Cryptographic Foundations For Control And Optimization: Making Cloud-Based And Networked Decisions On Encrypted Data

Abstract

Advances in communication technologies and computational power have determined a technological shift in the data paradigm. The resulting architecture requires sensors to send local data to the cloud for global processing such as estimation, control, decision and learning, leading to both performance improvement and privacy concerns. This thesis explores the emerging field of private control for Internet of Things, where it bridges dynamical systems and computations on encrypted data, using applied cryptography and information-theoretic tools.Our research contributions are privacy-preserving interactive protocols for cloud-outsourced decisions and data processing, as well as for aggregation over networks in multi-agent systems, both of which are essential in control theory and machine learning. In these settings, we guarantee privacy of the data providers' local inputs over multiple time steps, as well as privacy of the cloud service provider's proprietary information. Specifically, we focus on (i) private solutions to cloud-based constrained quadratic optimization problems from distributed private data; (ii) oblivious distributed weighted sum aggregation; (iii) linear and nonlinear cloud-based control on encrypted data; (iv) private evaluation of cloud-outsourced data-driven control policies with sparsity and low-complexity requirements. In these scenarios, we require computational privacy and stipulate that each participant is allowed to learn nothing more than its own result of the computation. Our protocols employ homomorphic encryption schemes and secure multi-party computation tools with the purpose of performing computations directly on encrypted data, such that leakage of private information at the computing entity is minimized. To this end, we co-design solutions with respect to both control performance and privacy specifications, and we streamline their implementation by exploiting the rich structure of the underlying private data.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

First Advisor

George J. Pappas

Keywords

Control theory, Cryptography, Optimization, Privacy

Subject Categories

Computer Sciences | Electrical and Electronics

CRYPTOGRAPHIC FOUNDATIONS FOR CONTROL AND OPTIMIZATION: MAKING CLOUD-BASED AND NETWORKED DECISIONS ON ENCRYPTED DATA

Andreea B. Alexandru

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2021

Acknowledgment

The past years were a true rollercoaster. I am indebted to the people that helped me navigate both the highs and the lows, and shaped my experience.

I want to thank my advisor George Pappas for encouraging me to pursue a topic at the interface of two very different areas, for all the appreciation and support, and for all the TV series recommendations.

My thesis committee members were also my mentors and I am grateful for their help along the way. I want to thank Manfred Morari for his perpetual honesty and insights, I enjoyed all our discussions. I remember that the day before I visited UPenn as a prospective PhD student, I interviewed with Manfred for a scholarship at ETHZ, and felt torn between the two opportunities. It was an unexpected and great surprise that things aligned so well and I got the best of both worlds. I met Tal Rabin less than one year ago, and yet her impact is already tremendous. I want to thank her for all the help, advice and availability for all kind of discussions. Sebastian Angel's interview presentation was the first talk that mentioned homomorphic encryption that I had seen at UPenn and it came at a moment when I felt very isolated with my research. I enjoyed his seminars and I am grateful for involving me in the security group and help me feel more connected with the security side of the department.

I am grateful to all my collaborators for the discussions and feedback, which helped me write papers of better quality. Starting from my first year, I want to thank Sérgio Pequito, Ali Jadbabaie, George Pappas, Konstantinos Gatsis, Manfred Morari, Yasser Shoukri, Paulo Tabuada, Sanjit Seshia, Moritz Schulze Darup, Anastastios Tsiamis and Daniel Quevedo. I also want to thank the Duality Technologies team for a very impactful internship, enabled by their openness towards me, although I was coming from a different field.

For useful feedback for my dissertation proposal and dissertation presentation and pressure to remove content (which I strongly resisted but of course lead to a better result), I would like to thank Matei, George, Tasos, Aritra, Edo, Heejin, Marcella, Alëna and Achin.

The Security Reading Group was a pleasant and useful way to spend Thursdays' lunches and I want to thank to Marcella and Daniel for leading it. I would also like to thank the administrative staff in the ESE office and GRASP lab for their support and goodwill.

All work and no play would have made Andreea a dull girl, but thanks to the brunch group: Achin, Tasos, Alëna, Heejin, Kuk, Yash, my GRASP lab corner: Christine and Diego, and new year's eve group: Paul, Tache, Tudi, Adi, Bogdan, Ana, Andrei and Raluca, the dullness part did not happen. I am eternally grateful for all the brunches, lunches, dinners, coffee breaks, walks, parties, trips, and lately zoom calls, that helped me get through it all. There are many, many other people left to thank for making my PhD memories beautiful.

I am grateful to my (growing) family for being supportive and understanding of the distance, for visiting, for going together on a different adventure every summer and for finally adopting a cat: my parents Aura and Cristi, my sister Alice, Marius, Diana, Magda, my grandfather.

Thank you Matei for e v e r y t h i n g.

Last but not least, thanks to you, the reader!

ABSTRACT

CRYPTOGRAPHIC FOUNDATIONS FOR CONTROL AND OPTIMIZATION: MAKING CLOUD-BASED AND NETWORKED DECISIONS ON ENCRYPTED DATA

Andreea B. Alexandru

George J. Pappas

Advances in communication technologies and computational power have determined a technological shift in the data paradigm. The resulting architecture requires sensors to send local data to the cloud for global processing such as estimation, control, decision and learning, leading to both performance improvement and privacy concerns. This thesis explores the emerging field of private control for Internet of Things, where it bridges dynamical systems and computations on encrypted data, using applied cryptography and information-theoretic tools. Our research contributions are privacy-preserving interactive protocols for cloud-outsourced decisions and data processing, as well as for aggregation over networks in multi-agent systems, both of which are essential in control theory and machine learning. In these settings, we guarantee privacy of the data providers' local inputs over multiple time steps, as well as privacy of the cloud service provider's proprietary information. Specifically, we focus on (i) private solutions to cloud-based constrained quadratic optimization problems from distributed private data; (ii) oblivious distributed weighted sum aggregation; (iii) linear and nonlinear cloud-based control on encrypted data; (iv) private evaluation of cloud-outsourced data-driven control policies with sparsity and low-complexity requirements. In these scenarios, we require computational privacy and stipulate that each participant is allowed to learn nothing more than its own result of the computation. Our protocols employ homomorphic encryption schemes and secure multi-party computation tools with the purpose of performing computations directly on encrypted data, such that leakage of private information at the computing entity is minimized. To this end, we co-design solutions with respect to both *control performance* and *privacy specifications*, and we streamline their implementation by exploiting the rich structure of the underlying private data.

Contents

Chapter 1

Introduction

1.1 Motivation

Following the Internet of Things (IoT) revolution, billions of interconnected sensors and actuators are deployed in every aspect of our daily lives: in intelligent infrastructure and transportation systems, medical monitoring devices, industrial manufacturing and robotics. This increase in the number of available sensors and generated data has led to an increase in the required computational capacity for decision-making and global processing that involves estimation, control and learning.

One solution is to outsource the computations from low-power platforms to powerful remote cloud servers that offer on-demand storage, aggregation and processing capabilities. Apart from storing and processing databases, cloud computing has been employed for machine learning applications in e.g., healthcare monitoring and social networks, smart grid control and autonomous vehicle control, and integration with the IoT paradigm [46, 189]. Alternatively, distributed architectures where agents jointly carry out computations over a communication graph are preferred at the edge level [157].

This emerging paradigm makes privacy a fundamental issue in both cloud-based and distributed applications, due to the sensitive nature of the collected data, e.g., individual location, healthcare information or energy consumption. The privacy concerns of outsourced computing include tampering with the stored data and interfering with the computation, which can be maliciously or unintentionally exploited by the cloud service or other tenants of the service. Recent examples of data leakage and abuse by cloud servers have drawn attention to the risks of storing data in the clear and urged more measures against untrustworthy participants [17, 119, 191]. Moreover, concerns about intellectual property may prevent competitors from pooling their data or models together, and thus hinder the development of better solutions for customers. Under these circumstances, securing communication channels between the participants is necessary but not sufficient to protect the users' data; we also need to address corrupt participants and secure the data.

As seen from the surge in smart infrastructure and widespread robotics, these new trends are also arising in control systems and require transitioning to cloud-based and networked algorithms. Furthermore, the paradigm of data-driven decision, which lies at the core of adaptive control solutions, is reimagined and demanded for in the era where the collection, transmission and processing of data is no longer a technological bottleneck.

Since networked control systems are a fundamental layer in critical infrastructure such as electric power, transportation, and water distribution networks, cyberattacks have been common in this context too. Well-known examples are the malwares Stuxnet, Duqu, Industroyer, or Triton [61], as well as inference attacks using smart meters as surveillance devices [109, 168]. Furthermore, stealth, false-data injection, replay, covert, and distributed denial-of-service (DDoS) attacks were identified [181]. Cyberattacks on control systems can be highly critical. Unlike attacks on classical IT systems, attacks on control systems may influence *physical* processes through digital manipulations [207] and cause substantial damage.

In conclusion, there is a major industrial need for both cloud-based and distributed algorithms which can perform estimation, control and optimization tasks while maintaining the privacy of the user data.

1.2 Overview of the thesis

An overarching theme in this thesis is achieving privacy for *structured data*, i.e., data which is connected by models or optimality conditions. Control and optimization problems, as well as training and inference over neural networks, involve time-series data in instances of dynamical systems. Knowledge regarding the evolution and control of such systems can be advantageous in providing security, however, the structure of dynamical systems raises challenges, such as preserving privacy over multiple time steps and iterations, or efficiently deploying security measures for constrained devices with fast response times. This is unlike the data in one-shot computations in static databases, for which many privacy techniques have been already developed.

