2$) is shown in Figure 3.4. The average reputations of both types of peers swiftly diverge from the very beginning.

Figure 3.5 displays, for every update during the 500 global training epochs (50,000 updates), the goodness probability of the generator versus the reputation of the submitter. Both values are highly correlated (0.799). However, the correlation is higher after the system stabilizes (0.9854 from epoch 100 onwards) and all good peers reach high reputations. Initially, reputations have not yet adjusted and hence the updates generated by good peers can be submitted by peers with reputation only slightly above or even slightly below T.

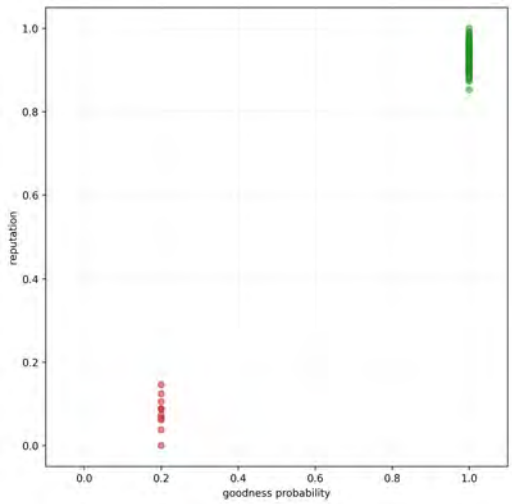


Figure 3.3: Scenario 2. Goodness probability vs reputation for each peer. Honest peers are shown in green while bad peers are shown in red. Correlation: 0.998.

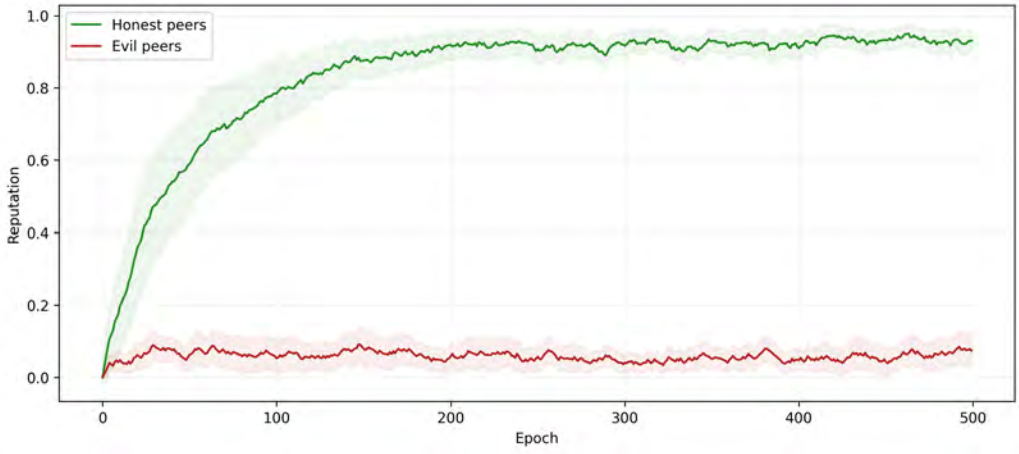


Figure 3.4: Scenario 2. Evolution of the average (depicted as a line) and the standard deviation (depicted as a gray band) of the reputations of good peers and bad peers as a function of the epoch.

Finally, observe in Figure 3.6 the effectiveness of making reputation-based decisions to filter out bad updates. Out of the 50,000 updates generated over the

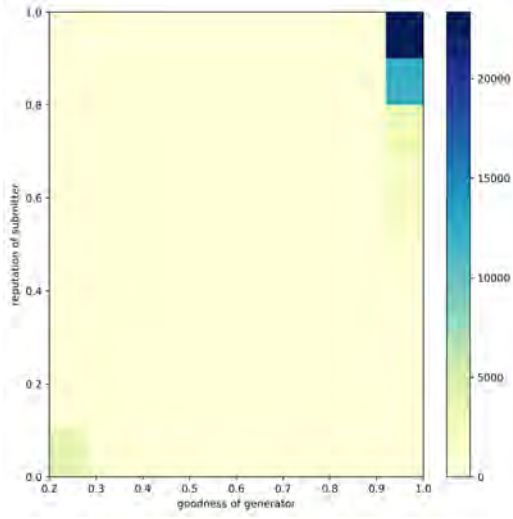


Figure 3.5: Scenario 2. Generating peer's goodness probability vs submitting peer's reputation, for all updates. The color scale indicates the number of peers in each 2-dimensional interval. Correlation: 0.799.

500 epochs, around 46,000 are good, while around 4,000 are bad. Based on the submitting peer's reputation, the model manager M discards 2,831 updates. The figure shows that, when the system stabilizes, on average 80% of the updates discarded by M are bad. This is the right proportion, because malicious peers do not always generate bad updates (they generate bad updates with probability $1 - \pi_g = 0.8$).

Note that reducing the proportion of bad updates processed by the model manager is also a good security defense. Indeed, the fewer the bad updates processed by the model manager, the more those bad updates are likely to stand out as outliers, which will enable M to detect and discard them. Additionally, fewer bad updates processed by M also mean less detection overhead for M and, especially, less punishment and tracing overhead for peers (both normal peers and accountability managers).

Finally, we considered the case of newcomers by introducing two new peers immediately after iteration 100: A good one and a bad one. Figure 3.7 shows that the reputation of the good newcomer gradually increases to the level of the other good peers, while the reputation of the bad newcomer remains as low as the other

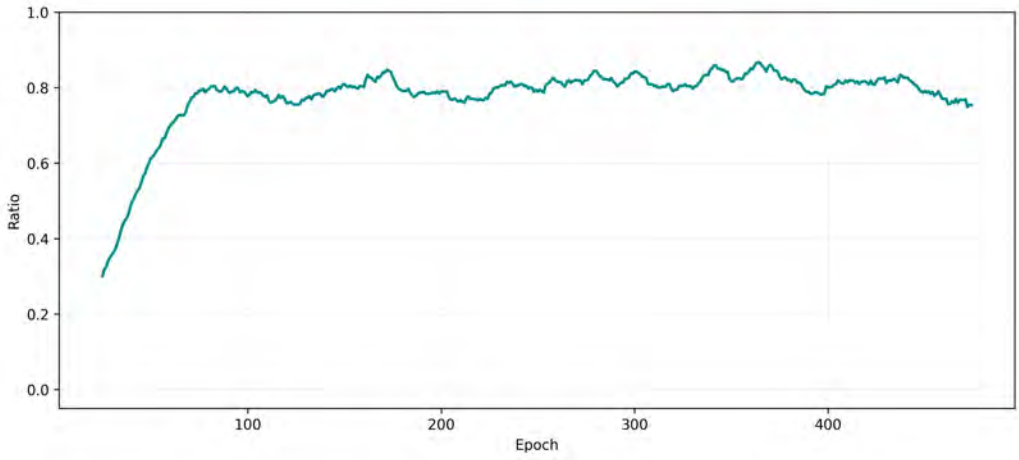


Figure 3.6: Scenario 2. Ratio of bad updates discarded by the model manager as a function of the training epoch.

bad peers in the network. Therefore, a bad peer does not gain anything by leaving the system and rejoining with a new pseudonym (whitewashing).

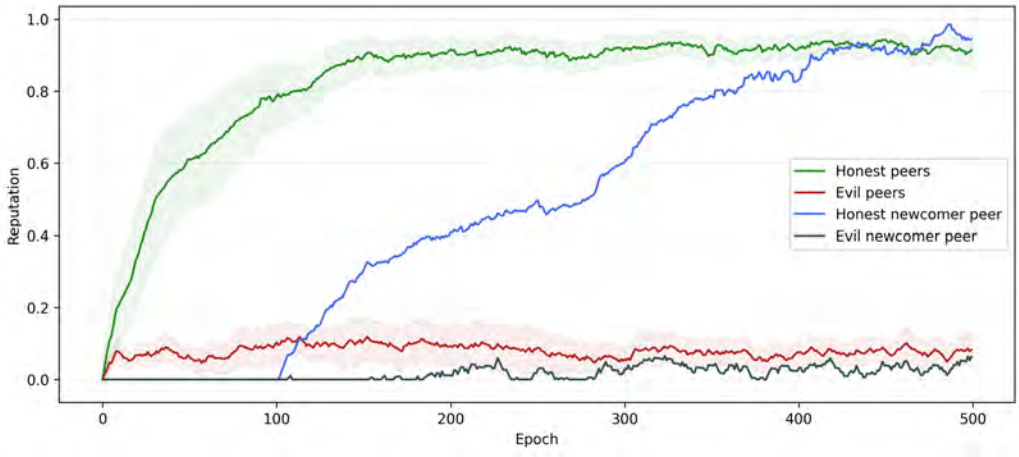


Figure 3.7: Scenario 2. Evolution of the reputation of newcomers.

3.7 Summary

We have presented protocols to improve privacy and security in federated learning while perfectly preserving the model accuracy. Our protocols rely on the notion of co-utility, that is, they are self-enforcing if players are rational. We use a decentralized reputation management scheme that is itself co-utile to incentivize peers to adhere to the prescribed protocols.

In this way, peers do not need to be honest-but-curious *per se*: as long as they are rational they will behave honestly, and even a minority of malicious peers that do not respond to the same incentives as the other peers can be tolerated. Confidentiality of the peers' private data is guaranteed by the unlinkability of updates: when a peer generates an update, neither the model manager nor the other peers can identify the update generator. This way to provide privacy is superior to the state-of-the-art alternatives:

- Unlike privacy protection via differential privacy [28], our protection mechanism does not alter the value of updates and hence does not affect the accuracy of the learned model. Furthermore, our privacy notion based on unlinkability is also strong.
- Unlike privacy protection based on update aggregation, our solution is compatible with punishing the peers that generate bad updates. Also, our solution entails less computational overhead than aggregation based on homomorphic encryption.

Security, *i.e.* protection against bad updates, is pursued in our approach via reputation. Whereas state-of-the-art security countermeasures do nothing to reduce the number of bad updates that are processed by the model manager, we address this issue in a way to achieve two beneficial effects: first, to decrease the overhead for the model manager and the peers related to processing, tracing and punishing bad updates; and, second, to make the (fewer) bad updates processed by the model manager more identifiable as outliers. The design of our protocols also renders whitewashing and Sybil attacks ineffective.

Chapter 4

Secure, Accurate and Privacy-Aware Fully Decentralized Learning via Co-Utility

4.1 Introduction

Fully decentralized learning (FDML) is the extreme form of decentralized machine learning [20, 50, 51, 21, 52]. In FDML, each peer in a peer-to-peer (P2P) network trains a deep learning model with the help of the other peers. In her role as a model manager, each peer periodically sends the current model to other peers and the latter return model updates based on the private data they hold.

Unlike in the related federated learning (FL, [2]), where there is a fixed split between a model manager and the workers computing model updates based on their private data, in FDML all nodes may be managers of their own models and workers for others' models and exchange messages without any central coordination.

The strong point of FDML is that it is more robust and scalable as the number of nodes increases. Note that in centralized machine learning, respectively in FL, an

attacker who wants to disrupt the learning process only needs to "intoxicate" the model manager with bad data, respectively bad updates. In contrast, in FDML it is harder for the attacker to disrupt a significant proportion of learning processes.

In FDML a conflict between accuracy, security and privacy arises. On the one hand, model updates returned by a peer can leak some of the peer's private data [27]. A countermeasure that can be implemented by the peers themselves to protect their privacy is to distort their updates via, *e.g.*, differential privacy [29]. Alas, this distortion works against accuracy of the trained model. Another countermeasure consists in securely aggregating the updates of several peers (*i.e.*, via secure multiparty computation) and then sending only the update aggregation to the model manager [33]. However, this goes against security, because the model manager cannot filter out individual bad updates. Additionally, peers are autonomous and it cannot be taken for granted that they will help the model manager by supplying honest model updates. This may compromise the sustainability of the network.

4.2 Contributions

In this chapter we present a co-utile approach (see Section 2.2) to fully decentralized machine learning that has the following properties:

- Perfectly accurate individual updates can be returned by peers to the model manager in a privacy-preserving manner.
- Rational (that is, self-interested) peers can be expected to follow the FDML protocol we propose without deviating, thanks to a tit-for-tat mechanism that renders the protocol naturally co-utile. This makes our proposal sustainable and robust against rational security attacks.

In the previous chapter we presented a co-utile protocol for federated learning. In order to preserve the peer-to-peer (P2P) computing paradigm, global reputations were also computed in a P2P decentralized way and their correctness was ensured by co-utility as well. However, decentralized management of global reputations complicates protocols and entails a significant cost: in Chapter 3 reputation management requires any reputation updates for a peer to be sent to the peer's set of AMs. Moreover, to obtain consistent global reputations, values are normalized into the range $[0, 1]$ by dividing them by the largest reputation in the network; to

that end, at each epoch AMs need to broadcast all reputation values greater than 1.

The fact that we tackle here fully decentralized machine learning, makes the complexity of the above reputation management even more unaffordable.

Our experiments show that peers receive good updates for their respective models from other peers if and only if they compute good updates for the other peers' models. Furthermore, our approach based on tit-for-tat incurs less overhead to achieve co-utility than protocols based on decentralized global reputation.

The chapter is organized as follows: Section 4.3 introduces a co-utile protocol for FDML. Section 4.4 justifies that the proposed protocol suite achieves co-utility (and thus is rationally sustainable). Section 4.5 shows that it satisfies privacy for the inputs and the outputs, and that correct computation is the best option for rational peers. Section 4.6 presents experimental results. Finally, Section 4.7 summarizes conclusions and future research avenues.

4.3 Co-utile FDML based on tit-for-tat

Global reputation-based incentives for workers are needed in federated learning, due to the divide between the model manager and workers: since all workers contribute to a single model, there should be a common –global– way to account for their behavior. However, in fully decentralized learning peers can be managers of their own models and at the same time be workers for other peers' models. In this symmetric setting, co-utility can be attained in a more efficient way, specifically using tit-for-tat and local reputations. This is the idea underlying Protocol 1. Before starting the protocol description, we summarize in Table 4.1 the notation used in the rest of this chapter.

In Protocol 1, every time a peer P_i receives a model m_k from another peer P_k, P_i computes her update u_{ki} on m_k. Such an update can be viewed as a "currency" that can be redeemed at P_i against an update on P_i's model m_i. It is important to note that, when P_i sends her model m_i, m_i carries P_i's identity so that every peer receiving m_i knows it is managed by P_i. In contrast, when P_i returns an update u_{ki} on m_k, u_{ki} carries P_k's identity but *not* the identity of the update generator P_i. In fact, the protocol prevents P_k from directly receiving u_{ki} from P_i, in order to protect the update generator's privacy. Indeed, u_{ki} might leak information on P_i's private data, and hence dissociating u_{ki} from P_i's identity protects the latter peer's privacy.

Table 4.1: Notation in this Chapter

Notation	Concept	Notation	Concept
P_i, P_k, P_l, P_ℓ	Peers	i, k, l, ℓ	Pseudonyms of P_i, P_k, P_l, P_ℓ, respectively
m_i	Model managed and learned by P_i	u, u_{**}	Model updates
m^u	Model m after updating it with update u	u_{ki}	Update on P_k's model computed by P_i
u_{i*}	Update on P_i's model computed by some other peer	t_i	Number of updates computed by P_i on her private data
κ_i	Privacy parameter such that there is a probability $1/\kappa_i$ that an update exchanged by P_i has been computed by P_i on her private data	θ	Centroid of a batch of updates
$Dest(\cdot)$	Function that computes the pseudonym of the peer to whom the message in the argument is to be sent	$Sign_i(\cdot)$	Function computing P_i's signature on the argument
$Ver_i(\cdot,\cdot)$	Verification function for P_i's signature	$H(\cdot)$	Cryptographic hash function
$Enc_i(\cdot,\cdots)$	Function that encrypts the message and the nonce in the arguments under P_i's public key	R_i	Request for updates message sent by P_i
τ	Epoch identifier	$r, r_2, r^*, \cdots$	Random nonces
$U^i_{j\to k}$	Update message that contains $Enc_i(u_{ij})$ signed and sent by P_j to P_k	$U^i_{j\to *}$	Same as above but sent by P_j to any peer
$U_{u_{i*}}$	Set of update messages containing u_{i*}	$S_{u_{i*}}$	Set of peers having sent update messages in $U_{u_{i*}}$
$C_i = \{c_{i1}, ..., c_{in}\}$	Local reputations managed by P_i on the other peers she has interacted with	δ	Reputation quantum for punishment and reward
T_i	Interaction threshold, which is used by P_i to decide whether to interact with other peers		

To account for the actions of the peers involved in the interactions, Protocol 1 relies on local reputations: each peer P_i maintains local reputation scores $C_i = \{c_{i1}, ..., c_{in}\}$, which represent P_i's opinion on the other peers she has interacted with. These local reputations are never shared. Hence, the interactions between two peers depend only on their corresponding local reputations.

Local reputations take values within the range $[0, 1]$. Depending on the actions of P_k during the execution of Protocol 1, P_i's opinion on peer P_k (c_{ik}) can either improve or worsen. Generating a bad update for P_i or duplicating updates (to save computation) decreases the generator's reputation c_{ik} by a certain quantum δ (*hard punishment*). Furthermore, unintentionally forwarding bad or duplicated updates to P_i slightly decreases the forwarder's reputation c_{ik} by a fraction of δ (*soft punishment*). On the other hand, helping a good update reach the model manager or helping disseminate the model brings a reward (fraction of δ) to one of the helping peers.

After each tit-for-tat phase, every peer P_i computes an interaction threshold T_i. This threshold defines the minimum reputation value P_i requires to another peer in order to *trust* him and, therefore, interact with him. The threshold is computed as the average of the local reputations managed by a peer minus their standard deviation:

$$T_i = \overline{C_i} - \sigma(C_i). \tag{4.1}$$

In what follows, we will use the term *trusted* (by a certain peer P_i) to denote a peer P_k with a local reputation equal or above T_i; that is, "P_i trusts P_k" means $c_{ik} \geq T_i$. Inversely, we will use the term *untrusted* for a peer P_l such that $c_{il} < T_i$.

Protocol 1 works as follows. At each epoch, each P_i disseminates her model m_i to some of the others peers, trusted or not (subprotocol DISSEMINATE(P_i,m_i)). Since P_i would anyway be unable to prevent her model from reaching peers untrusted by her (because dissemination is multi-hop), it makes no sense for P_i to limit dissemination to her trusted peers.

After disseminating her model, each peer P_i computes updates based on her local private data for as many models as the peer has received from trusted peers.

Next come two tit-for-tat blocks:

Protocol 1: Co-utile FDML based on tit-for-tat

Peer P_i calls DISSEMINATE(P_i,m_i);
for *each model* m_k *received* **do**
  if P_i *trusts* P_k **then**
    P_i computes an update u_{ki}
Let t_i be the number of updates computed by P_i;
`/* Privacy tit-for-tat */`
for $j = 1$ **to** $\kappa_i \times t_i$ **do**
  P_i randomly selects a P_l such that $c_{il} \geq T_i$;
  P_i calls EXCHANGE_UPDATE(P_i,P_l);
P_i waits until the first half of the epoch has elapsed;
P_i hard-punishes any peer to whom P_i has sent an update in the privacy tit-for-tat without being reciprocated with another update from that peer;
`/* Learning tit-for-tat */`
for *each* u_{k*} *received by* P_i *from a trusted peer and not yet forwarded* **do**
  if $c_{ik} \geq T_i$ **then**
    P_i tells P_k she has an update on P_k's model and requests an update from P_k, if possible an u_{i*} on P_i's model;
    if P_i *receives an update from* P_k **then**
      P_i sends u_{k*} to P_k;
for *each good update* u_{i*} *received by* P_i *from a trusted peer* P_j **do**
  P_i updates her model m_i with u_{i*}, where in case of duplication u_{i*} is considered only once;
  P_i rewards P_j;
  P_i also rewards the first destination peer that participated in the model dissemination;
P_i hard-punishes any peer to whom P_i has sent an update in the learning tit-for-tat without being reciprocated with another update from that peer;
for *each* P_k *from whom* P_i *has received a bad* u_{i*} **do**
  P_i calls PUNISH_BAD(P_i, u_{i*}, P_k);
for *each duplicate update* u_{i*} **do**
  P_i calls PUNISH_DUPLICATE(P_i, u_{i*}).

- Up to the end of the first half of the epoch, a "privacy tit-for-tat" is run. In parallel, every peer P_i peer calls subprotocol EXCHANGE_UPDATE(P_i,P_l) to exchange the updates she holds with other trusted peers P_l. The updates that a peer P_i holds may have been computed by P_i or may have been just received from other trusted peers (P_i systematically discards updates received from untrusted peers due to their demonstrated bad behavior in previous epochs).

  Hence, when a peer P_l receives an update from P_i, P_l cannot be sure that the update was really computed by P_i, which protects the privacy of P_i's private data. See the details of EXCHANGE_UPDATE in Section 4.3.2 below.

  If a peer P_i has computed t_i updates, she tries to exchange $\kappa_i t_i$ updates, where κ_i is a privacy parameter selected by P_i.

  As a result, the probability that an update exchanged by P_i has actually been computed by P_i on her private data is at most $1/\kappa_i$; thus, the greater κ_i, the more work for P_i, but also the more privacy.