General problem statement. Given a dynamical system which generates privacy-sensitive data, we design algorithms for cloud-based or networked decision-making which efficiently meet privacy requirements. Specifically, we stipulate that each participating party is only allowed to learn its own result of the computation and no other partial information.

In this thesis, we design multi-disciplinary solutions which ensure the privacy of structured data arising from critical infrastructure and user-related sensitive queries, weaving together knowledge of optimization, systems and control engineering, with modern cryptographic tools such as homomorphic encryption and secret sharing. The goal is to ensure that everything other than prior knowledge about the structure of the data remains private. The threat model assumed in this thesis is of passive adversaries that do not tamper with the data, but try to infer information from it.

Optimization and control foundations. In the following, I will give an overview of the problems investigated in the thesis, emphasizing the targeted foundational concepts. In Chapter 2, we provide the reader with a basis on the cryptographic notions we will use in this thesis. Chapter 3 is devoted to providing protocols that securely evaluate the most common optimization problems arising in control systems synthesis (and other decision-making algorithms from machine learning): quadratic optimization problems with or without constraints, with or without regularization. These solutions provide foundations for solving more general optimization problems in a private manner. In Chapter 4, we review and explain the concept of private sum aggregation, where an actor obliviously aggregates the individual contributions of other actors, and we expand it to the more complex, but widespread scenario of private weighted sum aggregation, with proprietary weights. Securely evaluating such affine laws is essential in distributed and federated computation, such as consensus, control, averaging and learning. From Chapter 5 onward, we focus on control-theoretic fundamental algorithms. We start with linear control policies, which are one of the pillars of control systems theory. Despite their pervasiveness and elementary formulation, under privacy requirements on the system's signals and control parameters, they bring significant challenges. To illustrate secure solutions, we consider a linear estimation problem, a linear optimal regulator and structured distributed linear control. The latter is based on the private weighted aggregation described in Chapter 4. We further consider a nonlinear control policy in Chapter 6, in particular, an optimal receding horizon controller. The private solution builds on the quadratic programs from Chapter 3, as well as on the solution from Chapter 5. In Chapter 7, we explore data-driven control problems, where the decisions have to be privately computed from input-output data, without prior knowledge of the system parameters. This framework comes with extra complexity, compared to a system with known model. The solutions presented in this chapter build on all the previous chapters, in terms of private optimization algorithms, insights on streamlining private computations and reducing complexity from aggregation and linear control, and connections to the nonlinear control. Finally, in Chapter 8, we complement the research contributions in this thesis with a discussion on future research directions targeting malicious adversaries that do not follow the prescribed protocols, secure preprocessing for end-to-end private control procedures, and more specialized and complex private algorithms.

1.3 Related work

Secure control for networked systems has been intensively studied in the literature during the last decade. Comprehensive surveys can be found in [53, 67, 137, 181, 207]. Most existing work focuses on the integrity and availability of networked control schemes using various defense mechanisms. For example, control-related concepts such as detectability and identifiability of deception attacks are investigated in [181] and game-theoretic approaches to deal with DDoS attacks are considered in [110, 150]. Nevertheless, interdisciplinary solutions are required to secure control systems.

The emerging field of encrypted control primarily aims for confidentiality of sensitive system states, control actions, controller parameters, or model data in the entire control loop. More generally, an encrypted controller can be defined as a networked control scheme that simultaneously ensures control performance and privacy of the client system(s), as well as privacy for the service provider's proprietary information, through specialized cryptographic tools. In the framework of networked control, attacks compromising confidentiality such as eavesdropping might seem less critical since they do not immediately cause physical misbehavior. However, "passive" spying often precedes "active" attacks compromising data integrity and availability [67]. Abstractly speaking, encrypted control is realized by modifying conventional control schemes such that they are capable of computing encrypted inputs based on encrypted states (or encrypted controller parameters) without intermediate decryptions by the controller. Encrypted control goes beyond secure communication channels by providing security against curious cloud providers or neighboring agents that, during controller evaluations, would have access to unsecured data. This is the key difference between encrypted control and existing secure control schemes focusing on confidentiality [145, 210, 211]. Meeting these privacy demands under real-time restrictions is non-trivial and requires a co-design of controllers and suitable cryptosystems. We further survey encrypted control in [200].

Among the tools used in the literature to protect the privacy of the data computed upon, we mention homomorphic encryption and secure multi-party computation [119], and differential privacy [84], that we survey below.

Secure Multi-Party Computation (SMPC) encompasses a range of cryptographic techniques that facilitate joint computation over secret data distributed between multiple parties, which can be both clients and servers. The goal of SMPC is that each party is only allowed to learn its own result of the computation, and no intermediary results such as inputs or outputs of other parties or other partial information. The concept of SMPC originates from [230], where a secure solution to the millionaire's problem was proposed. Surveys on SMPC can be found in [74]. SMPC involves communication between parties and can include individual or hybrid approaches between techniques such as secret sharing [29, 184, 201], oblivious transfer [172, 186], garbled circuits [30, 106, 230], (threshold) homomorphic encryption [49, 75, 171, 177], etc.

Homomorphic Encryption (HE), introduced in [190] as *privacy homomorphism*, refers to a secure computation technique that allows evaluating computations on encrypted data and produces an encrypted result. HE is best suited when there is a client-server scenario with an untrusted server: the client simply has to encrypt its data and send it to the server, which performs the computations on the encrypted data and returns the encrypted result. The first HE schemes were partial, meaning that they either allowed the evaluation of additions or multiplications, but not both. Then, somewhat homomorphic schemes were developed, which allowed a limited number of both operations. One of the bottlenecks for obtaining an unlimited number of operations was the accumulation of noise introduced by evaluating operations, which could eventually prevent the correct decryption. The first fully homomorphic encryption scheme that allowed the evaluation of both additions and multiplications on encrypted data was developed in [102], where, starting from a somewhat homomorphic encryption scheme, a bootstrapping operation was introduced. Bootstrapping allows to obliviously evaluate the scheme's decryption circuit and reduces the ciphertext noise. Other fully homomorphic encryption schemes include [50, 51, 62, 91, 103]. For a thorough history and description of HE, see the survey [159]. For multiple users, the concept of functional privacy is required, which can be attained by functional encryption [43], which was developed only for limited functionalities. Privacy solutions based on HE were proposed for genome matching, national security and critical infrastructure, healthcare databases, machine learning applications and control systems [19, 22, 188], etc. Of particular interest to us are the the works in control applications with HE, see [6, 11, 92, 108, 134, 170, 198], to name a few. Furthermore, there has been a soaring interest in homomorphically encrypted machine learning applications, from statistical analysis and data mining [22] to deep learning [188].

The privacy definition for SMPC stipulates that the privacy of the inputs and intermediary results is ensured, but the output, which is a function of the inputs of all parties, is revealed. For applications such as smart meter aggregation [2], social media activity [194], health records [81], deep learning [1], and where input-output privacy is valued over output accuracy, the SMPC privacy definition is not enough [238] and needs to be augmented by guarantees of differential privacy.

Differential Privacy (DP) refers to methods of concealing the information leakage from the result of the computation, even when having access to auxiliary information [84, 86]. Intuitively, the contribution of the input of each individual to the output should be hidden from those who access the computation results. To achieve this, carefully chosen noise is added to each entry such that the statistical properties of the database are preserved [87], which introduces a trade-off between utility and privacy. When applied in a distributed system [85, 212], DP provides a problematic solution for smaller number of entries in the dataset, as all parties would add noise that would completely drown the result of the computation. Several works combine SMPC with DP in order to achieve both computation privacy and output privacy, for instance [58, 185, 187, 202, 213]. An intuitive comparison between privacy-preserving centralized and decentralized computation approaches, with different privacy goals and utilities is given in Table 1.1.

Model	Utility	Privacy	Who Holds the Data
Fully Homomorphic (or Functional) Encryption	any desired query	everything (except possibly the result of the query)	original users or delegates/untrusted server holds encrypted data
Secure Multi-Party Computation	any desired query	everything other than the result of the query	original users or delegates
Centralized Differential Privacy	statistical analysis of dataset	individual-specific information	trusted curator
Multi-Party Differential Privacy	statistical analysis of dataset	individual-specific information	original users or delegates

Table 1.1: Comparison between centralized and multi-party privacy-preserving approaches [213].

1.4 Research contributions

Dynamical data challenges. Cryptographic solutions were developed for the most part for static data, such as databases, or independent data. However, dynamical systems are iterative processes that generate structured and dependent data. Moreover, output data at one iteration/time step will often be an input to the computation at the next one. Hence, special attention is needed when using cryptographic techniques in solving optimization problems and implementing control schemes. For example, values encrypted with homomorphic encryption schemes will require ciphertext refreshing or bootstrapping if the multiplicative depth of the algorithm exceeds the multiplicative depth of the scheme; when using garbled circuits, a different circuit has to be generated for different iterations/time steps of the same algorithm; the controlled noise added for differential privacy at each iteration/time step will accumulate and drown the result etc. Furthermore, privacy is guaranteed as long as the keys and randomness are never reused, but freshly generated for each time step; then, the (possibly offline) phase in which uncorrelated randomness is generated, (independent from the actual inputs), has to be repeated for a continuously running process. In this thesis, we design solutions with all these issues in mind.

The research contributions in this thesis concern cloud-based decision and data processing solutions that guarantee privacy of the users' local data over multiple time steps, as well as privacy of the service provider's proprietary data; and private computation and aggregation over networks in multi-agent systems. In all these scenarios, we stipulate that each participating party is only allowed to learn its own result of the computation and no other partial information. The idea is to leverage cryptographic tools which allow the evaluation of computations on encrypted data, ensuring in this way that the privacy of the encrypted data is never violated. This approach achieves strong privacy guarantees, which are highly desirable in a wide range of applications, from critical infrastructure to now ubiquitous Internet of Things devices. Such a privacy goal can be accomplished using SMPC, which encompasses a range of cryptographic techniques that facilitate joint computation over secret data distributed between multiple parties. In this thesis we employ, among others, homomorphic encryption, secret sharing and oblivious transfer.

Private cloud-based optimization and control

A fundamental control scenario for large systems involves a client outsourcing to a cloud service the computation of the control decisions, using model predictive control with control input constraints. Without privacy requirements, the cloud controller could solve the problem locally through an optimization algorithm. With privacy requirements on the states and control actions, and possibly on the system model, the cloud controller should not infer anything about this private data during and after the computations. This fits in the paradigm of encrypted computations. A popular solution is to have the cloud perform the computations on the client's encrypted data, without the need of decryption. This can be achieved via additively homomorphic encryption (AHE), which allows the evaluation of affine transformations on encrypted data. However, there are two subproblems that cannot be privately solved using solely AHE: constrained optimization and iterated multiplications. Based on the solutions to these two problems, which are described below, we show how to achieve cloud-based encrypted model predictive control with private input constraints in Chapter 6 (following the work presented in [9, 11]).

Privately solving constrained optimization problems. More generally, consider an optimization problem based on private costs and constraints from data providers, that needs to be solved at a cloud service and the solution given to a requesting party. Satisfying the constraints requires iteratively projecting the private optimization variables on the feasible space. The first challenge is that the cloud is unable to perform this nonlinear projection operation using only the native affine operations allowed by AHE. Another challenge is to *only* provide the solution to the optimization problem, and nothing else, despite publicly known optimality conditions that connect the private data and can reveal private auxiliary information. We solve both challenges using blinded communication and oblivious transfer between the cloud and the requesting party, which allow the evaluation of the projection operation on the homomorphically encrypted data of the providers in Chapter 3. This guarantees no information is leaked, even under collusions with data providers [10, 13].

Estimation and control with encrypted model and encrypted signals. In most of the literature on encrypted control, the dynamics model is considered public. In Chapter 5 (based on the work in [6]), we propose a protocol for linear state estimation and linear control of a dynamical system, while enabling privacy for both its model and signals. This introduces the challenge of computing fast multiplications between encrypted values, with little communication, and of continuing the encrypted computations over multiple time steps. We require that the majority of the work for estimation and control be delegated to the cloud controller, while the system, comprised of individual subsystems, and the actuator do minimal work. We exploited the dynamics of the data by leveraging labeled homomorphic encryption, which associates a label to each message, and uses secret shares based on the labels and AHE to enable multiplication between two encrypted messages. We used the time steps associated to the measurements as labels so that there is a natural synchronization between the subsystems and the actuator. We also proposed an extension to obtain the online encrypted evaluation of higher order polynomials, at the cost of offline communication and storage. Thus, we were able to privately compute the estimation over measurements from multiple subsystems, then privately compute the optimal control action, all in real-time.

Private data-driven Control as a Service. Control as a Service is becoming a reality, particularly in the case of building automation and smart grid management. Oftentimes, it is required to control the client's system without assuming knowledge of its model. Consequently, large quantities of input-output data collected from the client are uploaded to a cloud server. The goal is to privately outsource data-driven decision and control to cloud services. We optimize the control algorithm used on this privacy-sensitive data for both efficiency and privacy. The idea is to focus on the encryption-aware co-design of a data-driven control algorithm, so that the privacy of the client's uploaded data, desired reference and control actions are maintained. We achieve privacy by using a leveled fully homomorphic encryption scheme to enable the cloud to perform complex computations on the client's encrypted data. We achieve efficiency by manipulating the tasks required by the control algorithm, including matrix inversion, such that they only involve a cheap low-depth arithmetic circuit, as well as by exploiting parallelization and ciphertext packing [15]. We thoroughly redesign arithmetic matrix operations and introduce redundancy inside the ciphertexts (copies of elements) to enable low-depth representation, and, perhaps counter-intuitively, to require less total memory [14].

In the same context, we also investigate a sparse data predictive control problem, run at a cloud service to control a system with unknown model, using ℓ_1-regularization to limit the behavior complexity, which can be written as a *Lasso* problem [16]. The input-output data collected for the system is privacy-sensitive, hence, we design a privacy-preserving solution using homomorphically encrypted data. The main challenges are the non-smoothness of the ℓ_1-norm, which is difficult to evaluate on encrypted data, as well as the iterative nature of the Lasso problem. We use a distributed ADMM formulation that enables us to exchange substantial local computation for little communication between multiple servers. We first give an encrypted multi-party protocol for solving the distributed Lasso problem, by approximating the non-smooth part with a Chebyshev polynomial, evaluating it on encrypted data, and using a more cost effective distributed bootstrapping operation in Chapter 3. For the example of data-predictive control, we prefer a non-homogeneous splitting of the data for better convergence. We give an encrypted multi-party protocol for this non-homogeneous splitting of the Lasso problem to a non-homogeneous set of servers: one powerful server and a few less powerful devices, added for security reasons in Chapter 7.

Private distributed control and inference

The recent technological advances in communication speed and deployment of millions of devices have fostered the adoption of distributed computing frameworks. In turn, such frameworks require aggregating services in order to utilize the data collected for specific causes. Even as a step in distributed algorithms, aggregation shifts from a decentralized nature that inherently guarantees more privacy, to a centralized approach that poses severe privacy challenges.

The applications of this line of work range from smart grid control and autonomous vehicle coordination to federated learning and graph neural network inference. The challenge is to enable the agents, which act as aggregators, to obliviously aggregate the *weighted* contributions from their neighbors, with privacy constraints on the local data and proprietary weights/model, while minimizing communication.

Private weighted sum aggregation. Depending on the application, different parties could know the associated weights: the agents, the aggregator or neither–a system operator generates offline proprietary weights and wants to keep them hidden from all participants. We exploit the information structure in the system to propose solutions that achieve privacy for the three cases in Chapter 4. In the more complex latter case, we propose using two compatible layers of encryption, one to protect the weights from the agents and the other to protect the agents' individual contributions from the aggregator. In [12], we used one layer of encryption to be an AHE scheme, and the other to be random masks that sum up to zero, that can be either precomputed and stored locally or computed online in a fully distributed manner. The appeal of this solution is its simplicity and relatively small sizes of ciphertexts. For small to medium weight matrices' dimensions, we can batch multiple values in a single ciphertext and redesign the computations accordingly, achieving up to 80% improvement in computation time and communication [8]. Our second solution [7] embeds a ciphertext in the error term of a learning with errors sample. This enables specific encoding and batching properties that can reduce the number of operations and of ciphertexts even further. This solution has fewer rounds of communication and allows a larger range of functionalities to be performed locally at the agents, but implies larger ciphertext sizes, and is more suitable for large weight matrices' dimensions and more complex distributed aggregation schemes.

Chapter 2

Cryptographic Preliminaries

This thesis assumes familiarity with control-theoretic and optimization-theoretic notions and aims to introduce the reader to cryptographic approaches for securing control and optimization applications in a networked scenario. Therefore, in this chapter, we describe and provide formal definitions for the cryptographic notions that will be used subsequently in the thesis, as they will be required in all chapters. On the other hand, we will introduce as needed the concepts behind the optimization and control algorithms we use in this thesis in the subsequent chapters.

2.1 Adversarial model

In this thesis we are concerned with adversaries that corrupt parties in order to steal their private information, but do not intend to disrupt the service offered. Common examples for this model are database breaches or third-party access, where the adversary plans to sell the proprietary and/or private data obtained. This model is called *semi-honest* or *honest-but-curious* and is defined in the following.

Definition 2.1.1. (Semi-honest model) A party is **semi-honest** if it does not deviate from the steps of the protocol, but may store the transcript of the messages exchanged and process the data received in order to learn more information than stipulated by the protocol.

This model also holds when considering eavesdroppers on the communication channels.

The semi-honest model differs from a model that only considers eavesdroppers as adversaries by the fact that, apart from the untrusted channels, the parties that perform the computations are also not trusted. *Malicious* or *active adversaries*–that diverge from the protocols or tamper with the messages–are not considered in this thesis but are crucial future research avenues for the author.

2.2 Security definitions

Definition 2.2.1. A function $\eta : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is **negligible** if for every positive $c \in \mathbb{R}_{>0}$, there exists $n_c \in \mathbb{Z}_{\geq 1}$ such that for all integers $n \geq n_c$, we have $|\eta(n)| \leq n^{-c}$.

In what follows, we will use $\eta(\cdot)$ to represent a negligible function and $\{0,1\}^*$ to define a sequence of bits of unspecified length. An *ensemble* $X = \{X_n\}_{n\in\mathbb{N}}$ is a sequence of random variables ranging over strings of bits of length polynomial in n denoted by $\mathrm{poly}(n)$.

Definition 2.2.2. (Statistical indistinguishability [104, Ch. 3]) The ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are **statistically indistinguishable**, denoted $\overset{s}{\equiv}$, if for all sufficiently large n, the following holds:

$$\frac{1}{2} \sum_{\alpha \in \mathrm{Supp}(X_n) \cup \mathrm{Supp}(Y_n)} \left| \Pr[X_n = \alpha] - \Pr[Y_n = \alpha] \right| < \eta(n),$$

where the quantity on the left is called the **statistical distance** between the two ensembles.