- The second tit-for-tat is called "learning tit-for-tat" and starts in the second half of the epoch. In parallel, peers send to the corresponding trusted model managers all the updates that: (i) they got from trusted peers; and (ii) they did not forward in the privacy tit-for-tat.

  Due to the first tit-for-tat, the peer P_i that sends an update u_{k*} to P_k is not in general the peer having computed the update, which is good for privacy. On the other hand, if P_i sends an update u_{k*} to P_k (tit), P_i expects to receive from P_k an update that, ideally, should be u_{i*} (tat). However, if P_k does not have any u_{i*}, P_i will also accept any other update from P_k, say u_{l*}. This will enable P_i to contact another model manager (P_l). By repeating this exchange of updates, P_i increases the chance of receiving updates u_{i*} on her own model. At a network level, this will make it possible for most updates to reach their corresponding model managers.

  Before sending u_{k*} to P_k, P_i asks P_k whether she has got any update for her (ideally u_{i*}).

  If P_k sends an update to P_i, P_k expects to receive u_{k*} from P_i; if this does not happen, then P_k hard-punishes P_i by decreasing her local reputation drastically.

If P_k is not able to send any update to P_i (because she has run out of them), P_i selects another update to be exchanged with its corresponding model manager, and keeps u_{k*} to be exchanged later with other peers holding u_{i*}.

At the end of the epoch, P_i checks whether the updates u_{i*} she has received as a model manager are good (see below). Then P_i uses each good update to improve her model m_i. *If a good update has been received several times (duplicate), it is only used once.* Also, P_i rewards the last peer that forwarded the update, and the first destination peer that helped disseminate the model.

Finally, P_i launches punishment protocols involving those peers P_k from whom P_i has received bad or duplicate updates u_{i*}. The peers who computed these updates will eventually be hard-punished. The reason is that bad or duplicate updates correspond to malicious behaviors that should be discouraged: bad updates can bias the model or prevent it from converging [23], whereas duplicate updates signal that some peer is trying to avoid the cost of calculating updates on her own private data[1].

After the two tit-for-tat blocks, each peer will have received a number of updates for her own model that is highly correlated with the number of updates computed by her for the models of other peers (see Figure 4.10 in the experimental section). In this way, the protocol fosters the computation of new updates.

Figure 4.1 shows an example of the execution of Protocol 1. P_0 and P_3 disseminate their respective models, which reach P_1 and P_3 in the case of P_0's model, and P_0, P_1 and P_2 in the case of P_3' model. Each peer computes an update and, afterwards, the "privacy tit-for-tat" phase begins. In this phase, the updates are exchanged among the peers and, at the end, P_0 owns u_{31}, P_1 owns u_{32} and u_{03}, P_2 owns u_{30} and P_3 owns u_{01}. For the sake of clarity, in the "learning tit-for-tat" phase we have just shown the interaction between P_0 and P_3. Since P_0 trusts P_3, P_0 tells P_3 that she has an update for her, P_3 sends u_{01} to P_0 and finally, P_0 sends u_{31} to P_3. The remaining updates would be exchanged among the other

[1] One might also think of "smart duplicators", who slightly change another peer's update to forgo calculation costs while escaping the duplicator punishment. However, this behavior is not possible because, as it will be seen below, updates are encrypted under the model manager's public key, and hence if they are altered in transit, no proper update will be recovered by the manager, who will count the resulting gibberish as a bad update and will punish the "smart duplicator" for having generated it.

peers in a similar way. After receiving an update, each peer should check it. In our example, P_0 checks u_{01}. If u_{01} is a good update, P_0 updates her model and rewards both the first disseminating peer (not shown in the diagram, say P_x) and the last forwardee (P_3). If u_{01} is a bad update or it has been duplicated, P_0 launches the corresponding punishment protocol.

In the next subsections, we describe the auxiliary (sub)protocols that appear in Protocol 1, namely DISSEMINATE, UPDATE_EXCHANGE, update forwarding, PUNISH_BAD, and PUNISH_DUPLICATE.

4.3.1 Model dissemination

To describe the DISSEMINATE subprotocol, we first define some notation.

Addressing in our P2P network is designed to function as a distributed hash table (DHT) using consistent hashing [53], rendezvous hashing [54], or similar approaches. In other words, all peers in the network are able to determine, using an agreed-upon cryptographic hash function denoted by $Dest(\cdot)$, which peer is responsible for processing a specific message. Additionally, peers are identified by a pseudonym consisting in an integer in the range $(0, 2^\lambda - 1)$ (*e.g.* for $\lambda = 160$ we have a 160-bit integer). We use $Sign_i(\cdot)$ to denote a function computing P_i's signature on the argument using her private key. Likewise, we denote by $Ver_i(\cdot, \cdot)$ the signature verification function for P_i's signature using P_i's public key. For correctness, $Ver_i(m, Sign_i(m)) = 1$ for any message m. Note that the identifier i of peer P_i is deterministically linked to P_i's public key, and so all peers are able to obtain P_i's public key from her identifier. Finally, $H(\cdot)$ is a cryptographic hash function.

DISSEMINATE(P_i, m_i) works as follows:

1. Peer P_i constructs a request for updates message

$$R_i = \langle i, \tau, m_i, Sign_i(H(i, \tau, m_i)), r \rangle, \tag{4.2}$$

   where i is P_i's pseudonym, τ is an epoch identifier, m_i is P_i's model, and r is a random nonce.

2. P_i computes $j = Dest(R_i)$.

3. P_i sends R_i to $P_{Dest(R_i)}$ who, upon receiving the message, checks that (i) she is the intended receiver by ensuring her identifier matches $Dest(R_i)$, (ii)

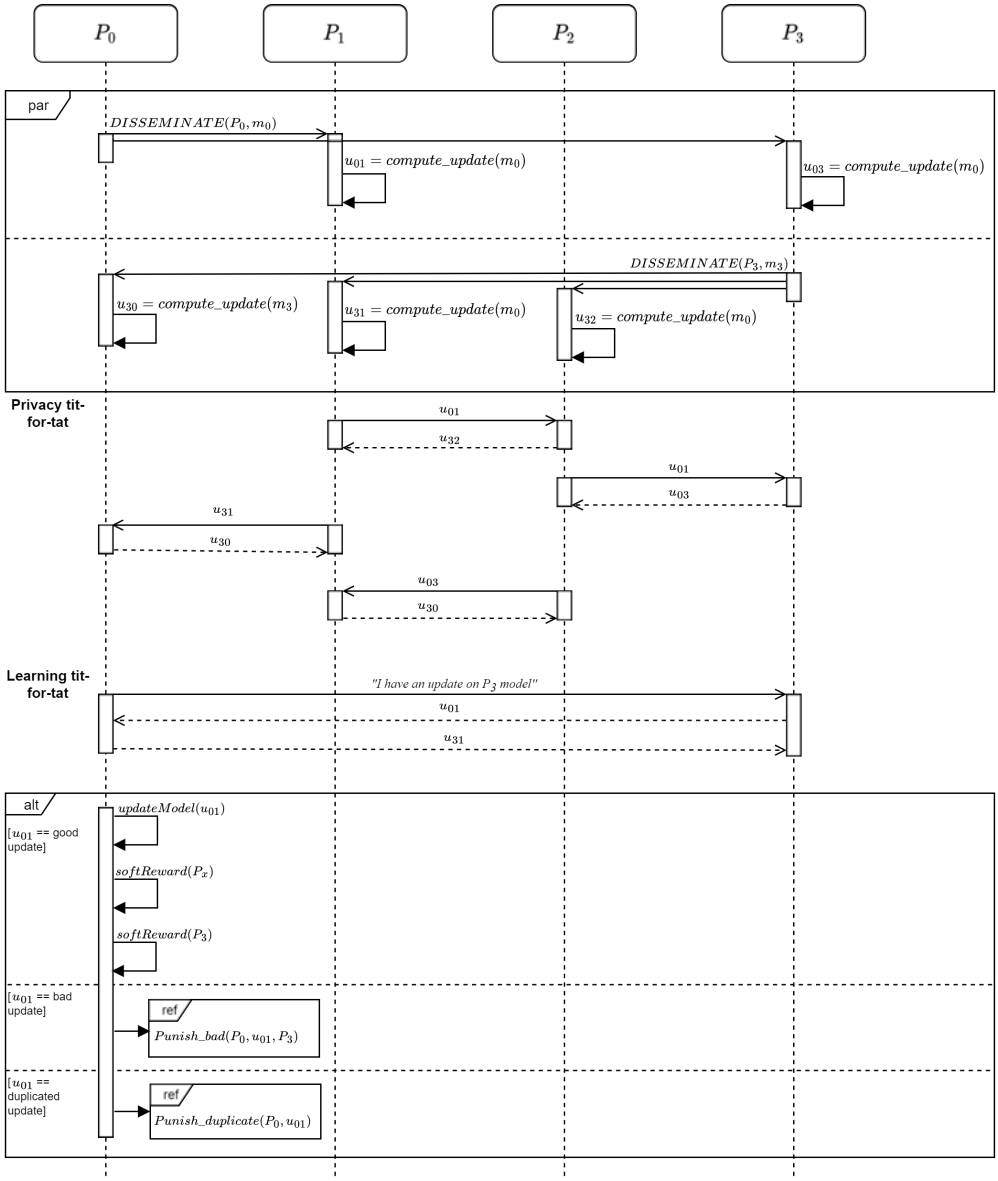


Figure 4.1: Sequence diagram showing the interactions among peers in an example execution of Protocol 1

the epoch identifier τ is correct, (iii) she has not received another request for updates for the same model m_i in the current epoch, and (iv) the signature on i, τ and m_i is valid.

4. If any of the previous checks fails, $P_{Dest(R_i)}$ drops R_i.
   Otherwise, $P_{Dest(R_i)}$ appends a new sequence number r_2 to obtain $R_i^2 = \langle i, \tau, m_i, Sign_i(H(i, \tau, m_i)), r, r_2 \rangle$, and then forwards R_i^2 to $P_{Dest(R_i^2)}$. Nothing prevents $P_{Dest(R_i)}$ from disseminating m_i to additional destination peers, by replacing r_2 with other sequence numbers.

The rationale of the DISSEMINATE subprotocol is as follows.

- The identifiers and the public keys of peers are linked to each other. Hence, peers cannot maliciously alter their identifiers.
- Due to the above and to the use of a cryptographic hash function (resistant to preimage attacks), a malicious peer P_i is prevented from choosing a designated target peer P_j as a destination for P_i's requests for updates (in view of attacking P_j's privacy by obtaining updates computed on P_j's private data). Indeed, R_i contains i, τ, m_i, a signature on i, τ and m_i, and a nonce r. Since i and τ cannot be altered and the signature has to be valid for the message to be accepted by P_j, P_i needs to find either a format-conforming m_i^*, a nonce r^*, or a combination of both such that $j = Dest(i, \tau, m_i^*, Sign_i(H(i, \tau, m_i^*)), r^*)$, which is computationally infeasible for cryptographically secure hash functions. The same holds for the second hop.

4.3.2 Update exchange

In the EXCHANGE_UPDATE(P_i,P_l) subprotocol, a peer P_i seeks to exchange an update with another peer P_l that is trusted by P_i.

When exchanging updates during the privacy tit-for-tat, *peers want to avoid receiving repeated updates*, that is, updates the peers had themselves generated or received from other peers in previous exchanges. Note that avoiding repeated updates also avoids loops in the path from the update generator to the model manager for whom the update is intended.

The reason to avoid repeated updates is that it is dangerous for a peer to forward an update more than once. Indeed, forwarding more than once due to a loop can hardly be distinguished from forwarding more than once due to malicious update duplication. In fact, subprotocol PUNISH_DUPLICATE (Section 4.3.5) punishes any peer that forwards the same update more than once.

Therefore, UPDATE_EXCHANGE(P_i,P_l) operates in the following way:

1. P_i shows to P_l the hash images $H(u)$ of all updates u computed by P_i or received by P_i, such that u has not been forwarded yet.

2. P_l shows to P_i the hash images $H(u')$ of all updates u' computed by P_i or received by P_i, such that u' has not been forwarded yet.

3. If P_i finds a hash $H(u_{**})$ among those shown by P_l that does not correspond to any update previously held by P_i, then P_i requests to P_l the update u_{**}.

4. If P_l finds a hash $H(u'_{**})$ among those shown by P_i that does not correspond to any update previously held by P_l, then P_l requests to P_i the update u'_{**}.

As specified in Line 10 of Protocol 1, if a peer does not reciprocate in the above exchange subprotocol, she will be hard-punished by the other peer at the end of the privacy tit-for-tat.

4.3.3 Update forwarding

Both the privacy tit-for-tat and the learning tit-for-tat require the exchange of model updates among peers. In order to protect the privacy of the model update generators against the peers in the update's path to the model manager, a peer P_j that computes an update u_{ij} of model m_i owned by P_i encrypts u_{ij} under P_i's public key.

Preserving the update generator's privacy as just discussed must be compatible with being able to trace bad or duplicate updates during the punishment subprotocols. To do so, every update sent by a peer P_j to a peer P_k includes the identities of both peers and is signed by P_j. In this manner, if the update contained in the message is flagged as suspicious, P_k can prove that she did not compute the update, but just forwarded it. The only peer that cannot show a signature from a previous peer is the update generator. Thus, an update message containing an

update for the model m_i managed by P_i, sent by peer P_j to P_k at epoch τ and timestamp T takes the form

$$\begin{aligned} U^i_{j\to k} = \langle & j, k, i, \tau, T, \\ & Enc_i(u_{ij}, r), \\ & Sign_j(H(j, k, i, \tau, T, r, Enc_i(u_{ij,r})))\rangle \end{aligned} \tag{4.3}$$

In Expression (4.3), r is the nonce inserted during the dissemination phase in the request-for-updates message R_i to which the update corresponds (see Expression (4.2)). Upon receiving a good update $U^i_{j\to k}$, the model manager P_i can reward the first destination peer $Dest(R_i)$ that took care of forwarding R_i rather than dropping it. The reward consists in a small increase (by a fraction of δ) of $Dest(R_i)$'s local reputation.

4.3.4 Punishing bad updates

In this section, we describe the PUNISH_BAD subprotocol for punishing bad updates. In Section 2.3.1 we discussed the approaches that a model manager can use to decide whether a received update is good or bad. In a nutshell:

- *Detection via model metrics* An update is labeled as bad if incorporating it to the model degrades the model accuracy. This approach requires a validation data set on which the model with the update and the model without the update can be compared. Also, the computation needed to make a decision on each received update is significant.
- *Detection via update statistics.* Given a batch of updates, an update is classified as bad if it is much more distant than the other updates from the centroid of the batch.
- *Neutralization of bad updates via special aggregation.* This approach consists in ignoring bad updates rather than seeking to explicitly detect them. It combines the above intuitions of distance-based detection and majority voting. Updates are aggregated using operators that are insensitive to outliers, such as the median, the coordinate-wise median, or Krum aggregation. In this way, updates too different from the rest have little or no influence on the learning process.

In our protocol, we want to actually detect bad updates in order to avoid interaction with the malicious peers generating them. Hence, we discard the third approach based on ignoring bad updates. On the other hand, the approach based on model metrics is quite costly and requires validation data. Thus, resorting to update statistics seems the most suitable option in our case.

The PUNISH_BAD(P_i, u_{i*}, P_k) subprotocol is called by P_i upon receiving a bad update u_{i*} from another peer P_k and it works as follows:

1. P_i asks P_k to prove that P_k did not generate u_{i*}.
2. If P_k did not generate u_{i*}, according to the update forwarding subprotocol (Section 4.3.3), P_k should have received a message $U^i_{l\to k}$ with the structure of Expression (4.3) and containing u_{i*} from another peer, say P_l. Now:

   P_k calls PUNISH_BAD(P_k, u_{i*}, P_l);

   If P_k decides to help P_i (*e.g.*, because P_k trusts P_i), then P_k sends $U^i_{l\to k}$ to P_i.
3. If P_i receives a proof that P_k did not generate u_{i*}, then P_i soft-punishes P_k by slightly decreasing (by a fraction of δ) her local reputation c_{ik}.
   Else P_i hard-punishes P_k by decreasing P_k's local reputation by δ.

If P_k proves she did not generate the bad update by showing $U^i_{l\to k}$ to P_i, P_k avoids a severe decrease of her local reputation c_{ik}. Yet, c_{ik} is decreased a little. The rationale is to have P_k share some of the damage caused by the bad update, so that P_k has an incentive to launch punishment against P_l. In this way, punishment hops up to the generator of the bad update, who cannot prove she got it from anyone else, and is therefore hard-punished by the peer who directly got the bad update from the generator.

Resuming the example in Figure 4.1, Figure 4.2 shows the case where the update received by P_0 (u_{01}) was bad. P_0 asks P_3 to prove that she did not generate u_{01} and P_3 returns the message $U^0_{2\to 3}$ received from P_2 to avoid being hard-punished. P_0 soft-punishes P_3 and P_3 executes the same protocol with P_2. P_2 can also prove she was not the generator of u_{01} and try to execute the same protocol with P_1. Since P_1 was the generator of the bad update, she cannot send a message $U^0_{*\to 1}$ to P_2 and, therefore, P_1 is hard-punished.

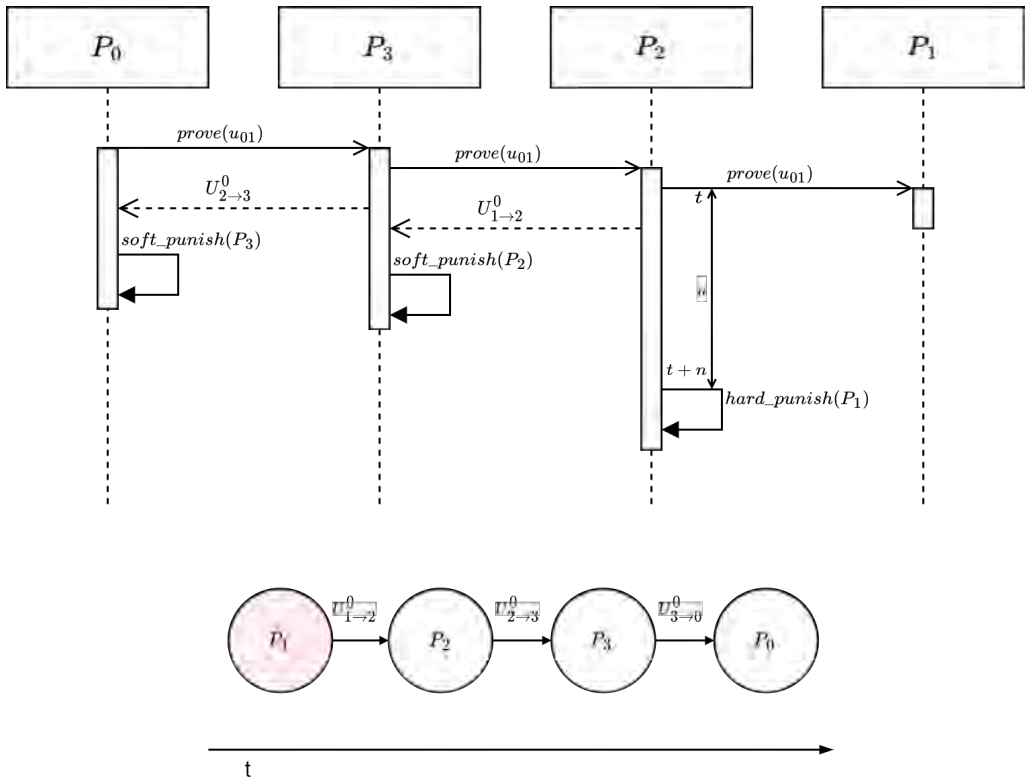


Figure 4.2: Sequence diagram showing the interactions among peers in an example execution of the PUNISH_BAD subprotocol. The diagram at the bottom shows the path followed by the update.

An alternative to the above hop-wise punishment would be for the model manager P_i to trace and punish the generator upstream with the help of the intermediate forwarders. While this procedure would allow the model manager P_i to directly punish the generator, it might also be misused: P_i might pretend that a good update is bad in order to find which peer generated it and thereby break the generator's privacy (to the extent that an update can leak information on the generator's private data). A peer P_l can defend against this potentially misusable alternative by never answering bad update claims for another peer P_i to whom P_l did not directly send the update.

4.3.5 Punishing duplicate updates

PUNISH_DUPLICATE(P_i, u_{i*}) aims at punishing the most downstream duplicator(s), that is, the peer(s) closest to P_i in the update forwarding path who duplicated an update to try to save her (their) own computing resources.