Computational indistinguishability is weaker than the statistical version, as follows:

Definition 2.2.3. (Computational Indistinguishability [104, Ch. 3]) The ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are **computationally indistinguishable**, denoted $\overset{c}{\equiv}$, if for **every probabilistic polynomial-time** algorithm $D : \{0,1\}^* \to \{0,1\}$, called the distinguisher, and all sufficiently large n, the following holds:

$$\left| \Pr_{x \leftarrow X_n}[D(x) = 1] - \Pr_{y \leftarrow Y_n}[D(y) = 1] \right| < \eta(n).$$

The previous definition is relevant when defining the privacy goals to be guaranteed by a protocol: *two-party privacy* of the sensitive data of the parties. The intuition is that a protocol privately computes a functionality if nothing is learned after its execution, i.e., if all information obtained by a computationally-bounded party after the execution of the protocol (while also keeping a record of the intermediate computations) can be obtained only from the inputs and outputs available to that party.

In the following definitions, we require the notion of *view*, which captures the information held by a party during an execution of the protocol. From this information, the respective party can try to infer more information than stipulated by the protocol.

Definition 2.2.4. (View of a party [105, Ch. 7]) Let Π be a protocol for computing an arbitrary functionality f on an input $\bar{x}$. The **view** of a party P during an execution of Π on the input $\bar{x}$, denoted $V^{\Pi}(\bar{x})$, is $(x, coins, m_1, \ldots, m_t)$, where x is the input of the party P, *coins* represents the outcome of party P's internal coin tosses, and m_j represents the j-th message it has received.

In this manuscript, we deal with deterministic functionalities, hence, we will present the simplified definitions of privacy in the semi-honest model for deterministic functionalities. For the more general case of probabilistic functionalities, as well as for a description of privacy definitions in the semi-honest and malicious models, we direct the reader to the tutorial [152].

Definition 2.2.5. (Two-party privacy w.r.t. semi-honest behavior [105, Ch. 7]) Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a functionality, and $f_1(x_1, x_2)$, $f_2(x_1, x_2)$ denote the first and second components of $f(x_1, x_2)$, for any inputs $x_1, x_2 \in \{0,1\}^*$. Let Π be a two-party protocol for computing f. Denote the view of the i-th party ($i = 1, 2$) as $V_i^{\Pi}(x_1, x_2)$. For a deterministic functionality f, we say that Π **privately computes** f if there exist probabilistic polynomial-time algorithms, called **simulators**, denoted S_i, such that:

$$\{S_i(x_i, f_i(x_1, x_2))\}_{x_{1,2} \in \{0,1\}^*} \overset{c}{\equiv} \{V_i^{\Pi}(x_1, x_2)\}_{x_{1,2} \in \{0,1\}^*}.$$

This definition assumes the correctness of the protocol, i.e., the probability that the output of the parties is not equal to the result of the functionality applied to the inputs is negligible. For semi-honest adversaries, this requirement is easy to check.

The next definition concerns security in the multi-party computation setup considering coalitions of a number of parties. Essentially, it extends Definition 2.2.5 and states that a multi-party protocol privately computes the functionality it runs if everything that a coalition of parties can obtain from participating in the protocol and keeping records of the intermediate computations can be obtained only from the inputs and outputs of these parties.

Definition 2.2.6. (Multi-party privacy w.r.t. semi-honest behavior [105, Ch. 7]) Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a n-ary functionality, where $f_i(x_1, \ldots, x_n)$ denotes the i-th element of $f(x_1, \ldots, x_n)$. Denote the inputs by $\bar{x} = (x_1, \ldots, x_n)$. For $I = \{i_1, \ldots, i_t\} \subset [n] = \{1, \ldots, n\}$, we let $f_I(\bar{x})$ denote the subsequence $f_{i_1}(\bar{x}), \ldots,\ f_{i_t}(\bar{x})$, which models a coalition of a number of parties. Let Π be a n-party protocol that computes f. Denote the view of the i-th party during an execution of Π on the inputs $\bar{x}$ by $V_i^{\Pi}(\bar{x})$, and denote the view of a coalition by $V_I^{\Pi}(\bar{x}) = (I, V_{i_1}^{\Pi}(\bar{x}), \ldots, V_{i_t}^{\Pi}(\bar{x}))$. For a deterministic functionality f, we say that Π **privately computes** f if there exist **simulators** S, such that, for every $I \subset [n]$, it holds that, for $\bar{x}_t = (x_{i_1}, \ldots, x_{i_t})$:

$$\{S(I, (\bar{x}_t), f_I(\bar{x}_t))\}_{\bar{x} \in (\{0,1\}^*)^n} \overset{c}{\equiv} \{V_I^{\Pi}(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^n}.$$

Remark 2.2.7. ([105, Ch. 7],[104, Ch. 4]) **Auxiliary inputs**, which are inputs that capture additional information available to each of the parties (e.g. local configurations, side-information), are implicit in Definition 2.2.5 and Definition 2.2.6.

We can also particularize Definition 2.2.5 to capture the setting of a client-server application, when we only require privacy of the client's input and output with respect to a semi-honest server.

Definition 2.2.8. (Client privacy w.r.t. semi-honest behavior of server) Let x_C be the private input of a client and x_S be the input of a server. The client wants the server to evaluate a functionality f and return the result $f(x_C, x_S)$. We will denote by $f_S(x_C, x_S)$ the corresponding result at the server. Let Π be a two-party protocol for computing f. Denote by $V_S^{\Pi}(x_C, x_S)$ the view of the server during an execution of Π on the inputs (x_C, x_S). For a deterministic functionality f, we say that Π **privately computes** f w.r.t. to the server if there exists a probabilistic polynomial-time algorithm, called **simulator** and denoted $\mathrm{S_S}$, such that, for any inputs x_C, x_S:

$$\{\mathrm{S_S}(x_S, f_S(x_C, x_S))\}_{x_C, x_S \in \{0,1\}^*} \overset{c}{\equiv} \{V_S^{\Pi}(x_C, x_S)\}_{x_C, x_S \in \{0,1\}^*}.$$

It is common for a server to be only employed for computing a result that is then sent to the client, and the server has no output formally, i.e., $f_S(x_C, x_S) = \emptyset$.

Finally, Definition 2.2.8 can be extended to the setup of a client and n servers, where at most $n-1$ servers are allowed to collude (Definition 3.3.1).

For encryption schemes and ciphers, we look at two definitions which tell us how secure the resulting ciphertext or code word is: *perfect secrecy* or *information-theoretic security* and *semantic security*.

Definition 2.2.9. (Perfect secrecy [133, Ch. 2]) An encryption scheme $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ over a message space $\mathcal{M}$ is **perfectly secret** if for every probability distribution over $\mathcal{M}$ and every ciphertext $c \in \mathcal{C}$ for which $\Pr[C = c] > 0$:

$$\Pr[M = m | C = c] = \Pr[M = m] \quad \text{or, equivalently,} \;\; \Pr[C = c | M = m] = \Pr[C = c].$$

Definition 2.2.10. (Semantic Security [105, Ch. 5]) An encryption scheme is **semantically secure** if for every probabilistic polynomial-time algorithm, A, there exists a probabilistic polynomial-time algorithm A' such that for every two polynomially bounded functions $f, h : \{0,1\}^* \to \{0,1\}^*$ and for any probability ensemble $\{X_n\}_{n \in \mathbb{N}}$, $|X_n| \leq poly(n)$, for any positive polynomial p and sufficiently large n:

$$\Pr\left[A(\texttt{Enc}(X_n), h(X_n)) = f(X_n)\right] - \Pr\left[A'(h(X_n)) = f(X_n)\right] < \eta(n),$$

where $\texttt{Enc}(\cdot)$ is the encryption primitive.

Semantic security is equivalent to the cryptosystem having indistinguishable encryptions, which, in essence, means that an adversary that has the plaintext messages a and b cannot distinguish between the encryptions $\texttt{Enc}(a)$ and $\texttt{Enc}(b)$.

An encryption scheme has a *computational security parameter* κ if all known attacks against the scheme, including breaking the encryption or distinguishing between the encryptions of different messages, take 2^κ bit operations. In practice, at least 128 bits of security are preferred [5].

The strength of a cryptosystem relies on the computational intractability of retrieving the private key from the public information–an adversary holding the public information cannot find the private key by brute force computations.

2.3 Pseudorandom objects

A *pseudorandom generator* (PRG) is an efficient deterministic function that expands short seeds into longer pseudorandom bit sequences, which are computationally indistinguishable from truly random sequences.

A *pseudorandom function* (PRF) is a family of efficiently-computable keyed functions that produce an output which is computationally indistinguishable from the output of a random function. More details can be found in [104, Ch. 3], [133, Ch. 3].

Pseudorandom functions can be constructed from pseudorandom generators. In practice, pseudorandom functions are instantiated from block ciphers AES [76], or cryptographic hash functions like SHA-3 or newer sponge-based functions [37].

2.4 Secret sharing

Secret sharing [184, 201] is a tool that distributes a secret message to a number of parties, by splitting it into random shares. Specifically, t-out-of-n secret sharing splits a secret message into n shares and distributes them to different parties; then, the secret message can be reconstructed by an authorized subset of parties, which have to combine at least t shares.

The t-out-of-n secret sharing is called polynomial secret sharing, while n-out-of-n secret sharing is usually called additive secret sharing.

One common scheme is the additive 2-out-of-2 secret sharing scheme, which splits a secret message m in a message space $\mathcal{M}$ into two shares by: generating uniformly at random an element $s \in \mathcal{M}$, adding it to the message and then distributing the shares s and $m + s \in \mathcal{M}$. (The message space is a finite abelian group.) Both shares are needed in order to recover the secret. This can be also thought of as a one-time pad [31, 220] variant on $\mathcal{M}$.