PUNISH_DUPLICATE traces the duplicator(s) upstream. It works as follows:

1. If u_{i*} is a duplicate, this means that P_i has received a set of messages $\mathbf{U}_{u_{i*}} = \{U^i_{\ell\to i} | U^i_{\ell\to i}$ contains $u_{i*}\}$ such that $|\mathbf{U}_{u_{i*}}| \geq 2$. Let $S_{u_{i*}} = \{P_\ell | U^i_{\ell\to i} \in \mathbf{U}_{u_{i*}}\}$.
2. Non-questioned_peers $= S_{u_{i*}}$.
3. While Duplicator= not_found and Non_questioned_peers $\neq \emptyset$ do

   If $\exists P_\ell \in$ Non_questioned_peers such that $\mathbf{U}_{u_{i*}}$ contains more than one message $U^i_{\ell\to *}$ then
   (a) P_i hard-punishes all P_ℓ that satisfy the condition;
   (b) P_i sets Duplicator = found;

   Else
   (a) P_i picks $P_\ell \in$ Non_questioned_peers such that P_ℓ sent the most recently timestamped message in $\mathbf{U}_{u_{i*}}$;
   (b) P_i questions P_ℓ by sending $\mathbf{U}_{u_{i*}}$ to P_ℓ;
   (c) If P_i receives from P_ℓ a message $U^i_{e\to\ell}$ that contains u_{i*} then
      i. P_i appends P_e to Non_questioned_peers;
      ii. P_i removes P_ℓ from Non_questioned_peers;

iii. P_i lets $\mathbf{U}_{u_{i*}} = \mathbf{U}_{u_{i*}} \cup \{U^i_{e\to\ell}\}$;

Else

i. P_i hard-punishes P_ℓ;

ii. P_i sets Duplicator = found.

Note that, unlike in PUNISH_BAD, in PUNISH_DUPLICATE a malicious P_i cannot make false claims: P_i must actually show a set $\mathbf{U}_{u_{i*}}$ of signed messages containing the same update in order to get help in upstream tracing.

Also, in the last step of PUNISH_DUPLICATE, a peer P_ℓ that sent only one message but does not help tracing the upstream duplicator is hard-punished. Thus, failing to collaborate to track a duplicator receives the same punishment as being a duplicator. Note that there is no danger that an honest update generator is punished in this way, even if the generator cannot by definition help tracing upstream (because she did not receive the update from anyone else). The reason is that PUNISH_DUPLICATE prioritizes the most recently timestamped updates, so that it will always find the real duplicator before reaching an honest generator.

Hence, if an honest generator comes to be questioned in PUNISH_DUPLICATE, she will hard-punish the questioning peer because the latter should not have questioned her.

Figure 4.3 shows the case where u_{01} was duplicated in Figure 4.1. If we assume the updates followed the paths shown in the bottom diagram, P_0 sends all the messages received with the duplicated update ($\mathbf{U}_{u_{01}}$) to P_3 (because $U^0_{3\to0}$ was the last message received, right before $U^0_{2\to0}$), and P_3 replies with the message she received from P_1 ($U^0_{1\to3}$). Then, P_0 questions P_2 by sending her all the collected messages. When P_2 sends back the message she received from P_1 ($U^0_{1\to2}$), P_0 knows that P_1 is the duplicator and hard-punishes her.

4.4 Co-utility analysis

In this section we will show that Protocol 1 will be adhered to by rational peers. Hence, the rational behavior of peers is to follow the protocol rather than attack it.

4.4.1 Model dissemination

Step 1 of Protocol 1 is rationally sustainable: without disseminating her model, a peer cannot expect to receive updates from other peers. A different issue is

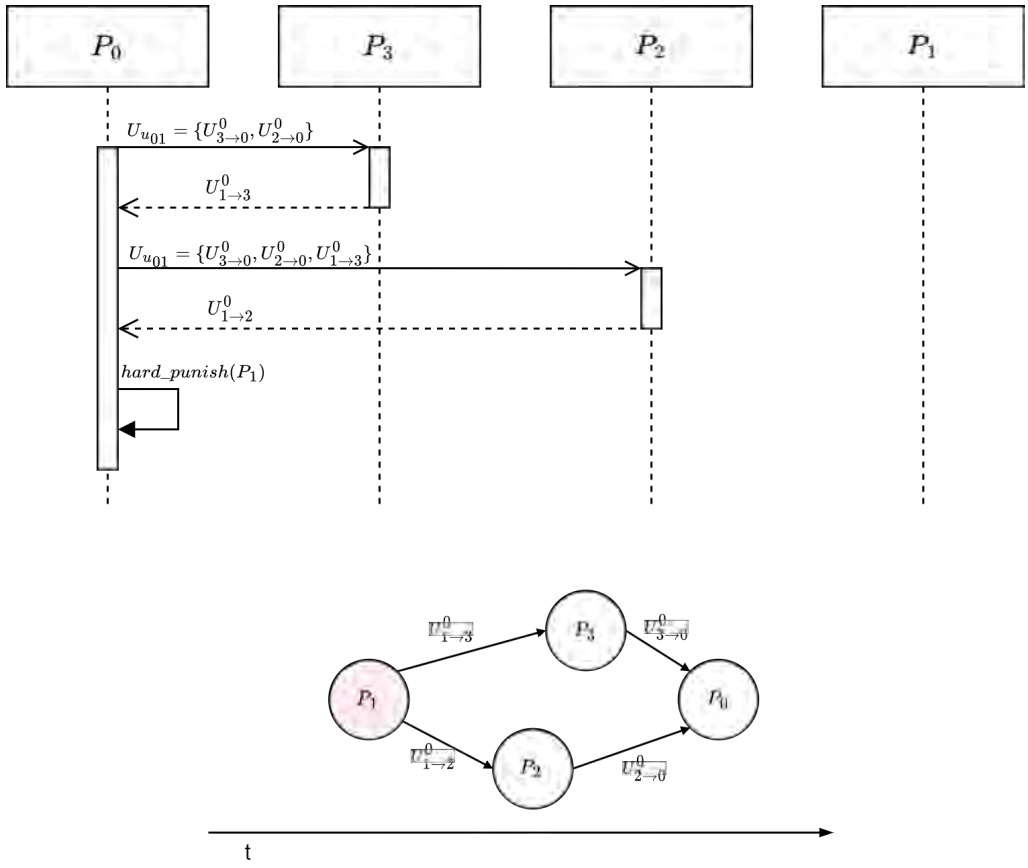


Figure 4.3: Sequence diagram showing the interactions among peers in an example execution of the PUNISH_DUPLICATE subprotocol. The diagram at the bottom shows the paths followed by the updates.

whether the DISSEMINATE subprotocol as described in Section 4.3.1 will be rationally followed by P_i and the other peers. We justify this.

On P_i's side, it is rational to avoid swamping the other peers with requests for updates: sending each request for updates has a computation cost (the signature) and a communication cost, and P_i knows that destination peers will reject duplicate requests for updates.

On the other hand, the destination peer, say P_j, is incentivized to avoid dropping the request for updates R_i, because if R_i brings a good update to P_i, P_i may be rewarded by P_j with a reputation increase.

However, P_j is interested in making sure she is not deliberately chosen by P_i and hence the target of a privacy attack by P_i. Hence, P_j is motivated to check the format of message R_i, to make sure she was chosen using the hash function and to check that R_i carries P_i's signature, in order to be sure about who is the model manager (this is necessary for the privacy tit-for-tat).

Last, P_j is rationally interested in launching a second dissemination hop to $P_{Dest(R_i^2)}$ in order to forestall an attack by a malicious P_i who would send m_i only to P_j (if the model owner sends her model to a single peer P_j, then the owner knows that any updates on her model have been computed by P_j, which compromises P_j's privacy). The more peers P_j sends m_i to, the more privacy P_j secures for herself.

4.4.2 Update computation

Step 4 of Protocol 1 is also rationally sustainable: by computing updates for the received models, a peer "mints" a currency that she will be able to use in the tit-for-tats. If P_i purposefully computes a bad update u_{k*} for P_k's model m_k, this is detected in the learning tit-for-tat and P_i is eventually hard-punished by the peer that firstly forwarded the bad update in the privacy tit-for-tat (Step 22). Hence, a peer who computes bad updates will receive fewer and fewer updates for her model from other peers as a consequence of a general worsening of her local reputations c_{*i}. Unless attacking another peer's model has a higher utility than training one's own model, the best option for a peer is to compute good updates in Step 4.

4.4.3 The privacy tit-for-tat

The privacy tit-for-tat is co-utile by design:

- At Step 8, peer P_i calls the UPDATE_EXCHANGE subprotocol to exchange non-repeated updates with another peer P_l. As described in Section 4.3.2, in this subprotocol P_i makes a tit move consisting of showing to P_l the hash images of all updates P_i has available for exchange. If no corresponding tat move by P_l follows (in which P_l shows the hash images of her available updates), P_i can exit the subprotocol. Otherwise, if P_i (resp. P_l) finds that P_l (resp. P_i) has a non-repeated update, P_i (resp. P_l) requests that update from P_l (resp. P_i).

  Both peers are rationally interested in getting non-repeated updates: indeed, PUNISH_DUPLICATE punishes a peer if she has sent more than once the same update, no matter whether this comes as a result of message duplication or as a result of a loop.

- In case P_l (resp. P_i) fails to provide the requested update, she will be hard-punished by P_i (resp. P_l) at Step 10. This can be viewed as a "currency" exchange. The exchange can involve updates that P_i and P_l have computed and updates they have received from other peers. The more updates a peer receives from other peers, the more updates she can send other than those she computes herself; the received updates are used as noise to hide the computed updates, which results in higher privacy. Hence, forwarding updates computed by other peers is co-utile.

4.4.4 The learning tit-for-tat

The learning tit-for-tat is also co-utile by design. Peer P_i makes a tit move by sending an update u_{k*} to P_k (Step 15) after having secured a tat move consisting of an update from P_k (Step 14). When requesting the tat move from P_k (Step 13), P_i expresses her preference for an update u_{i*} on her model.

In case P_k possesses an update u_{i*}, there is no reason why P_k should not send it to P_i; yet if P_k does not have any update u_{i*}, she can send another update to P_i as a valid tat move.

4.4.5 Model updating

For every good update received by P_i, it is in P_i's interest to use it to update her own model m_i (Step 17). After that, if the good update was sent by P_k, it is in P_i's interest to increase the local reputation c_{ik}, for the sake of keeping accurate records on the other peers' behavior and to increase the number of trusted

peers with whom P_i can interact. From the point of view of a newcomer peer P_k, the possibility of being rewarded with a reputation increase is an incentive to cooperate as an honest update generator or forwarder: as soon as the local reputation c_{ik} reaches P_i's interaction threshold T_i, P_k can expect P_i to compute or send to P_k updates for P_k's model.

4.4.6 Punishment

Finally, it is also in P_i's interest to punish the misbehavior by other peers, in order to discourage it. Punishment involves a decrease of the offender P_k's local reputation, which is a dissuasive punishment. If her local reputation c_{ik} is less than the interaction threshold T_i, P_k knows P_i will not send or compute updates for P_k' model.

Specifically,

- In Step 20, P_i hard-punishes those peers to whom P_i sent an update (in a requested tat-move) without receiving the corresponding tit.
- Although P_i can detect bad or duplicate updates and ignore them, it is in P_i's interest to discourage other peers from generating such updates. In the case of bad updates, P_i's incentive is to save detection work. In the case of duplicate updates, P_i's incentive is to deter bandwidth and processing wastage as a result of duplication.

  Hence:

  - P_i calls PUNISH_BAD to punish every peer P_k from whom P_i has received a bad update. If P_k did not generate that update, P_k's rational decision is to escape the most severe punishment (hard-punishing by P_i) by revealing the name P_l of the peer that sent the bad update to P_k. Even if P_k collaborates, P_k still shares some of the burden of the bad update, because P_k will be soft-punished by P_i, by slightly decreasing her local reputation c_{ik}. This motivates P_k to carry on the punishment upstream, in order to make sure the generator of the bad update gets hard-punished by the forwarder that first got the bad update.
    On the other hand, as pointed out in Section 4.3.4, a peer's rational behavior is, for privacy reasons, *not* to answer bad update claims from another peer to whom the former peer did *not* directly sent that update. See more detailed explanations at the end of Section 4.3.4.

– P_i calls PUNISH_DUPLICATE to punish duplicate updates and thus discourage them. In PUNISH_DUPLICATE, if a peer P_k did not generate a duplicate, it is in P_k's interest to escape punishment by revealing who sent that duplicate to her.

  On the other hand, although PUNISH_DUPLICATE only punishes the most downstream duplicator(s), this is enough. If an upstream peer decides to duplicate an update, there might not be any other duplicator further downstream, in which case she will be punished if at least two peers that have received her duplicates co-operate with the model manager to trace her. Hence duplicating always poses a risk to the duplicator.

4.5 Privacy and security

In this section we examine how the confidentiality of the peers' private data is protected and how bad and duplicate updates can be detected by model managers.

4.5.1 Privacy

As mentioned in Section 4.3 when describing the privacy tit-for-tat, breaking the link between an update and the peer that computed it goes a long way towards guaranteeing that the private data sets of peers stay confidential. We can state the following proposition.

Proposition 3. *In Protocol 1, the probability that a passive attacker (in particular, a peer) receiving updates from P_i can make inferences on P_i's private data is upper-bounded by $1/\kappa_i$.*

Proof. Any peer P_i sends the t_i updates she has computed on her private data set only in the privacy tit-for-tat. Thus, this is the only point in Protocol 1 where the confidentiality of P_i's private data set is at risk. This confidentiality can be violated if two circumstances concur: i) an update computed by P_i can be attributed by a peer P_l receiving it or by an attacker to P_i; ii) that update allows P_l or the attacker to make inferences on P_i's private data set.

Let us look at the attribution probability. In the privacy tit-for-tat, P_i sends a random mix of the t_i updates she has computed with $\kappa_i(t_i - 1)$ updates she has received. Hence, a peer P_l receiving an update from P_i or an attacker intercepting

it cannot decide whether the update was computed or merely received by P_i. From P_l or the attacker's point of view, the probability it was computed by P_i is at most $1/\kappa_i$, which is the proportion of updates computed by P_i among the updates sent by P_i. □

While Proposition 3 deals with passive attacks, a model manager might also attempt active attacks against privacy by trying to identify the generator of an update through abuse of the PUNISH_BAD or PUNISH_DUPLICATE subprotocols. More specifically:

- As explained in Section 4.3.4, the hop-wise operation of subprotocol PUNISH_BAD only allows the model manager to trace one hop upstream. Hence, the model manager cannot reach as far as the update generator.
- In PUNISH_DUPLICATE, the model manager certainly needs to show real duplicates to question peers. Yet, the model manager could decide to proceed further upstream even after finding the real duplicator, in order to reach the update generator. When the model manager hits a peer who does not help tracing further upstream, this peer may be just an uncollaborative peer or the update generator (the manager cannot know which is the case). Even if the manager guesses right that it is the generator, the latter can hard-punish the manager, who runs the risk of the attacked honest generator refusing to compute further updates for the manager: this will certainly occur if, as a result of the hard punishment, the manager's local reputation falls below the generator's interaction threshold. Hence, such an active attack offers an uncertain outcome for the manager and it is likely to entail hard punishing. Thus, this attack seems hardly rational for the model manager to carry out.

4.5.2 Security

In a decentralized learning process such as FDML, the main security goal is to ensure that no incorrect updates will be used to update the model being learned.

First of all, note that good updates in our approach are fully accurate, unlike noise-added updates obtained if using differential privacy.

On the other hand, as pointed out in Section 4.4, the co-utility of Protocol 1 ensures that rational peers are more interested in computing good updates than

bad ones. Thus, co-utility helps towards correctness by reducing the proportion of bad updates: having fewer bad updates means that they are more outlying with respect to good updates and hence easier to detect by model managers.

In the event that a bad update reaches a model manager P_i, the latter can detect it using the techniques mentioned in Section 4.3.4. As pointed out in that section, a distance-based approach [32, 31] is a convenient option provided that the model-managing peer waits to receive a batch of updates before deciding on the quality of each single update.

Regarding duplicate updates, it is straightforward for P_i to detect them if updates are continuous numerical values: it is very unlikely that two different peers generate exactly the same update u_{i*} for m_i. Hence, in case there is more than one update with the same value u_{i*}, this is a duplicate. If updates have low precision or are not continuous, equal updates for m_i may be genuinely computed by different peers. A fix is for each peer computing an update to make it unique by appending a random nonce; in this case, if P_i receives two equal updates with the same nonce, P_i knows they are duplicates.

4.6 Experimental results

In this section we report experimental work on several aspects. On the one hand, we evaluated how the rate of good updates received by the peers evolves over time depending on whether they compute good or bad updates and on whether they follow the protocol. On the other hand, we assessed the extent to which our approach reduces the communication overhead compared to systems based on decentralized global reputation management.

Note that the contribution of the chapter is largely independent of the actual models being learned, and of their particular parameters and performance. What we present is a protocol to incentivize the peers to generate and forward good updates (which improves accuracy), in such a way that updates cannot be linked to the peers generating them (privacy). Further, we show that our protocols are effective at discouraging misbehavior (security). Our mechanism can be combined with any decentralized learning models and it will help those models to get rid of bad updates and thus improve their performance (note that our protocol does not alter good updates).

Therefore, in this experimental section we use one specific machine learning model on a particular data set just for the sake of illustration. As justified below, the

only parameters of the learning process that are relevant to our protocol are the false positive rates and the false negative rates when model managers classify the received updates into good or bad.

4.6.1 Experimental setting

In our experiments we used the MNIST data set of handwritten digits, which we evenly split among $n = 100$ peers as their local data.

Following the experimental setting from [24], the baseline global model was a neural network where all convolutional layers and the first dense layer used the ReLU activation function. The second dense layer, which was the output layer, used softmax. The loss function was categorical crossentropy. The model substitution rate η was set to 0.25, meaning that in each epoch the model kept information about past training epochs, which made the training process more stable because there were no abrupt changes in the model performance from one epoch to the next. To compensate for the low substitution rate, the training process needed to be longer. Training was carried out for 100 epochs, in which all peers ran Protocol 1. Local updates were computed after 2 epochs of local training, with a batch size of 32. We used distance-based outlier detection to detect bad updates. The accuracy of the main task (classification of MNIST digits) was 97.87% due to the presence of attacks (bad updates), down from 99.6% in the absence of attacks. The outlier detection mechanism was highly effective, with a false negative rate (good updates detected as bad) as low as FNR=3.8%, and a false positive rate (bad updates detected as good) FPR=2.1% on average over all epochs. In other words, only a small proportion of updates generated by honest peers were classified as bad updates and were wrongly punished, and only a small proportion of bad updates were rewarded as good.

While the development and discussion of attack detection mechanisms on FL such as those recalled in Section 4.3.4 are out of the scope of this chapter, high FPR and FNR rates might degrade the performance of our protocol, as they lead to more frequent unjust punishment of good peers and incorrect reward of evil peers.

The model dissemination subprotocol (Section 4.3.1) requires the requester P_i to find a set of different destination peers by generating sequence numbers and computing $j = Dest(R_i)$ for each of the requests for updates of her model m_i. In case P_i finds a collision, she can choose new sequence numbers and recompute $j = Dest(R_i)$. Note that the more requests P_i wants to send, the more difficult

it is for P_i to find peers to directly send her request and, eventually, after the forwards of the other peers, randomly hit different final destination peers for all requests about m_i.

Therefore, the rational behavior of a requester P_i is to limit her number of requests so that the probability of sending all requests to different destination peers stays above an acceptable threshold, say 0.5. This problem is similar to the birthday problem[2], that consists in computing the probability that in a group of people all individuals have different birthday dates. In our setting, the number of possible birthdays was the number of peers ($n = 100$) and we found that a peer could send up to 12 requests while keeping the probability of sending all requests to different peers above 0.5. Hence, we set the number of requests sent by peers in every epoch to 12.

We should recall that a peer is soft-rewarded when she is the first peer to receive a request directly from a manager in the dissemination phase, and when she is the last forwardee of a good update. Even peers who are not trusted by others (with a reputation below the interaction threshold) participate in the dissemination phase, so they can be rewarded. Similarly, they can be the last forwardee of a good update if they start the interaction in the "learning tit-for-tat" protocol. These good behaviors should be rewarded (because they help good updates reach their respective managers), but the reputation increase should be much smaller than the hard punishment (δ) they receive for bad behavior (computing bad updates or duplicating them). Also, the soft punishment is used in the PUNISH_BAD protocol against peers that unintentionally forward bad or duplicated updates to incentivize them to launch punishment against the malicious peers. However, this punishment should also be much lower than the hard punishment received when a peer behaves badly.

We tested different values of δ with soft punishment and rewarding fractions $\delta/10$ and $\delta/4$, respectively, and 10% of malicious peers performing the attacks described in the following sections. We let the network of $n = 100$ peers evolve during 100 epochs and we computed the ratio between the number of useful updates received by a malicious peer and the number received by an honest peer. The main goal of our protocol is for honest peers to receive more useful updates than malicious peers. In this way, honest behavior will be the rational option. The results are shown in Table 4.2.

[2] `https://en.wikipedia.org/wiki/Birthday_problem`

Table 4.2: Ratio of useful updates received by a malicious peer vs an honest peer for different values of δ

delta	Evil	Duplicator	Whitewasher	Selfish
0,10	0,1849	0,2279	**0,1114**	0,2872
0,20	0,1774	0,2491	0,1324	0,2765
0,30	0,1722	0,2287	0,1520	0,2766
0,40	0,1810	0,2213	0,1773	0,2762
0,50	0,1849	0,2277	0,1816	0,2734
0,60	0,1799	0,2294	0,2049	0,2731
0,70	0,1748	0,2284	0,1996	0,2744
0,80	0,1684	0,2308	0,2047	0,2748
0,90	0,1896	0,2410	0,2095	0,2750
1,00	0,1669	0,2495	0,2047	0,2726

We can see different values of δ had no effect on the ratio of good updates received by malicious peers but they did in the case of whitewashers. The latter peers always compute bad updates and whitewash as newcomers every 10 epochs (see details in Section 4.6.3). Indeed, we can see that the lower the value of δ, the lower was the ratio of good updates whitewashers received. With $\delta = 0.1$ a whitewasher got only 11.14% of the good updates received by an honest peer, while with $\delta = 1.0$ the ratio increased to 20.47%. The reason is that a larger δ causes the local reputations c_{ik} to change swiftly. Newcomers enter the system with the lowest possible local reputation (0) and they should earn reputation by forwarding requests or sending useful updates to another peers. When the value of δ is larger, whitewashers surpass the interaction threshold of other peers earlier, and they can start flooding the system with bad updates. Given these results, we set the value of δ to 0.1.

We next show that our previous choice of fractions $\delta/10$ and $\delta/4$ for soft punishment and rewarding, respectively, is a good one. To that end, we tested different fraction choices with $\delta = 0.1$ and 10% of malicious peers performing the attacks described in the next sections, and we computed again the ratio between the number of useful updates received by a malicious peer and the number received by an honest peer. The results are shown in Table 4.3. We can see that the best combination is indeed to set the soft reward to 25% of δ, and the soft punishment to 10% of δ. With this combination, all malicious peers obtain the lowest ratio of good updates.

Table 4.3: Ratio of useful updates received by malicious peers vs honest peers for different values of soft punishment and reward fractions of δ

Soft reward	Soft punishment	Evil	Selfish	Duplicator	Whitewasher	Avg
1.00	1,00	0,657	0,726	0,233	0,268	0,471
	0.50	0,736	0,765	0,206	0,267	0,493
	0.25	0,760	0,756	0,198	0,265	0,495
	0.10	0,813	0,779	0,191	0,267	0,513
0.5	1,00	0,466	0,384	0,467	0,288	0,401
	0.50	0,380	0,345	0,169	0,277	0,293
	0.25	0,392	0,345	0,133	0,275	0,286
	0.10	0,414	0,374	0,120	0,271	0,295
0.25	1,00	0,997	0,286	0,971	0,339	0,648
	0.50	0,371	0,262	0,536	0,301	0,368
	0.25	0,224	0,253	0,172	0,290	0,235
	0.10	**0,180**	**0,239**	**0,116**	**0,287**	**0,205**
0.10	1,00	1,001	0,606	1,002	0,433	0,760
	0.50	0,997	0,316	0,989	0,400	0,675
	0.25	0,885	0,268	0,890	0,351	0,599
	0.10	0,231	0,224	0,226	0,320	0,250

Hence, in what follows we set the parameter δ used for hard punishment to 0.1 and the soft punishment and rewarding fractions to $\delta/10$ and $\delta/4$, respectively.

4.6.2 Baseline scenario

Ideally, in our protocol a peer should receive good updates if and only if that peer computes good updates for other peers. Thus, we first tested how the system evolved when all the peers in the network were honest: they adhered to the protocol without deviating and they always computed good updates for the other peers.

Figure 4.4 shows the ratio between the number of useful updates (where an update is useful if it is good and non-duplicate) received and the number of requests sent by all the peers. The green line represents the average ratio over all peers for the epoch in the abscissae, while the shaded region represents the standard deviation of the ratio. In the first epochs, the behavior of the peers had not yet been fully captured and, for this reason, the misclassification of some updates caused a decrease in the ratio of useful updates received. When the system stabilized, the peers received around 72% of their requested updates, because some requests or updates got lost in different phases of the protocol execution.

These losses can be explained as follows. In spite of limiting to 12 per epoch the number of requests for updates sent by each model managing peer, around

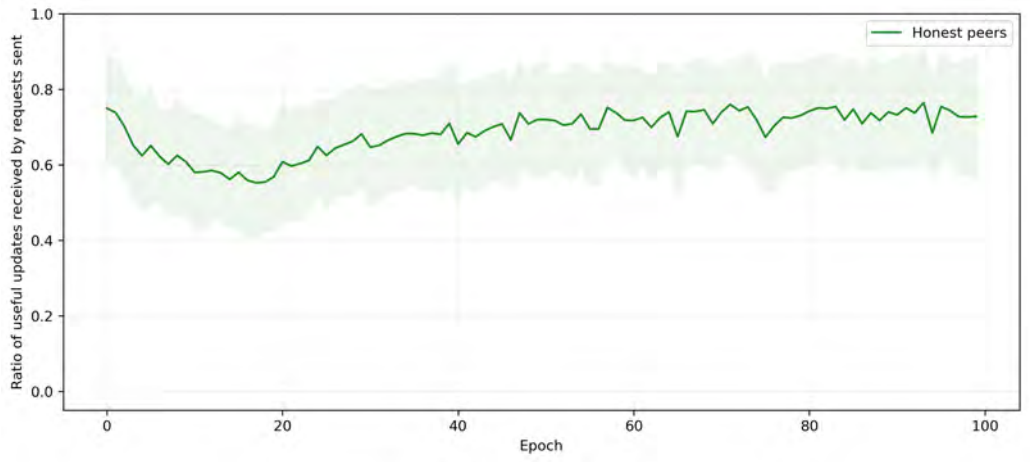


Figure 4.4: Baseline scenario. Ratio of useful (good and non-duplicate) updates received vs requests sent at each epoch.

6% of requests were dropped due to collisions at the dissemination stage. Then, some updates (around a 18%) were also dropped because some model managers could not offer other updates in exchange for them during the learning tit-for-tat. Last, around a 3.8% of updates were wrongly classified as bad, which triggered the PUNISH_BAD subprotocol. As a result of this misclassification, some honest peers were punished and therefore their local reputations temporarily fell below the interaction threshold of other peers, which prevented these honest peers from receiving some updates.

Figure 4.5 depicts the flow from the expected updates to be received by the model managers ("Models sent") to the actual good updates received ("Good updates received"). Some requests were dropped due to collisions in the dissemination phase ("Models ignored"). Other requests were not handled ("Updates not computed") due to a temporally model managers' low reputation in the initial epochs when the peer behavior was not yet fully captured. Additionally, as we explained above, some updates could not be exchanged with their corresponding model managers ("Updates not sent to MM") and some updates were wrongly classified as bad ("Bad updates received").

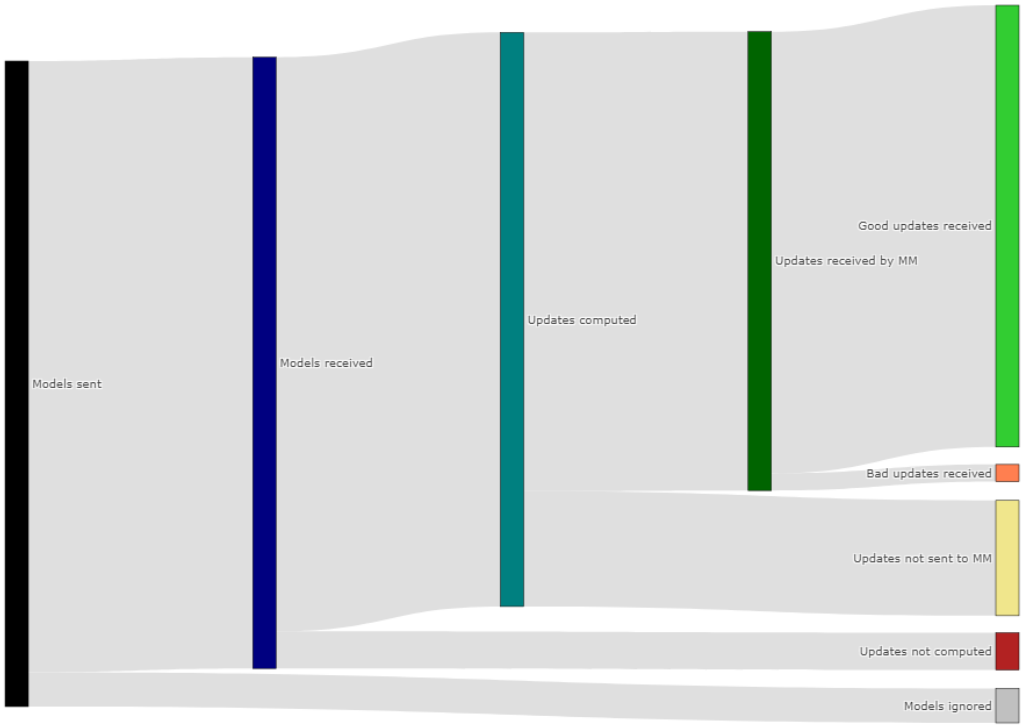


Figure 4.5: Flow of updates

4.6.3 Protection against attacks

We next considered attacks by malicious peers. We wanted to check to what extent honest peers did better than attackers in terms of received useful updates: if the former did better, then honest behavior is the rational option, as intended by the design of our protocol.

Specifically, we considered the following attacks by malicious peers: i) evil peers who follow the protocol but always compute bad updates; ii) selfish peers who try to free-ride on the other peers' computational effort; iii) duplicator peers who duplicate updates computed by other peers in order to save computational effort; and iv) whitewashers, who are peers with poor local reputations that choose to leave the network and re-enter it with a new identity as newcomers.

Evil peers

This kind of peers follow the protocol but always compute bad updates. Our protocol tries to thwart evil peers from finding other peers who compute good updates for them. Hence, if our protocol works well, evil peers should receive a lower ratio of good updates than honest peers.

To test this attack, we set up a scenario with the same parameters as the baseline scenario but with 10% of evil peers. The remaining 90% were honest peers.

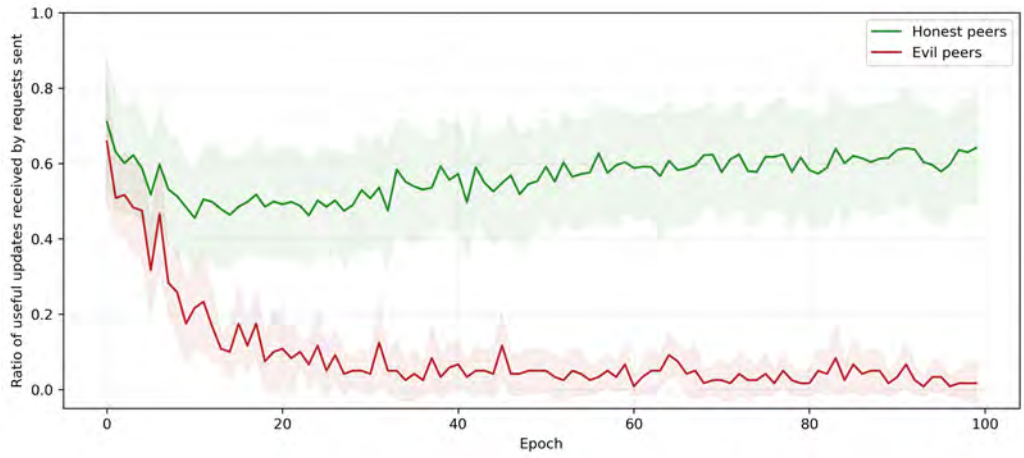


Figure 4.6: Scenario with evil peers. Ratio of useful updates received vs requests sent at each epoch.

Figure 4.6 displays the ratios between the number of useful updates received and the number of requests sent by honest and evil peers. As in the baseline scenario, during the first epochs when the behavior of the peers was being modeled, the ratio of useful updates received by the honest peers decreased. However, after this initial stage, this ratio increased constantly. After 100 epochs, honest peers were receiving around 64% of their requested updates, whereas evil peers received fewer than 3%. In the first epochs, evil peers were detected and hard-punished as soon as their updates reached the model managers. In this way, their local reputations eventually fell below the interaction threshold of the other peers and, at each subsequent epoch, evil peers found fewer and fewer other peers who were willing to interact with them.

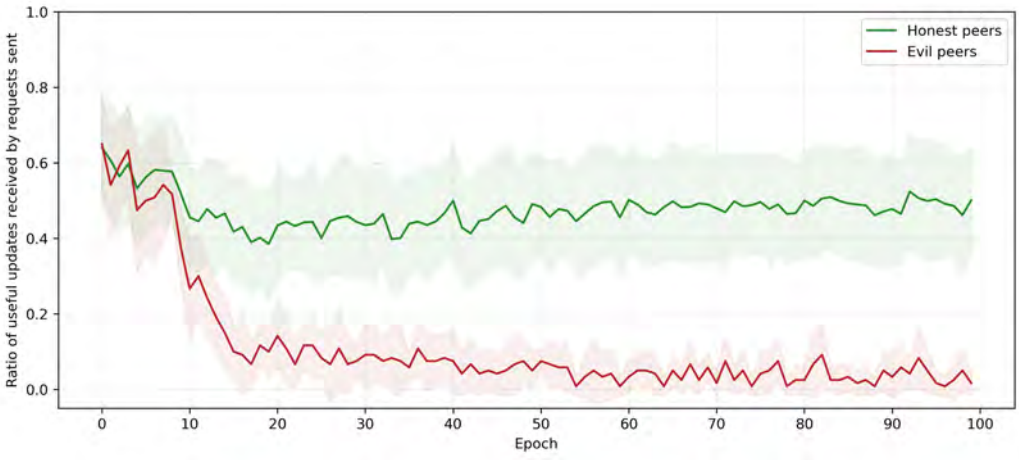


Figure 4.7: Scenario with evil peers. Ratio of useful updates received vs requests sent at each epoch when the effectiveness of the model manager's detection mechanism for bad updates is degraded to FNR=10% and FPR=8%.

In Figure 4.7 we show the behavior of our protocol when simulating a degraded outlier detection mechanism with a higher FNR of 10% (*i.e.*, 10% of good updates are classified as bad) and a higher FPR of 8% (*i.e.*, 8% of bad updates are classified as good). While these figures mean that the outlier detection procedure now misclassifies updates nearly three times more often than the procedure used in previous experiments (with FNR=3.8% and FPR=2.1%), our protocol is not significantly affected, and it is still able to differentiate between good and evil peers. Despite this poorer detection, evil peers receive only around 5% of useful updates out of their total sent requests. These results demonstrate that, even if the model manager uses an ineffective detection mechanism, malicious peers are still substantially penalized, and thus they have a rational motivation to stop their behavior.

Figure 4.8 shows the ratio between the number of useful updates received and the number of requests made as a function of the proportion of evil peers, after 100 iterations. Honest peers obtained a higher ratio of useful updates than evil peers when the proportion of evil peers remained below 35%. However, this ratio decreased as the proportion of evil peers increased, due to both the lower number of trusted peers and the higher total number of bad updates computed. Fewer

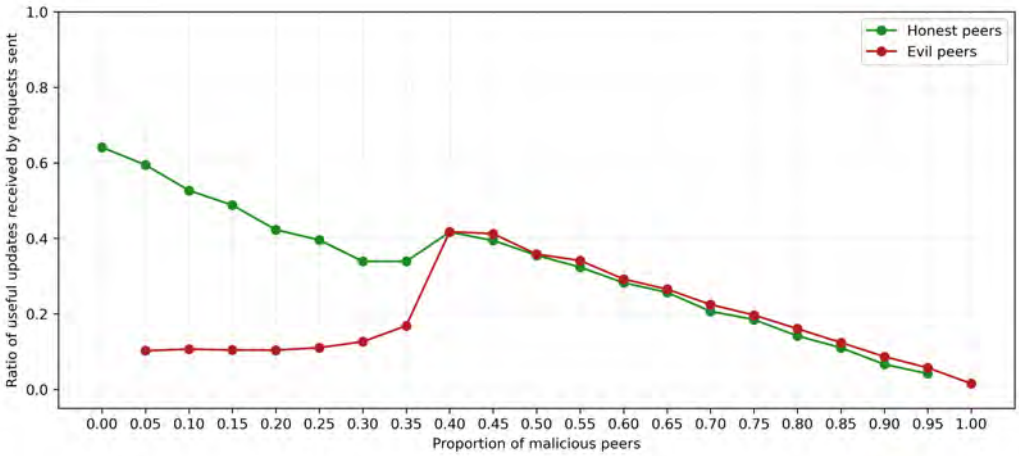


Figure 4.8: Scenario with evil peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of evil peers.

trusted peers imply fewer peers to exchange updates with, and fewer good updates that can be sent to their corresponding model managers. Evil peers kept a low ratio of useful updates when their proportion was below 30%. Nevertheless, at a proportion of around 40%, the computation of the interaction thresholds T_i was not able to capture the peers' behavior correctly, and all peers began to interact with each other and received the same ratio of useful updates. As the proportion of evil peers increased, fewer good updates were generated and the ratio of useful updates dropped to almost zero. Anyway, proportions of evil peers higher than 30% can be regarded as unrealistic.

Selfish peers

This type of peers disseminate their model and request new updates, but they ignore all the requests from other peers except one. In this way, they try to obtain new good updates while saving most of their own computing resources.

Note that if selfish peers did not compute any update, they would not be able to participate in the tit-for-tats.

We tested this attack in a scenario with the same parameters as the baseline scenario but with 10% of selfish peers. The remaining 90% were honest peers.

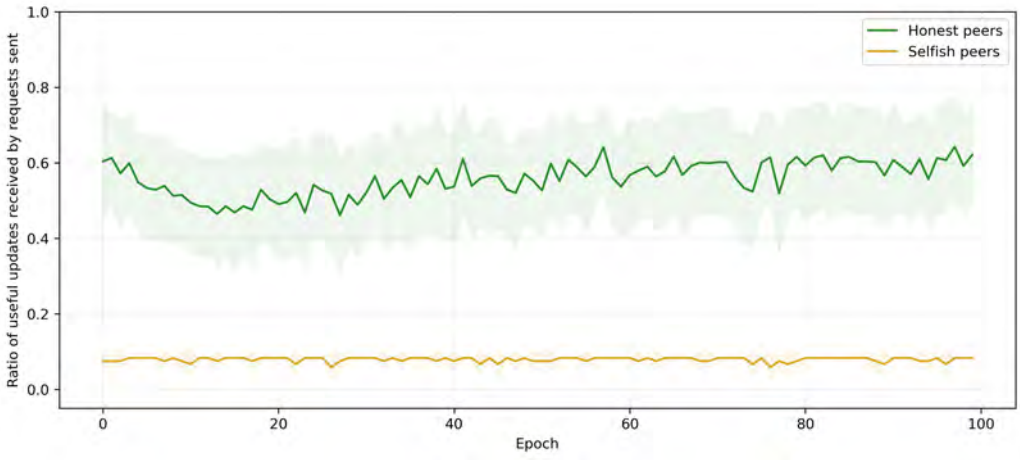


Figure 4.9: Scenario with selfish peers. Ratio of useful updates received vs requests sent at each epoch.

Figure 4.9 shows the ratios between the number of useful updates received and the number of requests sent by honest and selfish peers. Honest peers ended up receiving around 60% of their requested updates, whereas selfish peers received at most 8%. Indeed, since selfish peers computed only one update, they only had one update to trade with the other peers in the learning tit-for-tat; hence, they received at most one new good update per epoch.

Additionally, Figure 4.10 plots the number of updates computed by selfish and honest peers against the number of updates they receive from other peers. The bottom-left points correspond to the 10 selfish peers who compute and receive 100 updates each, one per epoch, which explains why they appear as a single (overprinted) point. On the other hand, the top-right points correspond to the 90 honest peers. The correlation between the number of computed and received updates is really high (0.983), as anticipated in Section 4.3 above.

Figure 4.11 shows the ratio between the number of useful updates received and the number of requests made as a function of the proportion of selfish peers, after 100 iterations. Regardless of their proportion, since selfish peers computed only one update per epoch, they received at most one new good update (8% of the 12 update requests they sent per epoch). Meanwhile, honest peers received more

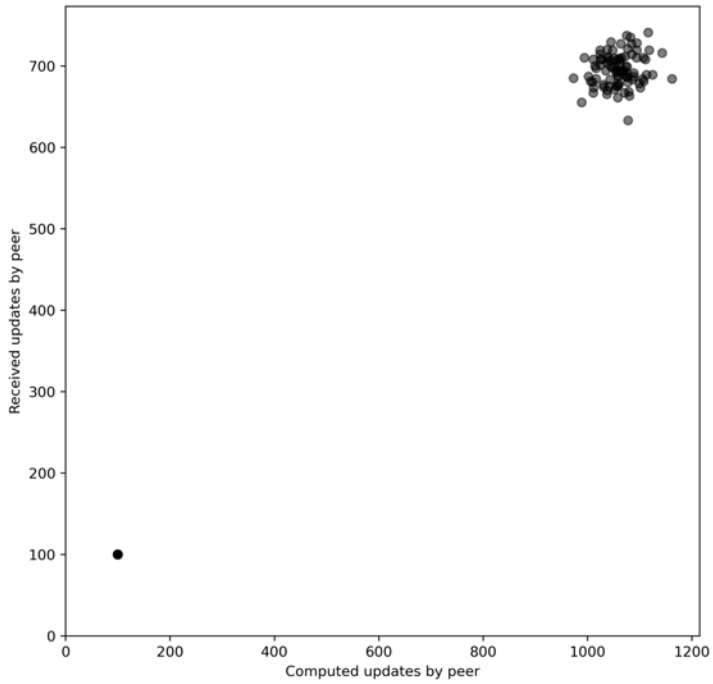


Figure 4.10: Scenario with selfish peers. Computed updates against received updates for selfish and honest peers. Correlation: 0.983

useful updates than selfish peers for almost any proportion of selfish peers, but this ratio decreased as the proportion of selfish peers increased, due to the lower overall number of updates computed.