Additive blinding is a weaker scheme than secret sharing, but sufficient for many applications. For a message $m \in \mathcal{M}$ of l bits, s will be uniformly sampled from $(0, 2^{l+\lambda})$, where λ is the *statistical security parameter*. The statistical security parameter can take values of around 80 bits.

Theorem 2.4.1. *(a) Additive secret sharing is perfectly secret when $s \in \mathcal{M}$ [74];*
(b) Additive blinding is λ-statistically secure when $s \in (0, 2^{l+\lambda})$.

Since $\mathcal{M}$ is a finite abelian group, addition in $\mathcal{M}$ involves wrap-around, which ensures that the distributions of m and $m + s$ are identical. This proves (a).

The proof of (b) follows from computing the advantage an adversary has for distinguishing between $m + s \in (0, 2^{l+\lambda} + 2^l)$ and a uniformly sampled random value $r \in (0, 2^{l+\lambda})$, which is $1/2 + 2^{-\lambda}$ (the statistical distance between the two sets is $2^{-\lambda}$).

2.5 Homomorphic encryption schemes

Let $\texttt{Enc}(\cdot)$ denote a generic encryption primitive, with domain the space of private data, called *plaintexts*, and codomain the space of encrypted data, called *ciphertexts*. $\texttt{Enc}(\cdot)$ also takes as input the public key, and probabilistic encryption primitives also take a random number. The decryption primitive $\texttt{Dec}(\cdot)$ is defined on the space of ciphertexts and takes values on the space of plaintexts. $\texttt{Dec}(\cdot)$ also takes as input the private key. Asymmetric or public key cryptosystems involve a pair of keys: a public key that is disseminated publicly, and which is used for the encryption of the private messages, and a private key which is known only to its owner, used for the decryption of the encrypted messages.

The decryption primitive of a homomorphic encryption scheme is a homomorphism from the space of ciphertexts to the space of plaintexts. An encryption scheme is called *partially homomorphic* if it supports the encrypted evaluation of either a linear polynomial or a monomial.

Specifically, *additively homomorphic* schemes satisfy the property that there exists an operator $\oplus$ defined on the space of ciphertexts such that:

$$\texttt{Enc}(a) \oplus \texttt{Enc}(b) \subset \texttt{Enc}(a+b), \tag{2.5.1}$$

for any plaintexts a, b supported by the scheme. We use set inclusion instead of equality because the encryption of a message is not unique in probabilistic cryptosystems. Intuitively, equation (2.5.1) means that by performing this operation on the two encrypted messages, we obtain a ciphertext that is equivalent to the encryption of the sum of the two plaintexts. Formally, the decryption primitive $\texttt{Dec}(\cdot)$ is a homomorphism between the group of ciphertexts with the operator $\oplus$ and the group of plaintexts with addition $+$, which justifies the name of the scheme. It is immediate to see that if a scheme supports addition between encrypted messages, it will also support subtraction, by adding the additive inverse, and multiplication between an integer plaintext and an encrypted message, obtained by adding the encrypted messages for the corresponding number of times.

Multiplicatively homomorphic schemes satisfy the property that there exists an operator $\otimes$ defined on the space of ciphertexts such that:

$$\texttt{Enc}(a) \otimes \texttt{Enc}(b) \subset \texttt{Enc}(a \cdot b), \tag{2.5.2}$$

for any plaintexts a, b supported by the scheme.

Furthermore, an encryption scheme is called *somewhat* or *leveled homomorphic* if it supports the encrypted evaluation of a polynomial with a finite degree. In other words, it satisfies both (2.5.1) and (2.5.2) but only for a limited amount of operations

Finally, an encryption scheme is called *fully homomorphic* if it supports the encrypted evaluation of arbitrary polynomials, i.e., it satisfies both (2.5.1) and (2.5.2) for an unlimited amount of operations. Leveled homomorphic encryption schemes can be turned into fully homomorphic encryption schemes if a *bootstrapping* operation is enabled, which can *refresh* the ciphertext after the levels were consumed, such that further operations are allowed while still guaranteeing correct decryption.

Remark 2.5.1. A homomorphic cryptosystem is malleable, which means that a party that does not have the private key can alter a ciphertext such that another valid ciphertext is obtained. Malleability is a desirable property in order to achieve third-party outsourced computation on encrypted data, but allows ciphertext attacks. In this work, we assume that the parties have access to authenticated channels, therefore an adversary cannot alter the messages sent by the honest parties.

In the rest of the thesis, we will slightly abuse the notation $\texttt{Enc}(\cdot)$ to describe encrypting vectors and matrices (i.e. each element is individually encrypted).

2.5.1 Additively homomorphic encryption

Additively homomorphic encryptions schemes, abbreviated as AHE, can be instantiated by various schemes such as [80, 107, 130, 177]. Let AHE $=(\mathrm{KeyGen}, \mathrm{E}, \mathrm{D}, \mathrm{Add}, \mathrm{cMlt})$ be an instance of an asymmetric additively homomorphic encryption scheme, with $\mathcal{M}$ the message space and $\mathcal{C}$ the ciphertext space. We will use the following abstract notation: $\oplus$ denotes the addition on $\mathcal{C}$ and $\otimes$ denotes the multiplication between a plaintext and a ciphertext.

1. $\mathrm{KeyGen}(1^\sigma)$: Takes the security parameter σ and outputs a public key pk and a private key sk.
2. $\mathrm{E}(\mathrm{pk}, m)$: Takes the public key and a message $m \in \mathcal{M}$ and outputs a ciphertext in $\mathcal{C}$.
3. $\mathrm{D}(\mathrm{sk}, c)$: Takes the private key and a ciphertext $c \in \mathcal{C}$ and outputs the message that was encrypted $m' \in \mathcal{M}$.
4. $\mathrm{Add}(c_1, c_2)$: Takes ciphertexts $c_1, c_2 \in \mathcal{C}$ and outputs ciphertext $c = c_1 \oplus c_2 \in \mathcal{C}$ such that: $\mathrm{D}(\mathrm{sk}, c) = \mathrm{D}(\mathrm{sk}, c_1) + \mathrm{D}(\mathrm{sk}, c_2)$.

5. cMlt(m_1, c_2): Takes plaintext $m_1 \in \mathcal{M}$ and ciphertext $c_2 \in \mathcal{C}$ and outputs ciphertext $c = m_1 \otimes c_2 \in \mathcal{C}$ such that: $\mathrm{D}(\mathrm{sk}, c) = m_1 \cdot \mathrm{D}(\mathrm{sk}, c_2)$.

The **Paillier cryptosystem** [177] is an asymmetric additively homomorphic encryption scheme. The private data are elements of the ring of integers modulo N, denoted by $\mathbb{Z}_N$, where N is a large integer of σ bits, called the Paillier modulus. The ciphertexts take values in the multiplicative group of integers modulo N^2, denoted by $\mathbb{Z}^*_{N^2}$. Paillier is a probabilistic encryption scheme, which means that the encryption primitive also takes as an argument a random number which, for simplicity, we omit in this notation. For compactness, we will denote the Paillier encryption of a message $a \in \mathbb{Z}_N$ by $[[a]]$ as a shorthand notation for $\mathrm{E}(\mathrm{pk}, a)$, for an instance of the random number. For readability, we will use throughout the manuscript the following abstract notation for the operations on the encrypted space:

$$[[a]] \oplus [[b]] \overset{d}{=} [[a+b]], \qquad b \otimes [[a]] \overset{d}{=} [[ba]], \qquad \text{for plaintexts } a, b \in \mathbb{Z}_N, \tag{2.5.3}$$

where $\overset{d}{=}$ means that the equality holds after applying the decryption primitive on both sides.

The pair of keys corresponding to this cryptosystem is $(\mathrm{pk}, \mathrm{sk})$, where the public key is $\mathrm{pk} = (N, g)$ and the secret key is $\mathrm{sk} = (\gamma, \delta)$. N called the modulus and is the product of two large prime numbers p, q of the same bit length, and g is an element of order N in $\mathbb{Z}^*_{N^2}$, commonly selected to be $N + 1$, which we also prefer here. Furthermore,

$$\gamma = \phi(N) = (p-1)(q-1), \quad \delta = \phi(N)^{-1} \bmod N,$$

where ϕ is Euler's totient function. For a plaintext $a \in \mathbb{Z}_N$, the Paillier encryption is:

$$[[a]] = g^a r^N \bmod N^2, \tag{2.5.4}$$

where r is a random nonzero element in $\mathbb{Z}_N$, which makes Paillier a probabilistic encryption scheme. The decryption primitive is the following:

$$a = \frac{[[a]]^\gamma \bmod N^2 - 1}{N}\, \delta \bmod N, \tag{2.5.5}$$

which uses the fact that $(1+N)^a = 1 + Na \bmod N^2$.

In the Paillier scheme, in order to obtain addition between plaintexts, the operation between ciphertexts is modular multiplication, which was denoted by $\oplus$ in the text:

$$[[a]] \cdot [[b]] = g^a r^N \cdot g^b r'^N \bmod N^2 = g^{a+b}(rr')^N = [[a+b]] \bmod N^2. \tag{2.5.6}$$

The multiplication between a plaintext value c and an encrypted value $[[a]]$, which was denoted by $c \otimes [[a]]$ in the text, is obtained in the following way:

$$[[a]]^c = g^{ac}(r^c)^N \bmod N^2 = [[ca]]. \tag{2.5.7}$$

It follows that negation is achieved by modular inverse.

Theorem 2.5.2. *The Paillier cryptosystem is semantically secure [177] under the Decisional Composite Residuosity assumption.*

The Decisional Composite Residuosity hardness problem is described in Appendix A.1.

The **DGK cryptosystem**. In [77, 78], Damgård, Geisler and Krøigaard describe a protocol for secure comparison. Motivated by that functionality, they propose the DGK additively homomorphic encryption scheme, in which it is efficient to determine if a given ciphertext is an encryption of zero. This property is useful for comparisons and when working with bits.

The plaintext space for DGK is $\mathbb{Z}_u$, where u is a small prime divisor of $p-1$ and $q-1$, p and q are large, same size prime numbers, and $N = pq$. The parameters v_p and v_q are t-bit prime divisors of $p-1$ and $q-1$. The numbers g and h are elements of $\mathbb{Z}_N^*$ of order uv_pv_q and v_pv_q. The DGK encryption scheme has public key $\mathrm{pk}_{\mathrm{DGK}} = (N, g, h, u)_{\mathrm{DGK}}$ and $\mathrm{sk}_{\mathrm{DGK}} = (p, q, v_p, v_q)_{\mathrm{DGK}}$. In order to distinguish a DGK ciphertext from a Paillier ciphertext we use the notation $[\cdot]$. For a plaintext $x \in \mathbb{Z}_u$, a DGK encryption is:

$$[a] = g^a h^r \bmod N,$$

where r is a random $2t$-bits integer. We use this encryption scheme with the purpose of encrypting and decrypting bits that represent comparison results, therefore, for decryption, we only need to check if the encrypted value is 0. For this, it is enough to verify:

$$[a]^{v_p v_q} \bmod p = 1$$

to see if $a = 0$. DGK is additively homomorphic so the following holds:

$$a + b = \mathrm{D}([a] \cdot [b] \bmod N), \qquad -a = \mathrm{D}([a]^{-1} \bmod N). \tag{2.5.8}$$

We will use again $\oplus$ and $\otimes$ to abstract the addition between and encrypted values, respectively, the multiplication between a plaintext and an encrypted values.

Theorem 2.5.3. *The DGK cryptosystem is semantically secure [77, 78], under the hardness of factoring assumption.*

Remark 2.5.4. We also notice the additive blinding scheme from Preamble 2.4 can be viewed as a one-time symmetric key-homomorphic and message-homomorphic encryption, as long as there is no overflow: $\mathrm{E}'(m) = m - s$, $\mathrm{D}'(\mathrm{E}'(m)) = \mathrm{E}'(m) + s = m$, with s being the secret-key. *This symmetric cryptosystem is compatible with the Paillier and DGK cryptosystems, in the sense that the two encryptions commute*: $\mathrm{D}(\mathrm{D}'(\mathrm{E}(\mathrm{E}'(m)))) = m$ by using $\mathrm{E}(s)$ instead of s for decryption D', where $\mathrm{E}'(m)$ is performed on the message space of the cryptosystem, and similarly $\mathrm{D}'(\mathrm{D}(\mathrm{E}(m) \oplus \mathrm{E}(s))) = m$.

2.5.2 Somewhat homomorphic encryption

Somewhat Homomorphic Encryption (SHE) schemes generally allow many to unlimited homomorphic addition operations and a limited number of sequential homomorphic multiplications. SHE differs from leveled HE in the following sense: from their construction, SHE schemes support a fixed number of sequential multiplications (e.g., one or two), whereas leveled HE take as a parameter the number of multiplications when instantiating the scheme and can be even made to support a different number of multiplications by bootstrapping.

Labeled Homomorphic Encryption (LabHE) is a SHE scheme that allows unlimited additions and one multiplication between ciphertexts. This scheme assumes that the decryptor knows the program to be executed on the encrypted data. The encryptor assigns a unique label to each message and sends the encrypted data along with the corresponding encrypted labels to the server.

Denote by $\mathcal{M}$ the message space. An *admissible function* for LabHE $f : \mathcal{M}^n \to \mathcal{M}$ is a multivariate polynomial of degree 2 on n variables. A program that has labeled inputs is called a labeled program [27]:

Definition 2.5.5. A **labeled program** $\mathcal{P}$ is a tuple $(f, \tau_1, \dots, \tau_n)$, where $f : \mathcal{M}^n \to \mathcal{M}$ is an admissible function on n variables and $\tau_i \in \{0,1\}^*$ is the label of the i-th input of f.

LabHE is constructed from an AHE scheme with the requirement that the message space must be a public ring in which one can efficiently sample elements uniformly at random. In what follows, we will use the Paillier cryptosystem as the underlying AHE scheme. The idea is that an encryptor splits their private message as described in Preamble 2.4 into a random value (secret) and the difference between the message and the secret. For efficiency, instead of taking the secret to be a uniformly random value, we take it to be the output of a pseudorandom function applied to the corresponding label. The label acts like the seed of the pseudorandom function. The encryptor then forms the LabHE ciphertext from the encryption of the first share along with the second share, yielding $\hat{\mathrm{E}}(m) = (m - \mathrm{b}, [[\mathrm{b}]])$, as described in Step 1 in the following. This enables us to decrypt one multiplication of two encrypted values, using the observation (2.5.9). The AHE scheme allows computing $(m_1 - \mathrm{b}_1) \otimes [[\mathrm{b}_2]]$, $(m_2 - \mathrm{b}_2) \otimes [[\mathrm{b}_1]]$ and $[[(m_1 - \mathrm{b}_1) \cdot (m_2 - \mathrm{b}_2)]]$, for plaintexts $(m_i - \mathrm{b}_i)$, $i = 1, 2$. Hence, we can obtain the AHE encryption of one multiplication $[[m_1 \cdot m_2 - \mathrm{b}_1 \cdot \mathrm{b}_2]]$ from $\hat{\mathrm{E}}(m_1)$ and $\hat{\mathrm{E}}(m_1)$, described in Step 4 in the following. Decryption, described in Step 5, requires that the decryptor knows the private key of the AHE scheme, and b_i, such that it can compute $m_1 \cdot m_2 = \mathrm{D}[[m_1 \cdot m_2 - \mathrm{b}_1 \cdot \mathrm{b}_2]] + \mathrm{b}_1 \cdot \mathrm{b}_2$.

$$m_1 \cdot m_2 - \mathrm{b}_1 \cdot \mathrm{b}_2 = (m_1 - \mathrm{b}_1) \cdot (m_2 - \mathrm{b}_2) + \mathrm{b}_1 \cdot (m_2 - \mathrm{b}_2) + \mathrm{b}_2 \cdot (m_1 - \mathrm{b}_1). \qquad (2.5.9)$$

Let $\mathcal{M}$ be the message space of the AHE scheme, $\mathcal{L} \subset \{0,1\}^*$ denote a finite set of labels and $F : \{0,1\}^k \times \{0,1\}^* \to \mathcal{M}$ be a pseudorandom function that takes as inputs a key of size k polynomial in σ the security parameter, and a label from $\mathcal{L}$. LabHE is defined as a tuple LabHE = $(\hat{\mathrm{Init}}, \hat{\mathrm{KeyGen}}, \hat{\mathrm{E}}, \hat{\mathrm{Eval}}, \hat{\mathrm{D}})$:

1. $\hat{\mathrm{Init}}(1^\sigma)$: Takes the security parameter σ and outputs master secret key msk and master public key mpk for AHE.

2. KeyGen(mpk): Takes the master public key mpk and outputs for each user i a user secret key $\mathrm{usk_i}$ and a user public key $\mathrm{upk_i}$.

3. $\hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{upk}, \tau, m)$: Takes the master public key, a user public key, a label $\tau \in \mathcal{L}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $C = (a, \beta)$. It is composed of an online and offline part:

- Off-$\hat{\mathrm{E}}(\mathrm{usk}, \tau)$: Computes the secret $\mathrm{b} \leftarrow F(\mathrm{usk}, \tau)$ and outputs $\mathrm{C_{off}} = (\mathrm{b}, [[\mathrm{b}]])$.

- On-$\hat{\mathrm{E}}(\mathrm{C_{off}}, m)$: Outputs $C = (m - \mathrm{b}, [[\mathrm{b}]]) =: (a, \beta) \in \mathcal{M} \times \mathcal{C}$.

4. $\hat{\mathrm{Eval}}(\mathrm{mpk}, f, C_1, \ldots, C_t)$: Takes the master public key, an admissible function $f : \mathcal{M}^t \to \mathcal{M}$, t ciphertexts and returns a ciphertext C. $\hat{\mathrm{Eval}}$ is composed of the following building blocks:

- $\hat{\mathrm{Mlt}}(C_1, C_2)$: Takes $C_i = (a_i, \beta_i) \in \mathcal{M} \times \mathcal{C}$ for $i = 1, 2$ and outputs $C = [[a_1 \cdot a_2]] \oplus (a_1 \otimes \beta_2) \oplus (a_2 \otimes \beta_1) = [[m_1 \cdot m_2 - \mathrm{b}_1 \cdot \mathrm{b}_2]] =: \alpha \in \mathcal{C}$.

- $\hat{\mathrm{Add}}(C_1, C_2)$: If $C_i = (a_i, \beta_i) \in \mathcal{M} \times \mathcal{C}$ for $i = 1, 2$, then outputs $C = (a_1 + a_2, \beta_1 \oplus \beta_2) =: (a, \beta) \in \mathcal{M} \times \mathcal{C}$. If both $C_i = \alpha_i \in \mathcal{C}$, for $i = 1, 2$, then outputs $C = \alpha_1 \oplus \alpha_2 =: \alpha \in \mathcal{C}$. If $C_1 = (a_1, \beta_1) \in \mathcal{M} \times \mathcal{C}$ and $C_2 = \alpha_2 \in \mathcal{C}$, then outputs $C = (a_1, \beta_1 \oplus \alpha_2) =: (a, \beta) \in \mathcal{M} \times \mathcal{C}$.