Duplicator peers

An alternative option for a peer to save her own computing resources is to become a duplicator rather than a selfish peer.

Instead of computing updates themselves, duplicator peers just duplicate some of the updates they receive from other peers. By doing so, the duplicator effortlessly obtains several updates to trade with the other peers in the learning tit-for-tat.

To test this scenario, we used again the same parameters as in the baseline scenario but with 10% of duplicators. These peers duplicated every update they sent in

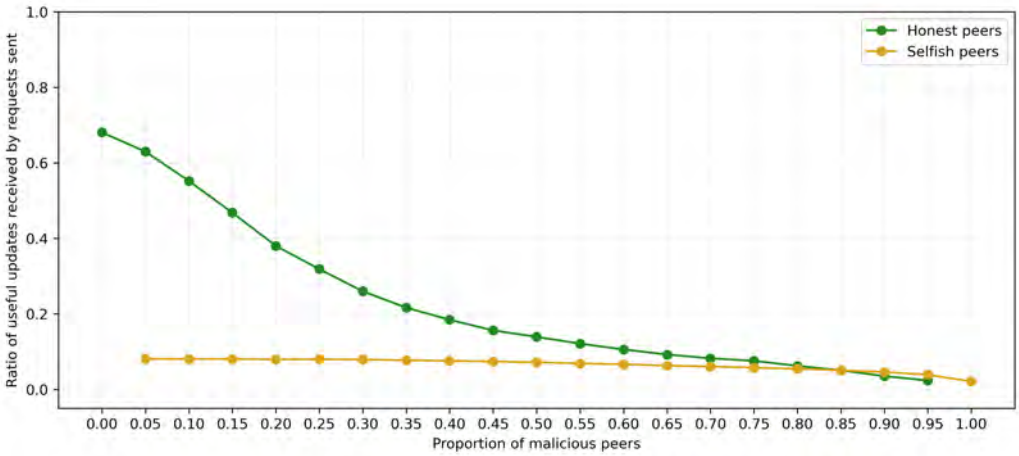


Figure 4.11: Scenario with selfish peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of selfish peers.

the privacy tit-for-tat, thereby amassing updates to be traded later.

Figure 4.12 shows the ratios between the number of useful updates received and the number of requests sent by honest and duplicator peers. While honest peers received almost 57% of their requested updates, duplicators only received around 5%. As soon as the model managing peers received a duplicated update, they launched the PUNISH_DUPLICATE subprotocol, identified the nearest duplicator and hard-punished her by decreasing her local reputation, thereby avoiding new interactions with that peer as soon as her reputation became lower than the interaction threshold. As duplicators kept duplicating updates, their local reputations never recovered.

Figure 4.13 shows the ratio between the number of useful updates received and the number of requests made as a function of the proportion of duplicator peers, after 100 iterations. Honest peers obtained a higher ratio of useful updates than duplicators when the proportion of duplicators remained below 45%. However, this ratio decreased as the proportion of duplicator peers increased, due to the lower number of trusted peers on the one hand, and to the higher total number of duplicated updates on the other hand. At 50% of duplicators, all peers received the same ratio of useful updates, and above this value, duplicators obtained a

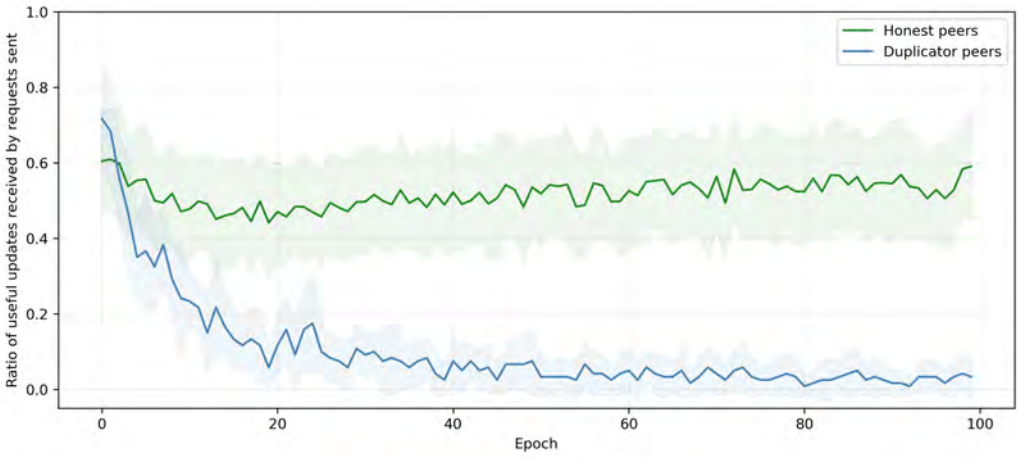


Figure 4.12: Scenario with duplicators. Ratio of useful updates received vs requests sent at each epoch.

higher ratio than honest peers. At this point, the interaction threshold failed to capture the peer's behavior and all peers began to interact with each other. Since duplicators produced more updates (by duplicating some of them) than honest peers, they were able to exchange them and received more updates for their own model. Nevertheless, duplicators never received more than 30% of the expected updates, a value far from the 64% received by the honest peers in the absence of duplicators.

Whitewashers

Evil peers and duplicators are hard-punished by several peers at each epoch. After a few epochs, as their local reputations have decreased to the point of being smaller than the interaction thresholds of most of the other peers, they receive a low percentage of the updates they request, as shown in the previous experiments. In these circumstances, evil peers and duplicators may be tempted to whitewash themselves by leaving the network and returning as newcomers. However, by design a newcomer enters the system with the lowest possible local reputation (zero), which is below the interaction threshold of almost every other peer.

We tested this scenario by using the same parameters as in the baseline scenario but with 10% of whitewashers. These peers always computed bad updates and

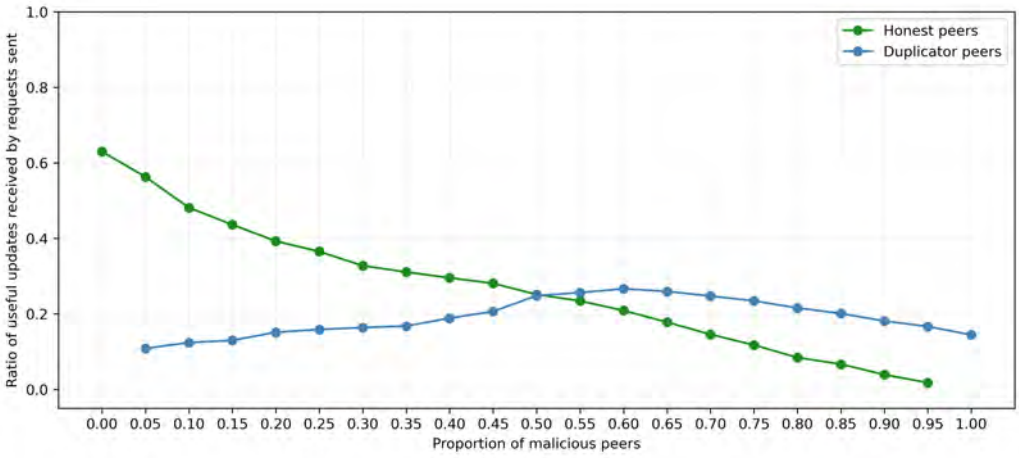


Figure 4.13: Scenario with duplicator peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of duplicator peers.

whitewashed as newcomers every 10 epochs.

Figure 4.14 depicts the ratios between the number of good updates received and the number of requests sent by honest peers and whitewashers. Honest peers eventually received around 60% of their requested updates, whereas whitewashers ended up receiving no useful updates.

Although the whitewashers had the option of improving their local reputation by forwarding requests or updates to other peers, they kept computing bad updates and thus they were hard-punished by the model managing peers who received these bad updates. Therefore, in general they never managed to be trusted by the other peers; they only could occasionally get good updates from peers they had not yet interacted with (which explains the peaks).

Figure 4.15 shows the ratio between the number of useful updates received and the number of requests made as a function of the proportion of whitewashers, after 100 iterations. The results are very similar to those we obtained with evil peers (see Figure 4.8). Again, when the proportion of whitewashers was below 35%, honest peers obtained a higher ratio of good updates than malicious peers. However, from around 40% upwards, the computation of the interaction thresholds T_i was not able to capture the peers' behavior correctly (most of the interaction thresholds

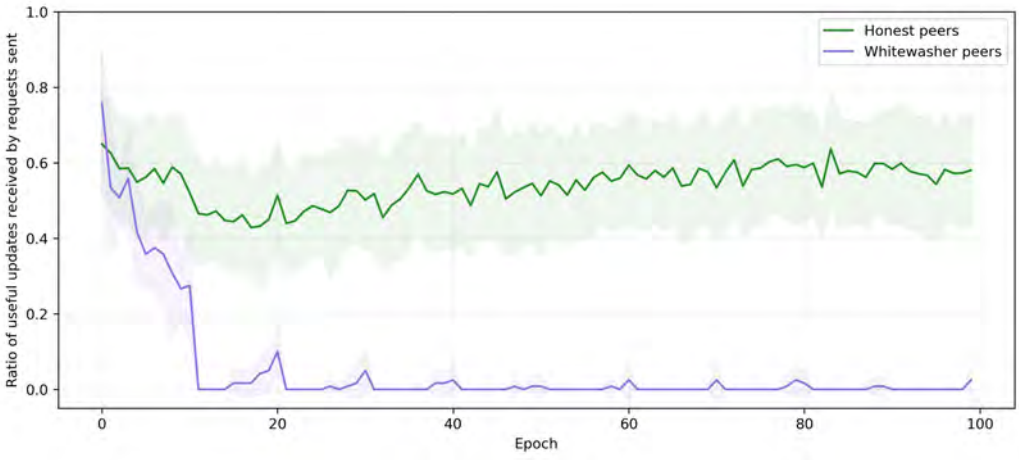


Figure 4.14: Scenario with whitewashers. Ratio of useful updates received vs requests sent at each epoch.

dropped close to zero, the initial reputation of newcomers), and all peers started to interact with each other and received the same ratio of good updates.

We finally considered the case of an honest newcomer, who computes good updates, to illustrate that she does much better than whitewashers. We had this newcomer join the network right after epoch 20. Figure 4.16 shows that the honest newcomer increased her ratio of received useful updates as a consequence of her participation in the protocol, by computing and forwarding new updates and requests for the rest of the peers. As soon as the newcomer forwarded a request from another peer in the dissemination protocol or sent a useful update to another peer in the learning tit-for-tat, the beneficiary peers rewarded her with an increase of her local reputation. Although in the first epochs after entering the system these small increases were not enough to surpass the interaction threshold of other peers, eventually they were and the newcomer became trusted to more and more peers, who sent updates to her.

4.6.4 Overhead incurred by the protocol

Decentralized global reputation can be used as an alternative to tit-for-tat and local reputations to build artificial incentives that yield co-utile protocols. However, as noted in Section 3.2, global reputation management requires a significant

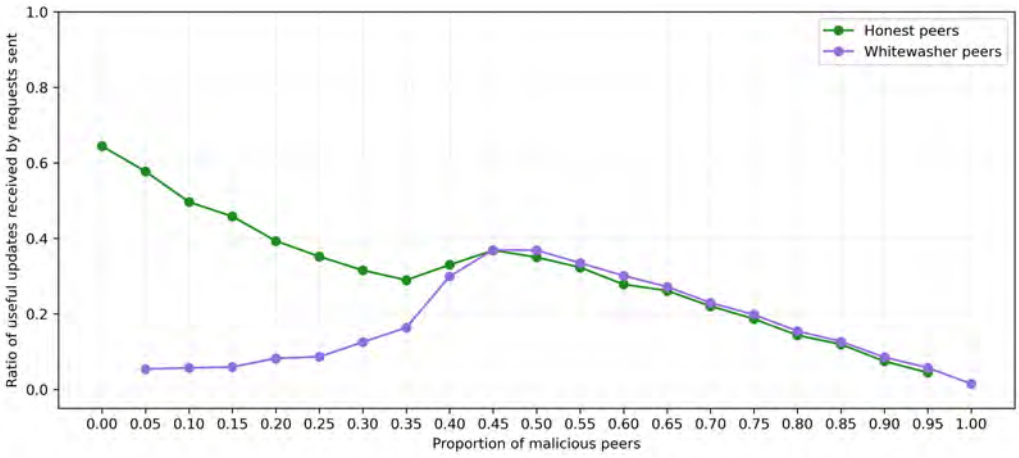


Figure 4.15: Scenario with evil peers. Ratio of useful updates received vs requests during 100 epochs, as a function of the proportion of whitewasher peers.

number of messages to be exchanged between the peers of the network to compute the global peer reputations.

In order to compare the overhead of decentralized global reputation management with that of our tit-for-tat protocol, we estimated the number of messages needed by an adaptation of the global reputation protocol of [55] to FDML. In this case, every peer expecting a reward after computing or forwarding an update needs to send at least one message to each of her accountability managers. Then, after all peers have received the requested updates, global reputations are computed. The reputation values maintained by the accountability managers are normalized into the range $[0, 1]$ by dividing them by the largest reputation in the network. To that end, it is necessary that accountability managers broadcast all reputation values greater than 1. Finally, a decentralized global reputation computation must be run, which entails a new exchange of messages among the accountability managers.

In contrast, the tit-for-tat based on local reputations does not require additional communication among peers because the local opinion of a peer on another peer is never shared. However, as some good updates are lost in the learning tit-for-tat when a model manager does not have more updates to exchange, we counted as

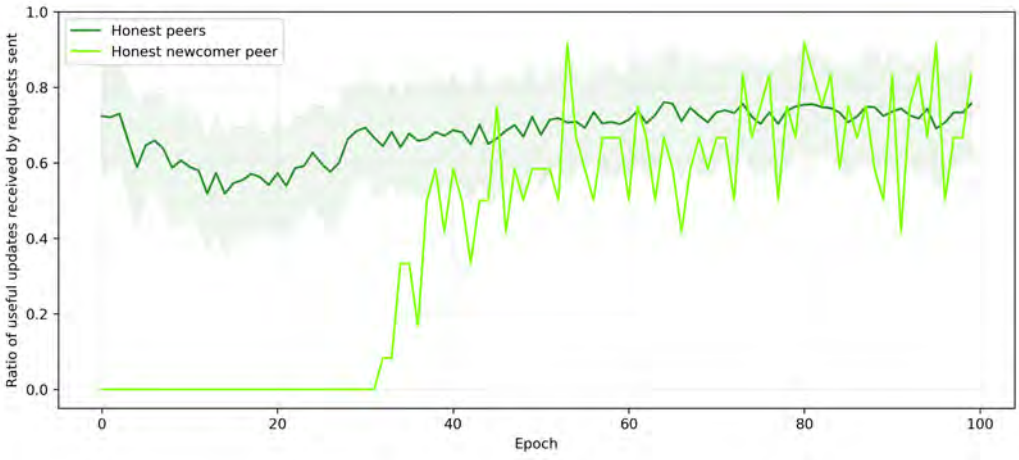


Figure 4.16: Scenario with an honest newcomer. Ratio of useful updates received vs requests sent at each epoch.

overhead the messages generated to forward these lost updates.

Figure 4.17 shows the overhead incurred by our tit-for-tat protocol and three versions of the protocol based on decentralized global reputation in a network where all peers are honest. The first global reputation-based protocol rewards with a reputation increase only the peer that generates a good update; the second rewards the generator of a good update and the first forwarder of that good update; finally, the third rewards the generator of a good update and all the forwarders along the path from the generator to the model manager. Every peer disseminated her model among 12 other peers and the number of epochs was 100.

The size of the network did not affect the overhead ratio incurred by any of the four protocols in the comparison. The tit-for-tat protocol added only a modest overhead in comparison with the global reputation-based protocols. Moreover, the greater the number of peers to be rewarded for each good update, the greater the overhead of global reputation-based protocols.

4.7 Conclusions and future work

We have presented a co-utile protocol for FDML. The fact that all participating peers are at the same time model managers and workers facilitates using tit-for-tat

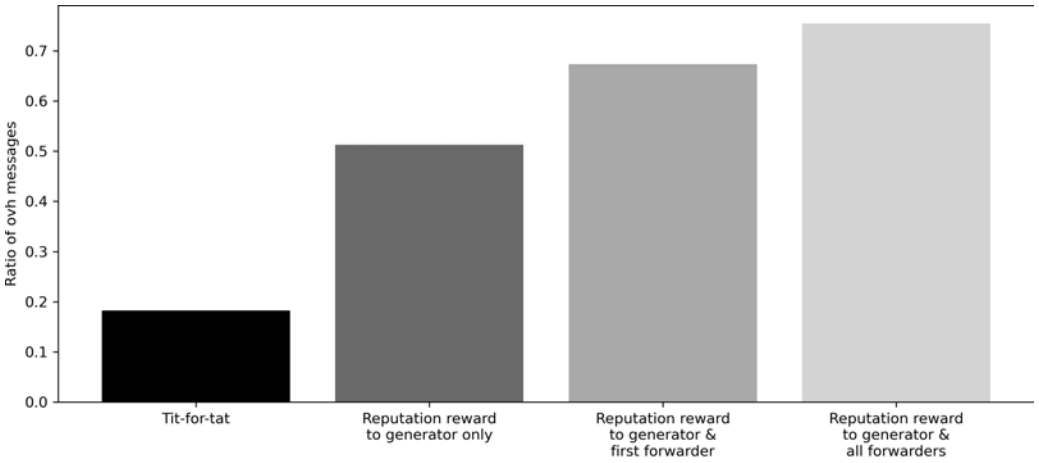


Figure 4.17: Overhead of the tit-for-tat protocol and three protocols based on decentralized global reputation

and local reputations to achieve co-utility. In this way, co-utility occurs naturally and complex incentives such as decentralized global reputations are not needed.

Our proposed protocol consists of a double tit-for-tat. The first tit-for-tat is used to hide the identity of the peer that computed each update, in order to protect the privacy of peers. In the second tit-for-tat, peers receive updates relevant to their own model, but they receive them from other peers that did not compute those updates. Deviations from the protocol are punished, which increases ostracism for the offending peers. On the one hand, a peer that fails to reciprocate in a tit-for-tat is punished by the non-reciprocated peer. On the other hand, after the second tit-for-tat, bad or duplicate updates can be detected by the model-managing peers, who launch punishment subprotocols aimed as tracing the offending peers. This ensures correctness of the learning process and incentivizes good and non-duplicate updates.

Future research will be devoted to fine-tuning the protocol in order to further reduce its overhead and the proportion of lost good updates. Also, we plan to explore adaptations of the proposed protocol to decentralized computations different from machine learning, such as decentralized anonymization or even multiparty computation among rational peers.

Chapter 5

Circuit-Free General-Purpose Multi-Party Computation via Co-Utile Unlinkable Outsourcing

5.1 Introduction

Multiparty Computation (MPC) is a cryptographic protocol that allows multiple parties to jointly compute a function over their private inputs without revealing any individual data. The fundamental concept behind MPC is to preserve privacy while achieving a collective computational result. It allows parties to jointly perform computations on sensitive data, such as financial records, health information or confidential research data, without the need for a trusted intermediary.

MPC has applications in several domains. One prominent use case is secure data analytics, where multiple organizations can pool their data without sharing them directly, which enables them to gain valuable insights while preserving privacy.

Despite its benefits, MPC faces certain threats and challenges. A key concern is the potential for malicious parties to subvert the protocol and manipulate the

results of the calculations. Adversaries could attempt to gain unauthorized access to sensitive data or tamper with the computation to obtain unintended information. Protecting against such threats requires rigorous security measures, including cryptographic techniques and robust protocols, as well as careful threat modeling and analysis.

In addition, due to its cryptographic nature, MPC introduces computational and communication overheads that can affect its practical scalability. Efficient implementation of MPC protocols is an ongoing area of research aimed at reducing computational and communication costs while maintaining security guarantees.

5.2 Contributions

In this chapter, we propose an MPC protocol based on the co-utility property (see Section 2.2) that has the following distinguishing properties:

- It is general-purpose without making use of circuits; it can work for any computation expressed as ordinary high-level programming code, no matter the depth of its loops, its use of recursion or its complexity.
- It does not rely on the cryptographic primitives usual in MPC; it uses cryptography only to guarantee confidential and authenticated communications.
- It assumes a peer-to-peer (P2P) community is available, where each peer accumulates a reputation in a decentralized and *co-utile* manner.
- It relies on the notion of *co-utile* anonymous channel to guarantee that inputs and outputs, although visible to some peers, cannot be linked to their corresponding parties.
- It delivers correct exact outputs as long as parties are *rational*.

We illustrate our approach in two example applications, one of them being electronic voting. Our empirical work shows that reputation captures well the behavior of peers and ensures that parties with high reputation obtain correct results.

Section 5.3 introduces a co-utile protocol suite for MPC. Section 5.4 discusses how the proposed protocol suite achieves co-utility (and thus is rationally sustainable). Section 5.5 shows how our protocols satisfy privacy for the inputs and the outputs, and how they make correct computation the best option for rational peers.

Section 5.6 sketches illustrating applications. Section 5.7 presents experimental results. Finally, Section 5.8 deals with conclusions and future research.