- $\mathrm{c\hat{M}lt}(c, C')$: Takes a plaintext $c \in \mathcal{M}$ and a ciphertext C'. If $C' = (a', \beta') \in \mathcal{M} \times \mathcal{C}$, outputs $C = (c \cdot a', c \otimes \beta') =: (a, \beta) \in \mathcal{M} \times \mathcal{C}$. If $C' = \alpha' \in \mathcal{C}$, outputs $C = c \otimes \alpha' =: \alpha \in \hat{\mathcal{C}}$.

5. $\hat{\mathrm{D}}(\mathrm{msk}, \mathrm{usk}_{1,\ldots,t}, \mathcal{P}, C)$: Takes the master secret key, a vector of t user secret keys, a labeled program $\mathcal{P}$ and a ciphertext C. It has an online and an offline part:

- Off-$\hat{\mathrm{D}}(\mathrm{msk}, \mathcal{P})$: Parses $\mathcal{P}$ as $(f, \tau_1, \ldots, \tau_t)$. For $i \in [t]$, it computes the secrets $\mathrm{b}_i = F(\mathrm{usk}_i, \tau_i)$, $\mathrm{b} = f(\mathrm{b}_1, \ldots, \mathrm{b}_t)$ and outputs $\mathrm{msk}_{\mathcal{P}}(\mathrm{msk}, \mathrm{b})$.

- On-$\hat{\mathrm{D}}(\mathrm{sk}_{\mathcal{P}}, C)$: If $C = (a, \beta) \in \mathcal{M} \times \mathcal{C}$: either output (i) $m = a + \mathrm{b}$ or (ii) output $m = a + \mathrm{D}(\mathrm{msk}, \beta)$. If $C \in \mathcal{C}$, output $m = \mathrm{D}(\mathrm{msk}, C) + \mathrm{b}$.

The cost of an online encryption is the cost of an addition in $\mathcal{M}$. The cost of online decryption is independent of $\mathcal{P}$ and only depends on the complexity of D.

Theorem 2.5.6. *([27])* LabHE *satisfies correctness (the probability of the scheme not correctly evaluating the allowed class of functions is negligible), succinctness (the ciphertexts have polynomial length in the security parameter), semantic security (the ciphertexts are indistinguishable from one another) and context-hiding (decrypting the ciphertext does not reveal anything about the inputs of the computed function, only the result of the function on those inputs).*

2.5.3 Leveled fully homomorphic encryption

Recall that leveled homomorphic encryption schemes allow the evaluation of a multivariate polynomial on encrypted data. The common term for such a multivariate polynomial functionality is *arithmetic circuit* and the logarithm of the degree of the polynomial is the *multiplicative depth* of the circuit.

The newer generations of leveled/fully HE schemes are based on lattices and rely on the Ring Learning with Errors [4, 156] hardness problem, described in Appendix A.2. Each operation evaluated on ciphertexts introduces some noise, which, if it overflows, can prevent correct decryption. Multiplications introduce the most noise, so we focus on the number of sequential multiplications allowed in an instance of a leveled homomorphic scheme, which we call *multiplicative budget*. Correct decryption is only guaranteed for the evaluation of circuits of smaller multiplicative depth than the multiplicative budget.

The **CKKS scheme** (Cheon-Kim-Kim-Song [62]) is a leveled homomorphic encryption scheme. CKKS can perform operations on encrypted real numbers with a smaller error than other leveled homomorphic schemes. Each real number is multiplied by a positive integer scaling factor and truncated, as commonly done, and one can remove the extra scaling factor occurring in the result after a multiplication, through the rescaling procedure, at very little error. This manages the magnitude of the underlying plaintexts, which could otherwise cause overflow in a large depth circuit.

A ciphertext's size grows with the number of sequential multiplications it supports. Each ciphertext, respectively plaintext, is characterized by a *level*, a *scaling depth* and a number of *moduli*. A new ciphertext (freshly encrypted, rather than obtained as the result of operations on other ciphertexts) is on level L, scaling depth 1 and has as many moduli as the multiplicative budget minus one. The number L minus the current level corresponds to the number of rescaling operations previously performed on the ciphertext and scaling depth corresponds to the number of multiplications without rescaling that have been performed. After a multiplication followed by a rescaling procedure, the number of levels decreases by one, the scaling depth remains the same and one modulus is dropped. To avoid some technicalities and parallel the notion of circuit depth, in the rest of the manuscript, when we refer to the *depth* of a ciphertext, we refer to the multiplicative depth of the arithmetic circuit used in order to obtain that ciphertext. If rescaling is done after every multiplication, then the depth will be the scaling depth minus one.

In the CKKS scheme, plaintexts and ciphertexts are polynomials in the ring of integers of a cyclotomic field. Intuitively, this enables us to encode multiple scalar values in a plaintext or ciphertext. Hence, we *obtain ciphertexts that contain the encryption of a vector*. This is a standard practice in lattice-based homomorphic encryption that allows performing single instruction multiple data (SIMD) operations, and can bring major computation and storage improvements when evaluating an arithmetic circuit. We can pack multiple values in one plaintext/ciphertext–packing can be thought of as the ciphertext having multiple independent data slots. Abstracting the details away, the SIMD operations that can be supported are addition, element-wise multiplication by a plaintext or ciphertext and data slot permutations that can achieve ciphertext rotations (e.g., used for summing up the values in every slot). However, once the values are packed, extractions of individual values from the ciphertexts can require multiplicative masking and increase the circuit multiplicative depth.

More specifically, let M be a power of two, and let $\Phi_M(X)$ denote the M-th cyclotomic polynomial of degree $N := \phi(M)/2$, i.e., $X^N + 1$. The plaintext space is the ring of polynomials $\mathcal{R} := \mathbb{Z}[X]/\langle \Phi_M(X) \rangle$. N is also called the ring dimension for the plaintext space. Define the residue ring $\mathcal{R}_l := \mathcal{R}/Q_l\mathcal{R} = \mathbb{Z}_{Q_l}[X]/\langle \Phi_M(X) \rangle$, where Q_l is the ciphertext modulus corresponding to level l. The ciphertext space for level l is the ring $\mathcal{R}_l \times \mathcal{R}_l$. The CKKS scheme supports an efficient packing of up to $N/2$ real values in a single plaintext, respectively a single ciphertext. A real-valued message in $\mathbb{C}^N$ is encoded in a plaintext in $\mathcal{R}$ via the inverse canonical embedding map and rounding, which corresponds to evaluation of the inverse Discrete Fourier Transform (with the primitive M-th roots of unity $\exp(-2\pi i/M)$). The decoding procedure is given by the forward Discrete Fourier Transform.

We next sketch the algorithms in the basic CKKS scheme. There are several optimizations available that employ the Residue Number System detailed in [40, 64, 117].

1. $\mathcal{S}\mathrm{etup}(1^\kappa, Q_L)$ Given the security parameter κ, for the largest ciphertext modulus Q_L, output the ring dimension N. Set the small distributions $\chi_{key}, \chi_{err}, \chi_{enc}$ over $\mathcal{R}$ for the secret, error and encryption.

2. $\mathcal{K}\mathrm{ey}\mathcal{G}\mathrm{en}(\mathcal{R}, \chi_{key}, \chi_{err}, \chi_{enc})$: Sample a secret $s \leftarrow \chi_{key}$, a random $a \leftarrow \mathcal{R}_L$ and error $e \leftarrow \chi_{err}$. Set the secret key $\mathbf{sk} \leftarrow (1, s)$ and output the public key $\mathbf{pk} \leftarrow (b, a) \in \mathcal{R}_L^2$, where $b \leftarrow -as + e(\mathrm{mod} Q_L)$.

3. $\mathcal{K}\mathcal{S}\mathcal{G}\mathrm{en}_{\mathbf{sk}}(\mathcal{R}, \chi_{key}, \chi_{err}, \chi_{enc})$: For $s' \in \mathcal{R}$, sample a random $a' \leftarrow \mathcal{R}_{2L}$ and error $e' \leftarrow \chi_{err}$. Output the switching key $\mathbf{swk} \leftarrow (b', a') \in \mathcal{R}_{2L}^2$, where $b' \leftarrow -a's +$

$e' + Q_L s' (\mathrm{mod} Q_{2L})$. Output the evaluation key $\mathbf{evk} \leftarrow \mathcal{K}\mathrm{ey}\mathcal{G}\mathrm{en}_{\mathbf{sk}}(s^2)$. Output the rotation key $\mathbf{rk}^{(\kappa)} \leftarrow \mathcal{K}\mathrm{ey}\mathcal{G}\mathrm{en}_{\mathbf{sk}}(\mathbf{s}^{(\kappa)})$, for rotation index κ.

4. $\mathcal{E}_{\mathbf{pk}}(m)$: For $m \in \mathcal{R}$, sample $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$. Output $\mathbf{ct} \leftarrow v \cdot \mathbf{pk} + (m + e_0, e_1)(\mathrm{mod} Q_L)$.

5. $\mathcal{D}_{\mathbf{sk}}(\mathbf{ct})$: For $\mathbf{ct} \leftarrow (c_0, c_1) \in \mathcal{R}_l^2$, output $\tilde{m} = c_0 + c_1 \cdot s (\mathrm{mod} Q_l)$.