5.3 A co-utile framework for multi-party computation

5.3.1 Players and security model

The players in our framework are peers in a P2P network as follows:

- *Clients* are the parties that agree on a joint computation to be conducted, and then provide private inputs to it and obtain private outputs from it. Since the purpose of MPC is for clients to obtain correct outputs, they can be assumed to provide truthful inputs. No MPC protocol can detect if one or more clients provide untruthful inputs in order to alter the outputs derived by the other clients.
- *Workers* perform computations for clients.
- *Forwardees* receive messages from clients and forward them to other forwardees or submit them to workers.
- *Accountability managers* are peers that manage the reputations of other peers in the network. Each peer P is assigned M accountability managers that are (pseudo)randomly determined by hashing the peer's pseudonym P. In this way, P cannot choose her accountability managers, which makes the latter more likely to perform their duty impartially and therefore honestly.

In our framework, the inputs provided by clients and the outputs they obtain cannot be unequivocally linked to them, even though such inputs and outputs may be seen by some other peers. Our privacy guarantee is predicated on unlinkability rather than on confidentiality. *If inputs or outputs are such that it is not acceptable to disclose them even unlinkably or such that their very values or formats make them linkable to certain clients, then our framework cannot be used.*

Clients are publicly known to each other (they are neither anonymous nor pseudonymous), because in general, a peer wants to know with whom she is engaging in joint computation. Forwardees, workers and accountability managers are pseudonymous. A client also uses a pseudonym to interact with forwardees, workers and accountability managers; clients do not know each others' pseudonyms. A peer's

pseudonym P_i and her public identity ID_i are related as $P_i = H(ID_i, nonce_i)$, where $H(\cdot)$ is a one-way hash function and $nonce_i$ is a random number only known to peer ID_i; in this way, the pseudonym P_i does not leak the underlying real identity ID_i, but the link between both can be proven if necessary by revealing $nonce_i$. During a protocol execution, *any peer can play any of the above four roles* (client, worker, forwardee, accountability manager).

We assume that *peers are rational*: given appropriate incentives they will honestly fulfill their roles in the protocol, but they may be curious to learn the inputs or outputs of specific clients. We will design protocols so that these rational peers find no incentive to deviate. However, there may be a minority of malicious peers that are *irrational*, that is, who are ready to deviate from the protocols even if doing so places them in a worse position.

Although we will show how to incentivize rational peers to adhere to our protocols, we cannot guarantee that *clients* will provide truthful inputs for any arbitrary multi-party computation; this can only be guaranteed if clients are assumed honest or honest-but-curious. For rational clients, input truthfulness depends on the specific MPC to be carried out. In some MPCs, a rational client may find incentives to provide a false input, *e.g.* if she knows her false input cannot be detected by the others but it allows her to be the only client that learns the correct output. However, in other MPCs the rational behavior is to provide correct inputs. Take for example the millionaires' problem [4]: if millionaire A inputs an amount higher than her actual fortune and the output is that A is richer than the other millionaire B, then A does not learn whether she is really richer or poorer than B; similarly, if A inputs an amount lower than her actual fortune and the output is that A is poorer than B, then A does not learn whether she is really poorer or richer than B.

The main aim of a rational client is to obtain a correct output of the joint computation and to keep her inputs and outputs confidential from the other clients and the other pseudonymous peers. As we will show in the next sections, a client needs to have a high reputation to fulfill the previous aim. The incentive for a rational worker or a rational forwardee to cooperate is to increase her reputation in order to be able to become a successful client. Finally, the incentive for a rational accountability manager is to preserve fairness in the community. More details on the rational behavior of all peers are given in the next sections.

In some applications reputation alone may be deemed an insufficient incentive,

for example because working to build up a high reputation entails substantial financial costs in equipment, bandwidth, electricity, etc. To mitigate this shortcoming, reputation can be periodically converted into payments. In fact, rational incentives to correct computation in the form of payments are proposed in [56], where they are implemented by a central "boss", and in [57, 58], where they are embodied in smart contracts in blockchains. However, unlike in our approach based on reputation only, incentives in these proposals require external payment infrastructures that may not (always) be available or that may themselves be centralized or not rationally sustainable.

5.3.2 Requirements

At the end of the previous section we have mentioned reputation as the "currency" that incentivizes peers. In order for reputation to be effective, the following requirements need to be fulfilled:

- *Reward.* If a worker correctly performs a certain computation for a client, the worker's reputation must increase.
- *Punishment.* If a worker incorrectly performs a certain computation for a client, the worker's reputation must decrease. Similarly, a client that does not follow the protocols as prescribed must be punished with a reputation decrease.
- *Probabilistic reward.* A peer acting as a forwardee for inputs or outputs should be motivated by a nonzero probability of obtaining a reputation increase.
- *Reputation utility.* Having high reputation must be attractive for peers. Specifically, the higher the reputation of a client, the easier it is for the client to retrieve her outputs while preserving her privacy. For a pseudonymous peer, a high reputation is the way to become a successful client.

5.3.3 General-purpose MPC in the honest-but-curious model

For the sake of clarity, we first present a basic version of our protocol suite assuming that all peers are honest-but-curious.

Several nodes in the P2P network decide to engage in a joint computation and play the role of clients. The number of clients must be at least 4: with only 2 clients,

if a client sees an input or an output that is not hers, she knows it corresponds to the other client; with only 3 clients, unequivocal inferences are still possible, as justified in our privacy analysis of Section 5.5.1.

In Protocol 2, each client P secretly selects a worker P_w (who does not know who P is). Every worker receives the inputs of all clients in an unlinkable way via the co-utile anonymous channel FWD-CH (Protocol 3). Once all workers have received all inputs, each client P also sends to her P_w via FWD-CH the computation to be performed and an encryption key to be used by P_w to return the results to P via the reverse anonymous channel REV-CH (Protocol 4). Unlinkability in FWD-CH is achieved by random hopping among peers until a peer submits the message to the destination worker. REV-CH backtracks the hopping path of FWD-CH up to the client.

The idea of using an anonymous channel for MPC was first proposed in [59]; the novelty here is that we present an anonymous channel that does not depend on any central authority and is rationally sustainable.

Note 7 (Sending input to all peers)**.** Line 6 of Protocol 2 shows that the input values are sent to all network participants. This is necessary so that all workers receive the input values from all clients, without forcing them to share the workers they have selected. In fact, a node does not know that it is a worker until she receives the message with the computation to be performed (line 8).

5.3.4 General-purpose MPC in the rational model

In the rational model peers may deviate from their prescribed behavior if they are not properly incentivized. For example, a worker might return a random output without actually performing a computation that could be hard, or a messenger might not forward received messages, in both cases with the objective of safeguarding his own resources. We will add two mechanisms to cope with this problem: (i) redundancy to detect wrong computation or wrong forwarding; (ii) decentralized reputation to reward correct computation and correct forwarding, and punish wrong behavior.

In the protocols in this section, we assume the public-key cryptosystem used to encrypt to peers is *probabilistic*, that is, it uses randomness so that an observer cannot determine whether two ciphertexts correspond to the same cleartext. Also, we assume all messages encrypted under this public-key cryptosystem can be made

Protocol 2: HONEST-BUT-CURIOUS GENERAL-PURPOSE MPC

1 Clients $ID_1, \ldots, ID_m$ among the n peers (where m, n are public and $4 \leq m \leq n$) know each other and agree to jointly perform the computation $(O_1, O_2, \ldots, O_m) = C(I_1, I_2, \ldots, I_m)$, where input I_i and output O_i must stay private to ID_i;

2 **for** $i = 1$ **to** m **in parallel do**

3 Client ID_i uses her pseudonym P_i;

4 P_i prunes the code C into the part C_i that computes O_i, *i.e.* $O_i = C_i(I_1, I_2, \ldots, I_m)$; `/* In case` C_i `requires knowledge of which of the inputs is` P_i`'s,` I_i `is also embedded in` C_i `*/`

5 P_i randomly and secretly selects a peer P_{w_i} among the n peers as a worker;

6 **for** $l = 1$ **to** n **do**

7 P_i calls FWD-CH$(P_i, I_i || nonce_i, nil, P_l)$; `/*` P_i `sends her private input` I_i `to all peers` P_l `via the FWD-CH`$(\cdot)$ `anonymous channel, so that in particular` P_{w_i} `gets it.` I_i `is appended a random nonce to make it unique. */`

8 P_i calls $O_i =$ FWD-CH$(P_i, PK_{w_i}(K_i), C_i, P_{w_i})$. `/*` P_i `sends to` P_{w_i} `a key` K_i `encrypted under` P_{w_i}`'s public key and the computation` C_i `(a peer's public key may be her pseudonym). Then` P_{w_i} `will return` O_i `symmetrically encrypted under` K_i `via the reverse channel. */`

Protocol 3: FWD-CH($P_s, msg, comp, P_d$)

Parameter $p \in [0, 1]$;
if $P_s = P_d$ **then**
  if *comp* $= nil$ *AND msg carries a new nonce* **then**
    P_d extracts the input I in msg and appends it to $Ilist$;
    /* When $comp = nil$, msg is an input I; $Ilist$ is initially empty and is appended all inputs with different nonces received by P_d in FWD-CH calls with $comp = nil$ */
  else
    P_d waits until $Ilist$ contains m different inputs;
    P_d performs computation $out = comp(Ilist)$;
    P_d cleans $Ilist$;
    P_d decrypts $K = SK_d(msg)$;
    /* When $comp \neq nil$ and all inputs have been received, P_d performs computation $comp$ on the received $Ilist$; also, $msg = PK_d(K)$, where the symmetric key K is to be used by P_d to encrypt the output of $comp$ before returning it */
    P_d calls REV-CH($P_d, E_K(out), comp, P_{prev}$);
    /* REV-CH returns the encrypted output of $comp = C_i$ to originator P_i without knowing who P_i is; P_{prev} is the pseudonym of the peer from whom P_d received $comp$ */
else
  if P_s *is the originator of* msg **then** $p_{forward} = 1$ **else** $p_{forward} = p$;
  /* The originator always hops but other peers hop with prob. p */
  if *Bernoulli*($p_{forward}$) $= 1$ **then**
    P_s randomly chooses another peer P';
    P_s sends $(msg, comp)$ to P';
    P' calls FWD-CH($P', msg, comp, P_d$);
  else
    P_s directly sends $(msg, comp)$ to P_d;
    P_d calls FWD-CH($P_d, msg, comp, P_d$).

Protocol 4: REV-CH($P_s, E_K(out), comp, P_{prev}$)

1 P_s sends $(E_K(out), comp)$ to P_{prev};
2 **if** P_{prev} *knows the key* K **then** /* P_{prev} is the client */
3 P_{prev} decrypts *out*; /* *out* is P_{prev}'s private output O_{prev} */
4 **else**
5 Let P_{prev2} be the peer from whom P_{prev} received the FWD-CH message with *comp*;
6 P_{prev} calls REV-CH($P_{prev}, E_K(out), comp, P_{prev2}$). /* P_{prev} backtracks to P_{prev2} */

of the same length, if necessary by padding the cleartext, so that ciphertext length cannot be used by an observer to guess information on the cleartext. Additionally, we assume peers can digitally sign messages.

The main protocol

Protocol 5 is the version of Protocol 2 augmented with redundancy (r different workers are used by each client) and decentralized reputation. In Protocol 5 we use co-utile versions C-FWD-CH (Protocol 7) and C-REV-CH (Protocol 8) of the previous FWD-CH and REV-CH auxiliary protocols. For the rest and unless otherwise said, the notations are the same as in Protocol 2.

Each client P secretly selects r workers P_w and sends her input to all peers via Protocol 7. Afterwards, each client P sends to her workers P_w via the same channel the computation C_i to be performed and the encryption key K to be used by P_w to return the results. As soon she received all the expected outputs (via Protocol 8) from her workers, each client proves to her accountability managers that she has rewarded the first forwardee, compares the outputs with each other and considers the most frequent the final output. Finally, she rewards or punished every worker depending on the output received.

The main idea of the protocols is that a node will only interact with other nodes of similar reputation (g_i). For example, a worker will only accept to compute for a client with a similar reputation, or a forwardee will only accept to forward a message if it comes from another node with a similar reputation. In this way, nodes with low reputations (due to bad computations or bad behavior) become excluded without the possibility of interaction with other nodes, and therefore,

without the possibility of receiving correct results.

Note 8 (Newcomers)**.** By design, a newcomer enters the system with the lowest possible global reputation $g = 0$, which is below the interaction threshold of almost every other peer. If we took no action, newcomers would never be selected by the honest nodes to be their workers since the difference between their reputations is too great. They would only be able to compute for the malicious nodes that would get correct computations thanks to the honest newcomers that enter the system. Therefore, when a node enters the system, for a certain number of iterations it is allowed to be chosen by the clients as a worker whatever the difference between the reputations. In this way, the node will be able to increase her reputation as long as it executes the computations correctly. However, on the other hand, in order to prevent malicious nodes from exiting the system and re-entering under a different pseudonym (whitewashing), newcomers will not be able to be clients and will not obtain any results.

Note 9 (On parameter κ_i)**.** Parameter κ_i is secret to each client P_i. If $\kappa_i \gg r$, then workers are selected from a large set, which makes it more difficult for P_i's workers to guess that their client is P_i. On the other hand, too large a κ_i is also risky for P_i, because it can result in choosing workers with reputation much higher than P_i's (who are likely to refuse working for P_i) or workers with much lower reputation (who may be unreliable).

Note 10 (On sharing workers)**.** A way to reduce the computation and communication of Protocol 5 while still providing redundancy would be for clients to share their workers, rather than each client choosing her own r workers. For example, if each client proposed one worker, a pool of m workers would be available to every client; to achieve the same redundancy level, Protocol 5 requires using m^2 workers. Yet sharing workers comes at a price. First, redundancy based on sharing workers means that the same computation C must be run for all clients. Second, clients are forced to trust workers that have been selected by other clients. This is particularly bad if the reputations of clients are very heterogeneous: high-reputation clients could find workers with higher reputation (and thus more reliable) than the workers proposed by the other clients. Also, the worker suggested by a client P_j might collude with P_j against other clients (see Section 5.5.1 on collusions).

Protocol 5: RATIONAL GENERAL-PURPOSE MPC

Clients $ID_1, \ldots, ID_m$ among the n peers (where m, n are public and $4 \leq m \leq n$) know each other and agree to jointly perform the computation $(O_1, O_2, \ldots, O_m) = C(I_1, I_2, \ldots, I_m)$, where input I_i and output O_i must stay private to ID_i;
for $i = 1$ **to** m **in parallel do**
    Client ID_i uses her pseudonym P_i;
    P_i prunes the code C into the part C_i that computes O_i, *i.e.* $O_i = C_i(I_1, I_2, \ldots, I_m)$;
    P_i secretly selects as workers r peers $P_{i_1}, \ldots, P_{i_r}$ randomly chosen among the $\kappa_i > r$ peers with closest reputation to g_i and newcomers;
    for $l = 1$ **to** n **do**
        P_i calls C-FWD-CH$(P_i, PK_l(I_i||nonce_i), PK_l(nil), P_l)$;
    for $k = 1$ **to** r **do**
        P_i calls $O_{i,k}$ = C-FWD-CH$(P_i, PK_{i_k}(K_i), PK_{i_k}(C_i), P_{i_k})$;
        forall *accountability managers AM of* P_i **do**
            if *P_i does not show to AM a reward receipt from the first forwardee* **then** `/* Not rewarding the first forwardee is punished */`
                AM assigns local reputation $\ell_{AM,i} = 0$ to P_i;
    Take as O_i the most frequent value in $\{O_{i,1}, \ldots, O_{i,r}\}$;
    for $k = 1$ **to** r **do**
        if $O_{i,k} = O_i$ *AND* $O_i \neq nil$ **then**
            P_i assigns $\ell_{ii_k} = 1$; `/* Majority result is rewarded */`
        else
            P_i assigns $\ell_{ii_k} = 0$ `/* Minority result is punished */`
Call Protocol 6 to update the public global reputation g_i of each peer P_i.

Protocol 6: Co-utile P2P global reputation computation

1 **for** $i = 1$ **to** n **do** `/* EigenTrust-like global reputation */`
2 P_i submits local reputation values $\{\ell_{ij} : j = 1, \ldots, n\}$ to all M accountability managers of P_i;
3 **forall** *pupils* P_d *of* P_i **do**
4 P_i collects local reputation values $\{\ell_{dj} : j = 1, \ldots, n\}$;
5 P_i normalizes $c_{dj} = \ell_{dj} / \sum_j \ell_{dj}$, $j = 1, \ldots, n$;
6 **forall** *pupils* P_d *of* P_i **do**
7 **for** $j = 1$ **to** n **do**
8 P_i queries all the accountability managers of P_j for $c_{jd} g_j^{(0)}$;
9 $k := -1$;
10 **repeat**
11 $k := k + 1$;
12 P_i computes $g_d^{(k+1)} = c_{1d} g_1^{(k)} + c_{2d} g_2^{(k)} + \ldots + c_{nd} g_n^{(k)}$;
13 **for** $j = 1$ **to** n **do**
14 P_i sends $c_{dj} g_d^{(k+1)}$ to all M accountability managers of P_j;
15 P_i waits for all accountability managers of P_j to return $c_{jd} g_j^{(k+1)}$;
16 **until** $|g_d^{(k+1)} - g_d^{(k)}| < \epsilon$; `// Parameter` $\epsilon > 0$ `is a small value`;

Protocol to send messages via the co-utile anonymous channel

Protocol 7 (C-FWD-CH) shows the operation of the co-utile anonymous channel. When a node receives a message, she firstly checks whether it was sent by a node with a higher or similar reputation (lines 21, if the node is a messenger, and 28, if the node is the recipient worker). If the reputation of the sender is too low, the node discards the message. If she accepts the message but she is not the recipient worker, she should decide randomly whether to send the message to the worker or to another forwardee (line 18). If she decides to hop to another forwardee, she will send the message to a forwardee with similar or higher reputation (see Note 12).

The workers will collect the received inputs from all clients (line 6), and will execute the required computation as soon as they receive m inputs and C_i (line 12). The output, encrypted under key K, will be sent back to the clients via Protocol 8 (C-REV-CH).

Note 11 (On the flexibility parameter δ)**.** In Protocol 7 a peer P_j does not discard a message from a peer P_i as long as P_i's reputation g_i is at least $g_j - \delta$, where δ is a small reputation amount. The value δ introduces some flexibility in the interaction and helps new peers (that start with 0 reputation) to earn reputation as first forwardees. A large δ is not acceptable from the rational point of view: high-reputation peers have little to gain by accepting messages or computations from peers much below them in reputation (if those low-reputation peers are the clients, they may not reward them).

Note 12 (On function SELECT)**.** In Protocol 7 function $P_t = \text{SELECT}(g_s, g_d)$ is used by a peer P_s to select a peer P_t as a forwardee towards the worker P_d. There are several ways in which this can be done. However, the rational choice is for P_s to select a forwardee P_t with a sufficient reputation so that P_d does not refuse to compute should P_t directly submit to the worker. Hence, if P_s's reputation is $g_s \geq g_d - \delta$, P_s can randomly pick any of the peers whose reputation lies in $[g_d - \delta, g_s + \delta]$: any forwardee P_t with reputation at least $g_d - \delta$ does not risk refusal from the worker P_d, but peers P_t with reputation above $g_s + \delta$ will discard P_s's messages. On the other hand, if $g_s < g_d - \delta$, P_s chooses the peer with the maximum reputation that does not exceed $g_s + \delta$, because any peer with reputation above that value will discard P_s's message.

Protocol 7: C-FWD-CH($P_s, msg, Ecomp, P_d$)

Parameter $p \in [0, 1]$;
if $P_s = P_d$ **then**
  P_d decrypts $comp = SK_d(Ecomp)$;
  if $comp = nil$ **then** `/* In this case,` msg `contains an input */`
    P_d decrypts $I||nonce = SK_d(msg)$;
    if P_d *has not previously received nonce* **then** P_d appends I to $Ilist$;
  else
    if $comp =$ *"refuse"* **then** `/* The worker has refused to compute */`
      P_d assigns $out := nil$; `/* The worker returns no output */`
    else
      P_d waits until $Ilist$ contains m different inputs;
      P_d computes $out := comp(Ilist)$; `/* The worker computes */`
    P_d cleans $Ilist$;
    P_d recovers $K = SK_d(msg)$;
    P_d calls C-REV-CH($P_d, E_K(out), Ecomp, P_{prev}$);
else
  if P_s *is the originator of* msg **then** $p_{forward} = 1$ **else** $p_{forward} = p$;
  P_s computes $decision =$ Bernoulli($p_{forward}$);
  if $decision = 1$ **then** `/*` P_s `decides to hop */`
    P_s sends $(msg, Ecomp)$ to $P_t = \textsc{Select}(g_s, g_d)$ `/* See Note 12 about function` SELECT `*/`;
    if P_s*'s reputation is at least* $g_t - \delta$ **then** `/* See Note 11 about` δ `*/`
      P_t calls C-FWD-CH($P_t, msg, Ecomp, P_d$);
    else
      P_t discards $(msg, Ecomp)$;
  else
    P_s directly sends $(msg, Ecomp)$ to P_d `/*` P_s `decides to submit to the worker */`;
    P_d decrypts $comp = SK_d(Ecomp)$;
    if $comp \neq nil$ *and* P_s*'s reputation is less than* $g_d - \delta$ **then**
      P_d sets $Ecomp := PK_d$("refuse"); `/*` P_d `refuses to compute */`
    P_d calls C-FWD-CH($P_d, msg, Ecomp, P_d$);

Protocol to send outputs via the reverse channel

Once the computation has been performed, every worker sends back the output via Protocol 8 (C-REV-CH), since, as she does not know the client's identity, she is not able to use the anonymous channel. The reverse path is followed until the output reaches client P_{prev} (who can decrypt the message since she knows key K), and the client rewards the first forwardee P_s. The first forwardee returns the receipt that the client has to show to her accountability managers in line 11 of Protocol 5.

Protocol 8: C-REV-CH($P_s, E_K(out), Ecomp, P_{prev}$)

1 P_s sends $(E_K(out), Ecomp)$ to P_{prev};
2 **if** P_{prev} *knows the key* K **then**
3 P_{prev} decrypts *out* `/* If` P_{prev} `knows` K`, then` P_{prev} `is the client and` P_s `the first forwardee */`;
4 P_{prev} sends S_{prev}("P_{prev} has set $\ell_{prev,s} = 1$") to P_s ; `/*` P_{prev} `commits to rewarding the first forwardee */`
5 P_s forwards S_{prev}("P_{prev} has set $\ell_{prev,s} = 1$") to P_{prev}'s AMs;
6 P_{prev}'s AMs set $\ell_{prev,s} = 1$;
7 P_s returns a signed reward receipt S_s("P_s acknowledges $\ell_{prev,s} = 1$") to P_{prev};
8 **else**
9 Let P_{prev2} be the peer from whom P_{prev} received the C-FWD-CH message with $Ecomp$;
10 P_{prev} calls C-REV-CH($P_{prev}, E_K(out), Ecomp, P_{prev2}$).

Note 13 (On rewarding the first forwardee only)**.** In Protocol C-REV-CH only the first forwardee is rewarded, rather than all forwardees, and only when the computation to be done is not *nil*. Note that in Step 5 of Protocol 6 the local reputations awarded by a peer are normalized before updating global reputations (this is done to prevent peers from increasing their influence by giving more opinions on other peers). Hence, if all forwardees were rewarded by the client, the reputation increase of each rewarded forwardee would be smaller. Thus, every forwardee would be better off by sending *msg* directly to the destination peer rather than forwarding it to another forwardee. As a consequence, there would be only one forwardee, who would know that the previous peer is the client who

originated *msg*. This would break the anonymity of the channel. Rewarding only the first forwardee when the computation to be done is not *nil* avoids this problem and is a sufficient incentive: the forwardees do not see which is the computation to be performed (it is probabilistically encrypted under the worker's public key) and any forwardee can hope to be the first for a non-nil computation and thus has a reason to collaborate.

5.4 Co-utility analysis

We argue that the framework formed by Protocols 5, 6, 7 and 8 is co-utile, that is, that these protocols will be adhered to by the rational players. We first give a sketch justification and then we analyze in detail the motivation of each of the above player categories to adhere to the protocols they are required to participate in. The sketch is as follows:

- The clients' goal is to perform a joint computation and obtain their respective correct outputs, while keeping their own inputs and outputs private. For that reason, the clients can be assumed to correctly perform their tasks in Protocols 5, 7 and 8. If a client fails to behave as prescribed (when acting as a client, as a forwardee or as a worker), her reputation will decrease and it will be more difficult for him/her to obtain correct results.

- Forwardees have no role in Protocol 5, but they are essential to the operation of the co-utile anonymous channel in Protocol 7 (C-FWD-CH) and Protocol 8 (C-REV-CH). Their incentive is the hope to be rewarded in Protocol 8 with a reputation increase in case they turn out to be the first forwardee.

- Workers are expected in C-FWD-CH to perform the required computation. Then in C-REV-CH they are expected to start the reverse path upstream to return the output. Their incentive to compute correctly is to be rewarded in Protocol 5 if the output they deliver is the majority output among those returned by redundant workers.

- Accountability managers have important roles in Protocols 5, 6 and 8. In our security model (Section 5.3.1), peers are assumed to be rational, even if rational attackers are not excluded. Given that peers interact in successive iterations, the interest of rational accountability managers is to favor correct computations, as they may be clients themselves in subsequent computation

rounds. On the other hand, the fact that the M accountability managers of every peer are (pseudo)randomly assigned thwarts conflicts of interest and facilitates honest management of the peer's reputation. Furthermore, if necessary, a countermeasure could be added right after Step 8 of Protocol 6 whereby P_i punishes with local reputation 0 those AMs that provide local reputations for P_j that do not agree with the majority reputation value.

Next, we elaborate on co-utility for each player category.

5.4.1 Co-utility for clients

In Protocol 5, $ID_1, \ldots, ID_m$ can be assumed to honestly agree on the joint computation, because jointly computing is their goal. Then at Step 3 they are also interested in switching to their respective pseudonyms $P_1, \ldots, P_m$. If a client used her real identity ID_i, in C-FWD-CH it would be trivial for a forwardee P_t to know whether she is the first forwardee (and hence the only forwardee that will be rewarded); in consequence, if P_t was asked to be a forwardee by a pseudonymous peer, P_t's rational decision would be to decline. In this way, there would be only one forwardee, which would weaken unlinkability for the clients.

Also in Protocol 5 it is rational for the pseudonymous clients to correctly perform steps up to Step 9. This means pruning C into their respective computations, selecting workers, and sending to the workers via the anonymous channel C-FWD-CH first their private inputs and then the pruned computation and the key for receiving encrypted outputs. Note that private inputs are sent *in parallel* at Step 7, so P_i cannot wait for the other peers to send their private inputs to P_i's workers and then free-ride without sending P_i's input to the other peers' workers. It is in all the clients' interest to perform honestly, as they want to obtain correct outputs. It is also rational for the clients to select newcomers as workers: the higher the number of honest nodes that participate, the more resistant to attacks the system will be. Hence, helping newcomers to demonstrate their behavior makes sense.

In Protocol 7, it is bad for the client P_i originating the message (called P_s inside the protocol) to directly send it to the worker P_{i_k} (called P_d inside the protocol), even if only with probability $1-p$, like in the Crowds system [45]. In that case, from the worker P_{i_k}'s viewpoint, the most likely sender of a message is the originating client P_i: indeed, P_i would send it with probability $1-p$, whereas the l-th forwardee would send it only with probability $p^l(1-p) < 1-p$. Hence, the originating client P_i is interested in looking for a forwardee, and therefore P_i will honestly

follow Steps 17 and 18. Further, the client P_i wants, if possible, a forwardee that will not risk refusal by the worker or, if this is not possible, a forwardee with the maximum possible reputation among those that will not discard P_i's message (see Note 12 for a justification). *Since a client P_i will only obtain her output O_i if she finds her own good forwardees and workers (the forwardees and workers of other clients do not take care of O_i), P_i is motivated to maintain a high reputation g_i.* An additional motivation for a client P_i to maintain a high reputation is that it allows P_i to randomly choose her first forwardee among a large set of peers that will not risk refusal by the worker, which increases unlinkability of successive messages sent by P_i. This ensures honest adherence to Step 20.

In Protocol 8, it is obviously in the interest of P_i (called P_{prev} inside the protocol) to decrypt her private output when she receives it (Step 3). Further P_i's best option is to reward the first forwardee by giving her local reputation 1 (Step 4), in order to obtain a reward receipt from the first forwardee (Step 7). This reward receipt is necessary for P_i to escape being punished with 0 local reputation in Step 12 of Protocol 5.

P_i could certainly decide to favor a false first forwardee P' of her choice, rather than the real first forwardee P. This would still work well for P_i, because P' would return a signed receipt for the same reasons that P would do it. However, if P_i wants to favor P', it entails less risk (of being discovered) for P_i to use P' as a *real* first forwardee. Thus, there is no rational incentive for clients to favor false first forwardees.

5.4.2 Co-utility for forwardees

In Protocol 7, there may be two types of forwardees:

1. P_s is a forwardee if she is not the originator of msg. P_s's incentive to perform her role properly is the hope of being the first forwardee *for a non-nil computation* and hence be rewarded with a reputation increase. Note that P_s does not know whether she is the first and whether the computation is non-nil, because the latter is encrypted and timing does not help, given that the C-FWD-CH calls with nil and non-nil computations are interleaved due to the hopping mechanism among forwardees. According to the protocol, P_s must decide between forwarding $(msg, comp)$ to some other peer P_t or directly sending $(msg, comp)$ to the destination peer P_d who will perform $comp$. Both actions take about the same effort, so it is rational for P_s to

make the decision randomly according to the prescribed probabilities (p for forwarding and $1-p$ for directly sending to P_d). In case of forwarding, P_s's rational procedure is like the originator's: use the SELECT function.

2. P_t is the forwardee selected by P_s in case of forwarding. P_t must decide on message acceptance or discarding. The incentive for P_t to accept to deal with a message is the hope to increase her reputation if P_t happens to be the first forwardee for a non-nil computation (which P_t does not know). However, P_t will not accept to deal with a message coming from P_s if the latter's reputation is too low: it might be a sign that P_s did not "pay" previous first forwardees in Step 4 of Protocol 8 and was therefore punished with reputation decrease in Step 12 of Protocol 5. Thus, the rational decision is for P_t to accept to deal with P_s's message only if P_s is not too inferior to P_t in reputation, that is, if $g_s \geq g_t - \delta$. Otherwise, in Step 24 P_t discards P_s's message.

In Protocol 8, forwardees backtrack along the reverse path from the worker to the originator. There are two forwardee roles:

1. In Step 5 P_s is the first forwardee because P_{prev} was the originator under C-FWD-CH. Hence, at Step 5 P_s is obviously interested in forwarding to P_{prev}'s AMs the reward P_s receives from P_{prev}. On the other hand, at Step 7 P_s's best option is to return the reward receipt to P_{prev}, because P_{prev} could otherwise blacklist P_s and never make P_s a first forwardee in future joint computations.

2. In Step 10 P_{prev} does not know whether she is the first forwardee or not. In the hope of being the first forwardee, P_{prev}'s best option is to forward $(E_K(out), comp)$ to the next upstream peer P_{prev2}. If P_{prev2} turns out to be the originator, then P_{prev} will be rewarded as the first forwardee (see previous item).

5.4.3 Co-utility for workers

In Protocol 7 at Step 5 P_d is the worker to whom the private inputs are sent by the clients. Decrypting and storing these inputs is necessary for P_d to be able to correctly perform the computation at Step 12. The motivation for P_d to return the correct computation result encrypted under the retrieved key K in Step 15 is

to receive a reputation reward in Step 16 of Protocol 5, rather than a reputation punishment in Step 18 of that protocol.

In Protocol 8, the worker P_d (called P_s within the protocol) merely sends ($E_K(out)$, $Ecomp$) to the previous peer in the path followed by C-FWD-CH. The worker's motivation to do this is to earn a reputation reward in Step 16 of Protocol 5.

Note that, workers being peers, they are also likely to be clients at some point. And for a client having a high reputation is the way to easily find forwardees that help preserve the client's input and output privacy.

5.4.4 Co-utility for accountability managers

In Protocol 5, the accountability managers punish those clients that do not reward their first forwardee (Step 12). Since they are pseudorandomly chosen, most AMs are rational and thus interested in favoring correct computation; hence, most AMs will discharge their role as described in the protocol.

Protocol 6 is run by all peers in their capacity of accountability managers. Since most AMs are rationally interested in favoring correct computation, they are also interested in correctly updating and maintaining global reputations. Therefore, most AMs can be assumed to run Protocol 6 correctly. For the same reason, all peers will supply local reputations to their AMs at the start of the protocol.

Finally, in Protocol 8 the role of AMs is to reflect the reputation rewards received by the first forwardees. Again, the rational interest of the AMs to preserve the correct operation of the protocol suite allows expecting them to do their job.

The bottom line that allows trusting AMs as a community is that they are pseudorandomly chosen and redundant (each peer has M AMs), coupled with the assumption that a majority of peers (and hence of AMs) is rational. Yet, it might be argued that assuming a majority of rational peers is not the same as assuming a majority of honest peers. If it is feared that a sizeable proportion of AMs might have rational motivations to deviate, additional countermeasures could be set up to punish bad AM behavior. For example, as hinted earlier in the sketch at the beginning of this section, after Step 8 of Protocol 6 P_i could punish with local reputation 0 those AMs that provide local reputations for P_j that do not agree with the majority reputation value. Yet, to avoid complicating pseudocodes, we have refrained from including such optional countermeasures.

5.5 Privacy and correctness

Here we examine how the protocols satisfy the requirements of Section 5.3.2, and thereby preserve the confidentiality of the peers' private inputs and outputs and incentivize correct computation by the peer workers.

5.5.1 Privacy

In our protocol suite, the privacy of client inputs and outputs is based on the co-utile anonymous channel implemented by protocols C-FWD-CH (downstream) and C-REV-CH (upstream). This channel breaks the link between clients and their inputs and outputs.

We first show that, in general, no collusion of peers can link with certainty an input to the corresponding client. Then we deal with the unlinkability of outputs and we argue that the only collusion that could link an output to the corresponding client is hardly rational. After showing that collusions are unproductive to link inputs and either unproductive or not rational to link outputs, we prove that in the absence of collusion a client can plausibly deny that a certain input is hers and that a certain output is hers.

Proposition 4. *If a client P_i's input I_i is not embedded in her computation C_i, no collusion of peers can link with certainty an input to P_i.*

Proof. A successful collusion must include at least a first forwardee (who knows the client's pseudonym) and a worker (who computes the client's output). Under the assumption of the proposition, client P_i sends her input only at Step 7 of Protocol 5, via C-FWD-CH. In this call to C-FWD-CH, P_i does not reward the first forwardee, which means that the latter does not learn she is the first forwardee. Hence the worker cannot find a first forwardee that knows for sure the pseudonym of the client to whom a specific input corresponds. □

Regarding outputs, when a client P_i calls C-FWD-CH to send the return key and the computation (Step 9 of Protocol 5), P_i does reward the first forwardee. Therefore, this first forwardee learns P_i's pseudonym and could collude with the worker to link the output to the client's pseudonym. Nonetheless, such a collusion is hardly rational:

- Peers are pseudonymous: the first forwardee only knows the worker's pseudonym and the worker does not even know the first forwardee's pseudonym unless the first forwardee tells the worker. People tend to collude with those they know, and thus pseudonymity is a defense against collusion.
- There is an asymmetry between the worker and the first forwardee. It is unclear why the worker should share with the colluding first forwardee the value of the client's private output after the first forwardee reveals the client's pseudonym: the worker cannot verify that the pseudonym actually corresponds to the client, and even if the worker accepts the pseudonym as good, it is not the client's real identity.
- In case a first forwardee or a worker P_t is also a client herself, colluding to link another client P_v's output with P_v's pseudonym makes P_t's own output more easily linkable (linkage possibilities reduce from m clients to $m-1$ after a successful collusion).

A final mitigating factor is that, in many joint computations the outputs are either not private (*e.g.* the tally in e-voting is public) or less confidential than the inputs.

Proposition 5. *If there is no collusion between the peers, a client P_i can plausibly deny that a certain private input is hers and can also plausibly deny that a certain private output is hers. Hence, client privacy would still hold if the mapping between the client's pseudonym P_i and the client's real identity ID_i was discovered.*

Proof. The privacy guarantee is based on unlinkability and input/output encryption.

Let us examine whether a worker can link an input to the corresponding client. By the design of Protocol C-FWD-CH, a worker P_{j_k} knows that when P_{j_k} receives $PK_{j_k}(I_i)$ from a peer P, P is unlikely to be the client P_i to whom I_i corresponds (a client always chooses to hop if she can). Yet, since we assume in Protocol 5 that $m \geq 4$, there are at least 4 clients; hence, even if both P_{j_k} and P happened to be also clients themselves, P_{j_k} cannot unequivocally determine which of the remaining clients is P_i. What P_{j_k} can do is to estimate the probability that $PK_{j_k}(I_i)$ was submitted by the l-th forwardee as $p^{l-1}(1-p)$, and hence that the most likely submitter is the first forwardee. Nevertheless, the first forwardee is

chosen by client P_i using the SELECT function, described in Note 12. If $g_i \geq g_d - \delta$, then P_i chooses the first forwardee randomly among the set of peers with reputation in the interval $[g_d - \delta, g_i + \delta]$, and this set depends on the current reputations and varies over time; hence, as long as there are several peers with reputations in the previous interval, even if P_{j_k} received two encrypted inputs from the same peer in two different successive joint computations, P_{j_k} cannot infer that both inputs correspond to the same client. If $g_i < g_d - \delta$, then P_i chooses as a first forwardee the peer with the maximum reputation that does not exceed $g_i + \delta$: if reputations do not change between two successive messages, P_i would choose the same first forwardee for both messages; yet P_d cannot be sure that the submitter of both messages is really the first forwardee, and hence P_d cannot be sure that both messages were generated by the same client. Hence, in no case can two different messages sent by the same client be unequivocally linked, even if the probability of correctly linking them is lower when $g_i \geq g_d - \delta$.

On the other hand, the worker P_{i_k} receives the computation C_i via C-FWD-CH from the same client P_i to whom output O_i must be returned via C-REV-CH. However, by the design of C-FWD-CH P_{i_k} receives C_i without learning who P_i is. Also, by the design of C-REV-CH P_{i_k} returns the client's output O_i by hopping upstream via the reverse path, with no knowledge of P_i's pseudonym or identity. Certainly, P_{i_k} could try to identify the client P_i by looking for which peers P_{i_k} is closest in reputation; however, P_{i_k} does not know the parameter κ_i (number of closest peers) used by P_i in Protocol 5 when choosing her workers.

Consider now linkability by a forwardee P_s. If P_s is a forwardee of a message from P_{prev}, in general P_s does not know whether P_{prev} originated the message or is merely forwarding it. The only exception is when P_s is the first forwardee (because in this case P_s receives a message from P_{prev} in Step 4 of Protocol 8). Yet, in this case P_s can only link the *encrypted* version of the output (that is, $E_K(out)$) to the client's identity P_{prev}.

Lastly, if AM is an accountability manager of a client P_i, AM only sees the reward receipts from the first forwardees of that client (see Protocol 5). In no case can AM access the inputs or the outputs of P_i.

Since no input or output can be unequivocally attributed to a client by any other peer, plausible deniability by the client holds. □

5.5.2 Correctness

To ensure correctness, the delivery of correct outputs must be the best rational option for the non-client peers (forwardees and workers). Protocols 5, 7 and 8 are designed to incentivize this option. Thus, from the discussion on co-utility for workers and forwardees in Section 5.4 the following proposition follows:

Proposition 6. *The rational behavior of workers and forwardees in Protocols 5, 7 and 8 is to return correct outputs to the clients.*

5.6 Example applications

Our framework is totally general and therefore it can accommodate any joint computation with input and output confidentiality. However, for the sake of illustration, we sketch two specific applications.

5.6.1 Finding differences with the previous and following parties in a ranking or an auction

In this application, each client P_i gives as private input I_i her value for a certain magnitude (*e.g.* wealth or auction bid) and wants as private output O_i the difference with the previous party in the ranking by that magnitude (*e.g.* how richer is the next richer client or how much more did the next higher bidder offer) and the difference with the following party in the ranking (*e.g.* how poorer is the next poorer client or how much less did the next lower bidder offer).

In this case, the computation C consists of ranking all clients and finding the differences between successive clients in the ranking. The pruned computation C_i of interest for P_i consists of ranking all clients and finding only the differences between P_i and its previous client and between P_i and its following client. Hence, in this application, P_i's input I_i must be embedded in C_i (yet this is no problem, because the worker running C_i cannot link I_i to P_i).

5.6.2 Electronic voting

In electronic voting, the clients $P_1, \ldots, P_m$ are the voters. The private inputs of the clients $I_1, \ldots, I_m$ are their secret votes. There is only one output O, the tally, which is public. Note the value of a vote does not reveal the identity of the voter, and hence it is safe for clients to send their votes to workers as long as they do it using an anonymous channel.

The computation C consists of computing the absolute frequencies of all options susceptible of being chosen by voters (*e.g.* candidates, parties, Yes/No/Blank etc.). In this case $C_1 = C_2 = \ldots = C_m = C$ and no client input needs to be embedded in C. Rather than each worker returning O to her client via C-REV-CH, it is simpler for all workers to publish O. If a worker publishes a tally O' that disagrees with the majority tally, O' will be discarded (and the worker's reputation will decrease). In fact, the computational overhead can be strongly reduced with the following simplification: each client chooses only *one* worker rather than r; in this case, there is a set of only m workers (rather than $m \times r$), but if $m \geq 4$, this set is enough to compute O as the majority output.

It is also interesting to consider the particular case in which all peers are voters/clients. Since all peers are clients and they are rational, correct computation of the common output (tally) interests them. Hence, most peers can be expected to behave honestly in their capacity of workers. As a consequence, *reputations are not needed* for the correct tally to be majority.

We can generalize the last remark into the following proposition.

Proposition 7. *If all peers are rational clients and there is a single common output of the joint computation, Protocols 5, 7 and 8 can be expected to yield a correct output even if all operations related to reputations are suppressed from the protocols.*

5.7 Experimental results

In this section, we report the results of the experiments with the proposed co-utile framework. On one hand, we evaluated how the rate of correct outputs received by the peers evolves over time depending on their behavior (ee considered good peers, who honestly do their job, and malicious peers, who do not). On the other hand, we compared the co-utile framework with a *baseline* framework in which there is worker redundancy but *no reputation* (that is, Protocols 5, 7 and 8 without using reputations to decide on workers, forwardees or clients).

Expected behavior. If the co-utile framework is well designed, good clients (that are good peers) should obtain a higher rate of correct outputs in it than in the baseline framework. Also, in the co-utile framework the rate of correct outputs for good clients should be higher than for bad clients (who are malicious peers). Further, the reputation of the peers should highly and positively correlate with

their goodness and with the rate of correct outputs they obtain. The rationale is that peers who perform wrong computations as workers are likely to end up having low reputation, and it is difficult for a peer with low reputation to find (as a client) other peers that are willing to perform a computation for her (as workers) or forward the former peer's messages (as forwardees). Moreover, the rate of correct outputs for an honest newcomer should be, in the medium term, very similar to the other honest clients. However, if a newcomer misbehaves, the rate of correct outputs should be always low, as the other malicious peers in the network. In this manner, malicious peers have no incentive to take a new virtual identity in order to "clean" their reputation after misbehaving with other peers.

Experimental setting. We built a P2P network with $n = 50$ peers and let it evolve for 150 iterations; in each iteration, a joint computation was conducted by $m = 10$ clients randomly chosen among the 50 peers. Each client gave a numerical value as a private input to the joint computation; the latter consisted of ranking the input values and returning as a private output the rank of the client's value. A malicious worker always returned a random output instead of the true output of a computation (so they kept a 10% chance of returning the correct result). Finally, we set $r = 3$ (three redundant workers per client).

5.7.1 Evaluation of the co-utile framework

To evaluate the co-utile framework, we set up a scenario with the parameters listed above and 20% of malicious peers. The remaining 80% were honest peers. Additionally, we introduced two nodes right after iteration 50: a malicious one and an honest one. A node will be considered newcomer during her first 25 iterations into the system. We gave an initial reputation $g = 1/n = 0.02$ to each client and we took $\delta = g * 0.75 = 0.015$.

Figure 5.1 shows the evolution of the reputations of every kind of node during the 150 iterations. The lines represent the average reputation of a particular type of node for the iteration in the abscissae, while the shaded regions represent the standard deviation. It can be seen that already in the first iterations, while the node behaviors are still being modeled, the reputation of malicious peers swiftly decreases. Around the ninth iteration, a clear difference exists between the reputation of honest and malicious peers. From iteration 50, the evolution of the reputation of the honest newcomer (blue) and the malicious newcomer (gray) appears. Since during their first 25 iterations after their entrance, the other nodes can select the newcomers as workers, the behavior of both nodes is

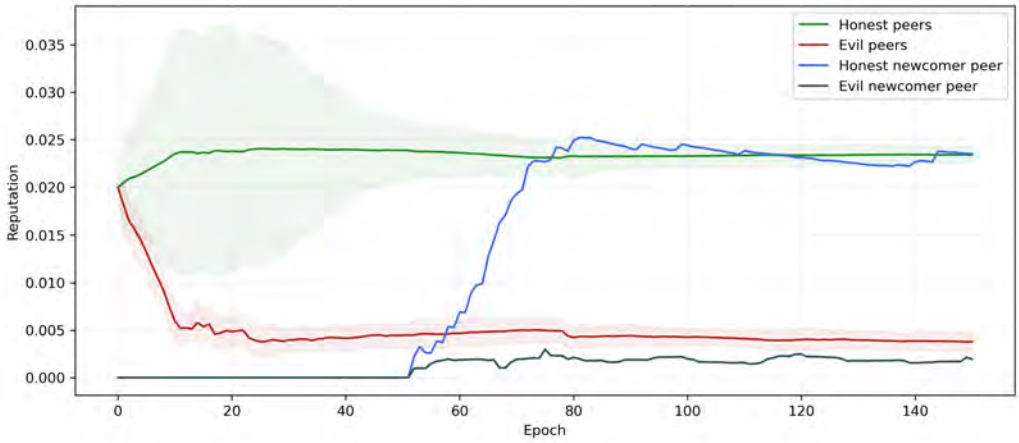


Figure 5.1: Evolution of the reputations of every kind of node in a scenario with 20% of malicious nodes.

properly modeled. The reputation of the honest newcomer gradually increases to the level of the other honest nodes, whereas the reputation of the malicious newcomer remains low, as the other malicious nodes in the network. Take into account that, during this phase, newcomers cannot be clients, just workers and forwardees. Hence, a malicious node does not gain any benefit from leaving and entering again the system with a new pseudonym (whitewashing).

The left chart of Figure 5.2 displays the rate of correct output as a function of the reputation of the clients, both magnitudes accumulated after 150 iterations. Honest nodes are depicted in green, malicious nodes in red, the honest newcomer in blue and the malicious newcomer in gray. Nearly all the outputs requested by clients with high reputation were correct, whereas on average less than 50% of the outputs requested by clients with low reputation were correct. This satisfied our expectations on the behavior of the co-utile framework. In fact, as it can be seen in the right chart of Figure 5.2, reputation was even more decisive in the last 50 iterations. In these last iterations, all computations requested by peers with high reputation (honest peers) were correct, whereas on average less than 20% of the outputs requested by clients with low reputation were correct. Hence, as soon the system stabilizes, the good peers *always* obtain correct outputs.

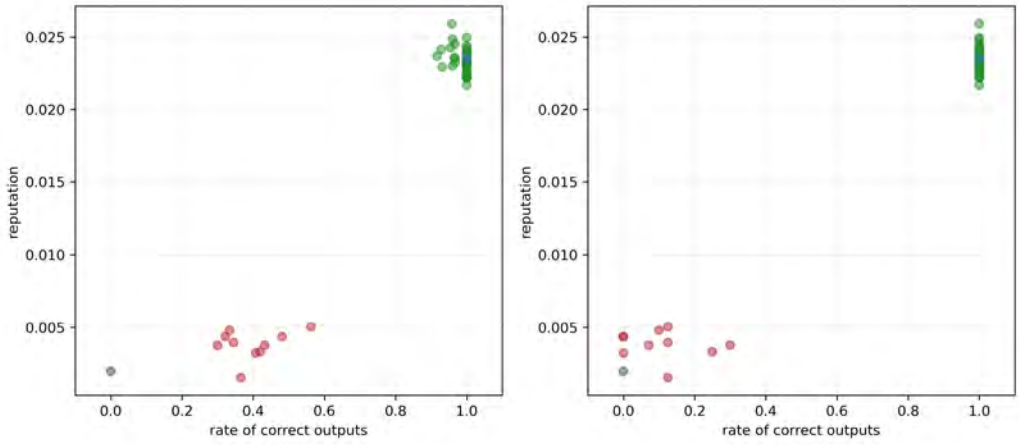


Figure 5.2: Rate of correct outputs as a function of the reputation of clients, with 20% of malicious nodes. On the left, in all 150 iterations; on the right, in the last 50.

5.7.2 Comparison with the baseline framework

In order to compare our co-utile framework with other similar options, we set up a *baseline* framework in which there was worker redundancy but *no* reputation management was used by peers to choose workers or forwardees, or to decide whether to act as a worker or a forwardee. We ran simulations for different proportions of malicious peers. In the co-utile framework, we gave an initial reputation $g = 1/n = 0.02$ to each client and we took $\delta = g * 0.75 = 0.015$.

Figure 5.3 shows the rate of correct outputs in the baseline and the co-utile frameworks as a function of the proportion of malicious peers, after 250 iterations. No matter the framework, the rate of correct outputs decreases as the proportion of malicious peers increases, which was to be expected. However, in the co-utile framework good clients obtained a much higher rate of correct outputs than bad clients. Furthermore, good clients in the co-utile framework obtained a higher rate of correct outputs than clients in the baseline framework, whereas bad clients in the co-utile framework obtained a *much* lower rate of correct outputs than clients in the baseline framework. Hence, reputation management in the co-utile framework is useful to discriminate between good and bad clients, which in fact encourages rational behavior: honest behavior turns out to be the best rational option for any peer that wants to become a client.

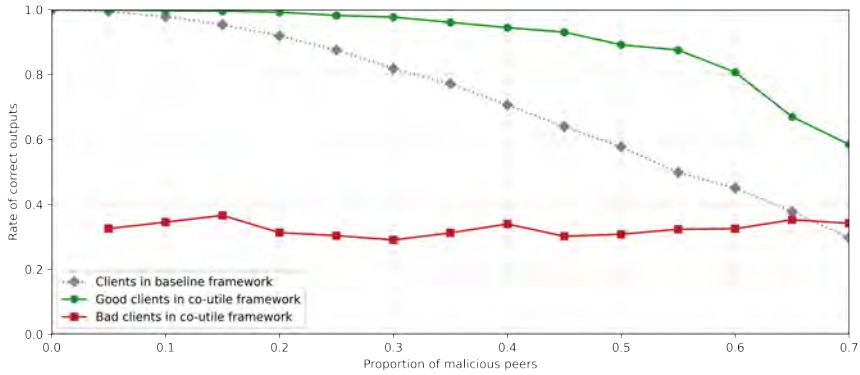


Figure 5.3: Rate of correct outputs in the baseline and the co-utile frameworks for different types of clients, as a function of the proportion of malicious peers.

5.8 Conclusions and future work

We have presented a peer-to-peer framework for multiparty computation that has the following innovative features: (i) it is general-purpose without making use of circuits, so that it can work for any computation expressed as ordinary high-level programming code, no matter its complexity, loops or recursions; (ii) unequivocal linkability of each party's inputs is prevented; (iii) if a majority of peers are rational (not necessarily semi-honest), collusion is unattractive and hence, unequivocal linkability of outputs is also prevented; (iv) the rational behavior of peers is to return correct outputs to the client parties.

Future research will be devoted to reducing the overhead caused by worker and accountability manager redundancy while preserving the current privacy and correctness guarantees. Further work will be performed on parameter tuning, and in particular on the value of the initial reputation to be assigned to newcomers.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Decentralized computing is an emerging paradigm that redistributes computational tasks and decision making across a network, offering benefits such as resilience, scalability, privacy, and reduced reliance on central authorities. The emergence of blockchain technology and other decentralized solutions, including peer-to-peer networks, distributed ledger technologies, and decentralized file systems, has fuelled the growth of the sector, providing new opportunities for collaboration and innovation. These developments empower individuals and organizations to collaborate and transact directly, bypassing traditional intermediaries and fostering a more open and transparent digital ecosystem.

Decentralized systems face several challenges due to the absence of a central coordinator. Achieving consensus among nodes, dealing with scalability issues, ensuring fault tolerance, maintaining security without a central authority, and establishing effective governance models are key areas that require attention and innovative solutions.

This thesis focuses on one of the major hurdles for decentralization to become widespread: how to ensure that all agents involved perform as expected. Nodes

that deliberately deviate may do so to attack the system or simply to take advantage of it without contributing.

Based on the notion of co-utility, we have designed several ethics-by-design frameworks to solve the conflict between some ethical values and security properties in three different scenarios: i) Federated learning; ii) Fully decentralized learning; and iii) Multi-party computation.

In our first contribution, presented in Chapter 3, we have presented co-utile protocols to improve privacy and security in federated learning while perfectly preserving the model accuracy. We use a decentralized reputation management scheme to incentivize peers to adhere to the prescribed protocols. Confidentiality of the peers' private data is guaranteed by the unlinkability of updates: when a peer generates an update, neither the model manager nor the other peers can identify the update generator. Security, *i.e.* protection against bad updates, is pursued in our approach via reputation. Whereas state-of-the-art security countermeasures do nothing to reduce the number of bad updates that are processed by the model manager, we address this issue in a way to achieve two beneficial effects: first, to decrease the overhead for the model manager and the peers related to processing, tracing and punishing bad updates; and, second, to make the (fewer) bad updates processed by the model manager more identifiable as outliers. The design of our protocols also renders whitewashing and Sybil attacks ineffective.

In Chapter 4 we presented a co-utile protocol for Fully Decentralized Machine Learning. The fact that all participating peers are at the same time model managers and workers facilitates using tit-for-tat and local reputations to achieve co-utility. In this way, co-utility occurs naturally, and complex incentives such as decentralized global reputations are not needed. Our proposed protocol consists of a double tit-for-tat. The first tit-for-tat is used to hide the identity of the peer that computed each update, in order to protect the privacy of peers. In the second tit-for-tat, peers receive updates relevant to their own model, but they receive them from other peers that did not compute those updates. Deviations from the protocol are punished, which increases ostracism for the offending peers. On the one hand, a peer that fails to reciprocate in a tit-for-tat is punished by the non-reciprocated peer. On the other hand, after the second tit-for-tat, bad or duplicate updates can be detected by the model-managing peers, who launch punishment subprotocols aimed at tracing the offending peers. This ensures the correctness of the learning process and incentivizes good and non-duplicate updates.

Finally, in Chapter 5 we proposed a peer-to-peer co-utile framework for multiparty computation that has the following innovative features: (i) it is general-purpose without making use of circuits, so that it can work for any computation expressed as ordinary high-level programming code, no matter its complexity, loops or recursions; (ii) unequivocal linkability of each party's inputs is prevented; (iii) if a majority of peers are rational (not necessarily semi-honest), collusion is unattractive and hence, unequivocal linkability of outputs is also prevented; (iv) the rational behavior of peers is to return correct outputs to the client parties.

6.2 Publications

The publications supporting the content of this thesis are stated below:

1. J. Domingo-Ferrer, A. Blanco-Justicia, J. Manjón and D. Sánchez, "Secure and privacy-preserving federated learning via co-utility", *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3988-4000, Mar 2022.

2. J. Manjón and J. Domingo-Ferrer, "Computación segura multiparte coútil para cálculo de funciones arbitrarias", in *XVII Reunión Española sobre Criptología y Seguridad de la Información - RECSI 2022*, Santander, Spain, Oct 2022.

3. J. Domingo-Ferrer and J. Manjón, "Circuit-Free General-Purpose Multi-Party Computation via Co-Utile Unlinkable Outsourcing", *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 539-550, Jan 2023.

4. J. Manjón, J. Domingo-Ferrer, D. Sánchez and A. Blanco-Justicia, "Secure, Accurate and Privacy-Aware Fully Decentralized Learning via Co-Utility", *Computer Communications*, vol. 207, pp. 1-18, Jul. 2023.

6.3 Future work

Regarding the contributions presented in this thesis, we foresee some open problems and extensions that will be addressed in the future.

An interesting avenue for future research in the federated learning scenario is to harden the proposed protocols so that they can filter out a greater proportion of bad updates in situations where a substantial share of the peers are malicious.

Concerning fully decentralized machine learning, we plan to fine-tune the protocol in order to further reduce its overhead and the proportion of lost good updates.

Regarding multiparty computation, we expect to be able to reduce the overhead caused by worker and accountability manager redundancy while preserving the current privacy and correctness guarantees.

Additionally, we plan to explore adaptations of the proposed protocols to decentralized computations different from machine learning and multiparty computation, such as decentralized anonymization.

Bibliography

[1] J. Domingo-Ferrer and D. Sánchez, eds., *Co-Utility - Theory and Applications*. Springer Cham, 2017.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282, 20–22 Apr 2017.

[3] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.

[4] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160–164, IEEE, 1982.

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM computer communication review*, vol. 31, no. 4, pp. 149–160, 2001.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 329–350, Springer, 2001.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the 2001 conference on*

Applications, technologies, architectures, and protocols for computer communications, pp. 161–172, 2001.

[8] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663, 1997.

[9] C. Buragohain, D. Agrawal, and S. Suri, "A game theoretic framework for incentives in p2p systems," in *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*, pp. 48–56, IEEE, 2003.

[10] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proceedings of the 5th ACM conference on Electronic commerce*, pp. 102–111, 2004.

[11] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma: A secure economic framework for peer-to-peer resource sharing," in *Workshop on Economics of Peer-to-peer Systems*, vol. 35, 2003.

[12] J. Crowcroft, R. Gibbens, F. Kelly, and S. Östring, "Modelling incentives for collaboration in mobile ad hoc networks," *Performance Evaluation*, vol. 57, no. 4, pp. 427–439, 2004.

[13] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Computing Surveys (CSUR)*, vol. 42, no. 1, pp. 1–31, 2009.

[14] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the 12th international conference on World Wide Web*, pp. 640–651, 2003.

[15] J. Domingo-Ferrer, S. Martínez, D. Sánchez, and J. Soria-Comas, "Co-utility: Self-enforcing protocols for the mutual benefit of participants," *Engineering Applications of Artificial Intelligence*, vol. 59, pp. 148–158, 2017.

[16] J. Domingo-Ferrer, O. Farràs, S. Martínez, D. Sánchez, and J. Soria-Comas, "Self-enforcing protocols via co-utile reputation management," *Information Sciences*, vol. 367-368, pp. 159–175, 2016.

[17] K. Leyton-Brown and Y. Shoham, "Essentials of game theory: A concise multidisciplinary introduction," *Synthesis lectures on artificial intelligence and machine learning*, vol. 2, no. 1, pp. 1–88, 2008.

[18] R. D. Luce and H. Raiffa, *Games and decisions: Introduction and critical survey.* Courier Corporation, 1989.

[19] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[20] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[21] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *International Conference on Machine Learning*, pp. 3478–3487, PMLR, 2019.

[22] M. Lotfi, G. J. Osório, M. S. Javadi, M. S. El Moursi, C. Monteiro, and J. P. Catalão, "A fully decentralized machine learning algorithm for optimal power flow with cooperative information exchange," *International Journal of Electrical Power & Energy Systems*, vol. 139, p. 107990, 2022.

[23] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[24] A. Blanco-Justicia, J. Domingo-Ferrer, S. Martínez, D. Sánchez, A. Flanagan, and K. E. Tan, "Achieving security and privacy in federated learning systems: Survey, research challenges and future directions," *Engineering Applications of Artificial Intelligence*, vol. 106, p. 104468, 2021.

[25] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1322–1333, 2015.

[26] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18, IEEE, 2017.

[27] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE symposium on security and privacy (SP)*, pp. 691–706, IEEE, 2019.

[28] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 603–618, 2017.

[29] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321, 2015.

[30] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the works of leslie lamport*, pp. 203–226, 2019.

[31] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[32] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, pp. 634–643, PMLR, 2019.

[33] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

[34] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–25, 2017.

[35] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*, pp. 5650–5659, PMLR, 2018.

[36] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "Sok: General purpose compilers for secure multi-party computation," in *2019 IEEE symposium on security and privacy (SP)*, pp. 1220–1237, IEEE, 2019.

[37] D. Evans, V. Kolesnikov, M. Rosulek, *et al.*, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.

[38] Y. Lindell, "Secure multiparty computation," *Communications of the ACM*, vol. 64, p. 86–96, dec 2020.

[39] J. Furukawa and Y. Lindell, "Two-thirds honest-majority mpc for malicious adversaries at almost the cost of semi-honest," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1557–1571, 2019.

[40] O. Goldreich, S. Micali, and A. Wigderson, *How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority*, p. 307–328. New York, NY, USA: Association for Computing Machinery, 2019.

[41] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, (New York, NY, USA), p. 1–10, Association for Computing Machinery, 1988.

[42] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 11–19, 1988.

[43] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Workshop on the theory and application of cryptographic techniques*, pp. 47–53, Springer, 1984.

[44] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: a comprehensive introduction.* Princeton University Press, 2016.

[45] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security*, vol. 1, p. 66–92, nov 1998.

[46] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*, pp. 223–238, Springer, 1999.

[47] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.

[48] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "{BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning," in *2020 USENIX annual technical conference (USENIX ATC 20)*, pp. 493–506, 2020.

[49] E. Barker, "Nist special publication 800-57 part 1, revision 5," *NIST, Tech. Rep*, 2020.

[50] I. Colin, A. Bellet, J. Salmon, and S. Clémençon, "Gossip dual averaging for decentralized optimization of pairwise functions," in *International Conference on Machine Learning*, pp. 1388–1396, PMLR, 2016.

[51] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "d^2: Decentralized training over decentralized data," in *International Conference on Machine Learning*, pp. 4848–4856, PMLR, 2018.

[52] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[53] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web caching with consistent hashing," *Computer Networks*, vol. 31, no. 11-16, pp. 1203–1213, 1999.

[54] D. G. Thaler and C. V. Ravishankar, "Using name-based mappings to increase hit rates," *IEEE/ACM Transactions on networking*, vol. 6, no. 1, pp. 1–14, 1998.

[55] J. Domingo-Ferrer, A. Blanco-Justicia, J. Manjón, and D. Sánchez, "Secure and privacy-preserving federated learning via co-utility," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3988–4000, 2021.

[56] A. Küpçü, "Incentivized outsourced computation resistant to malicious contractors," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 6, pp. 633–649, 2015.

[57] D. Harz and M. Boman, "The scalability of trustless trust," in *International Conference on Financial Cryptography and Data Security*, pp. 279–293, Springer, 2018.

[58] M. Nabi, S. Avizheh, M. V. Kumaramangalam, and R. Safavi-Naini, "Game-theoretic analysis of an incentivized verifiable computation system," in *International Conference on Financial Cryptography and Data Security*, pp. 50–66, Springer, 2019.

[59] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Cryptography from anonymity," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 239–248, IEEE, 2006.



UNIVERSITAT
ROVIRA i VIRGILI