6. $\mathcal{CA}\mathrm{dd}(\mathbf{ct}, c)$: For $\mathbf{ct} = (c_0, c_1) \in \mathcal{R}_l^2$ and $c \in \mathcal{R}$, output $\mathbf{ct}_{CAdd} \leftarrow (c_0 + c, c_1)(\bmod Q_l)$.

7. $\mathcal{A}\mathrm{dd}(\mathbf{ct}_1, \mathbf{ct}_2)$: For $\mathbf{ct}_1, \mathbf{ct}_2 \in \mathcal{R}_l^2$, output $\mathbf{ct}_{Add} \leftarrow \mathbf{ct}_1 + \mathbf{ct}_2 (\mathrm{mod} Q_l)$.

8. $\mathcal{CM}\mathrm{ult}(c, \mathbf{ct})$: For $\mathbf{ct} \in \mathcal{R}_l^2$ and $c \in \mathcal{R}$, output $\mathbf{ct}_{CMult} \leftarrow c \cdot \mathbf{ct} (\mathrm{mod} Q_l)$.

9. $\mathcal{M}\mathrm{ult}_{\mathbf{evk}}(\mathbf{ct}_1, \mathbf{ct}_2)$: For $\mathbf{ct}_i = (b_i, a_i) \in \mathcal{R}_l^2$, for $i = 1, 2$, let $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2)(\mathrm{mod} Q_l)$. Output $\mathbf{ct}_{Mult} \leftarrow (d_0, d_1) + \lfloor Q_L^{-1} \cdot d_2 \cdot \mathbf{evk} \rceil (\mathrm{mod} Q_l)$.

10. $\mathcal{R}\mathrm{otate}_{\mathbf{rk}^{(\kappa)}}(\mathbf{ct})$: For $\mathbf{ct} = (c_0, c_1) \in \mathcal{R}_l^2$ and rotation index κ, output $\mathbf{ct}_{rot} \leftarrow (c_0^{(\kappa)}, 0) + \lfloor Q_L^{-1} \cdot c_1^{(\kappa)} \cdot \mathbf{rk}^{(\kappa)} \rceil (\mathrm{mod} Q_l)$.

11. $\mathcal{R}\mathrm{e}\mathcal{S}\mathrm{cale}(\mathbf{ct}, p)$: For a ciphertext $\mathbf{ct} \in \mathcal{R}_l^2$ and an integer p, output $\mathbf{ct'} \leftarrow \lfloor 2^{-p} \cdot \mathbf{ct} \rceil (\mathrm{mod} Q_l / 2^p)$.

Theorem 2.5.7. *The CKKS scheme is semantically secure [62] assuming the parameters are selected so the Decisional Ring Learning with Errors problem is computationally hard.*

2.6 Oblivious transfer

Oblivious transfer is a technique used when one party wants to obtain a secret from a set of secrets held by another party [105, Ch. 7]. Party A has k secrets $(\sigma_0, \ldots, \sigma_{k-1})$ and party B has an index $i \in \{0, \ldots, k-1\}$. The goal of A is to transmit the i-th secret requested by the receiver without knowing the value of the index i, while B does not learn anything other than σ_i. This is called 1-out-of-k oblivious transfer. There are many constructions of oblivious transfer that achieve security as in the two-party version of the simulation definition (Definition 2.2.6). Many improvements in efficiency, e.g., precomputation, and security have been proposed, see e.g., [21, 128, 174].

We will use an 1-out-of-2 oblivious transfer, where the inputs of party A are $[[\sigma_0]], [[\sigma_1]]$ and party B holds $i \in \{0,1\}$ and the secret key and has to obtain σ_i. We will denote this by $\sigma_i \leftarrow \mathrm{OT}([[\sigma_0]], [[\sigma_1]], i, \mathrm{sk})$. We will also use a variant where party A has to obliviously obtain the AHE-encrypted $[[\sigma_i]]$, and A has $[[\sigma_0]], [[\sigma_1]]$ and party B holds i, for $i \in \{0,1\}$, and the secret key. We will denote this variant by $[[\sigma_i]] \leftarrow \mathrm{OT}'([[\sigma_0]], [[\sigma_1]], i, \mathrm{sk})$.

The way the variant OT' works is that A chooses at random r_0, r_1 from the message space $\mathcal{M}$, and sends shares of the messages to B: $[[v_0]] := \mathrm{Add}([[\sigma_0]], [[r_0]]), [[v_1]] := \mathrm{Add}([[\sigma_1]], [[r_1]])$. B selects v_i and sends back to A the encryption of the index i: $[[i]]$ and $\mathrm{Add}([[v_i]], [[0]])$, such that A cannot obtain information about i by comparing the value it received with the values it sent. Then, A computes:

$$[[\sigma_i]] = \mathrm{Add}\Big([[v_i]], \mathrm{cMlt}\big(r_0, \mathrm{Add}([[i]], [[-1]])\big), \mathrm{cMlt}(-r_1, [[i]])\Big).$$

Proposition 2.6.1. $[[\sigma_i]] \leftarrow \mathrm{OT}'([[\sigma_0]], [[\sigma_1]], i, \mathrm{sk})$ *is private w.r.t. Definition 2.2.5.*

The proof is given in Appendix A.3.

We will use this oblivious transfer functionality for protocols in Chapters 3 and 6.

2.7 Private two-party comparison

Consider a two-party computation problem under an encryption scheme that does not support comparison between encrypted data. A large number of secure comparison protocols on private inputs owned by two parties have been proposed in the literature, see [77, 98, 140, 153], with a survey of the state of the art given in [73]. In [77, 78], Damgård, Geisler and Krøigaard describe a protocol for secure comparison using the DGK additively homomorphic encryption scheme described in Preamble 2.5.1, whose ciphertexts are denoted as $[\cdot]$.

Consider two parties A and B, each having a private value α and β. Using the binary representations of α and β, the two parties exchange l blinded and encrypted values such that each of the parties will obtain a bit $\delta_A \in \{0,1\}$ and $\delta_B \in \{0,1\}$ that satisfy the following relation: $\delta_A \veebar \delta_B = (\alpha \leq \beta)$, after executing Protocol 2.7.1, where $\veebar$ denotes the exclusive or operation. The protocol is described in [221, Protocol 3], where an improvement of the DGK scheme is proposed. By applying some extra steps, as in [221, Protocol 2], one can obtain a protocol for private two-party comparison where party A has two encrypted inputs with an AHE scheme $[[a]], [[b]]$, with a, b represented on l bits, using the fact that the most significant bit of $(b - a + 2^l)$ is the bit that indicates if $(a \leq b)$, shown in Protocol 2.7.2.

Proposition 2.7.1. *([77, 221]) Protocol 2.7.2 for secure two-party comparison is private in the semi-honest model w.r.t. Definition 2.2.6.*

PROTOCOL 2.7.1: Private two-party comparison with plaintext inputs using DGK

```
Input: A: $\alpha$; B: $\beta, \mathrm{sk}_{\mathrm{DGK}}$
Output: A: $\delta_A$; B: $\delta_B$ such that $\delta_A \veebar \delta_B = (\alpha \leq \beta)$
 1: B: send the encrypted bits $[\beta_i]$, $0 \leq i < l$ to A.
 2: for each $0 \leq i < l$ do
 3:     A: $[\alpha_i \veebar \beta_i] \leftarrow [\beta_i]$ if $\alpha_i = 0$ and $[\alpha_i \veebar \beta_i] \leftarrow [1] \oplus (-1) \otimes [\beta_i]$ otherwise.
 4: end for
 5: A: Choose a uniformly random bit $\delta_A \in \{0, 1\}$.
 6: A: Compute the set $\mathcal{L} = \{i | 0 \leq i < l$ and $\alpha_i = \delta_A\}$.
 7: for each $i \in \mathcal{L}$ do
 8:     A: compute $[c_i] \leftarrow [\alpha_{i+1} \veebar \beta_{i+1}] \oplus \ldots \oplus [\alpha_l \veebar \beta_l])$ .
 9:     A: $[c_i] \leftarrow [1] \oplus [c_i] \oplus (-1) \otimes [\beta_i]$ if $\delta_A = 0$ and $[c_i] \leftarrow [1] \oplus [c_i]$ otherwise.
10: end for
11: A: generate uniformly random nonzero values $r_i$ of $2t$ bits (see [78]), $0 \leq i < l$.
12: for each $0 \leq i < l$ do
13:     A: $[c_i] \leftarrow r_i \otimes [c_i]$ if $i \in \mathcal{L}$ and $[c_i] \leftarrow [r_i]$ otherwise.
14: end for
15: A: send the values $[c_i]$ in random order to B.
16: B: if at least one of the values $c_i$ is decrypted to zero, set $\delta_B \leftarrow 1$, otherwise set $\delta_B \leftarrow 0$.
```

PROTOCOL 2.7.2: Private two-party comparison with encrypted inputs using DGK

```
Input: A: $[[a]], [[b]]$; B: $\mathrm{sk}, \mathrm{sk}_{\mathrm{DGK}}$
Output: B: $(\delta = 1) \equiv (a \leq b)$
1: A: choose uniformly at random $r$ of $l + 1 + \lambda$ bits, compute $[[z]] \leftarrow [[b]] \ominus [[a]] \oplus [[2^l + r]]$ and
   send it to B. Then compute $\alpha \leftarrow r \bmod 2^l$.
2: B: decrypt $[[z]]$ and compute $\beta \leftarrow z \bmod 2^l$.
3: A,B: execute Protocol 2.7.1.
4: B: send $[[z \div 2^l]]$ and $[[\delta_B]]$ to A.
5: A: $[[(\beta < \alpha)]] \leftarrow [[\delta_B]]$ if $\delta_A = 1$ and $[[(\beta < \alpha)]] \leftarrow [[1]] \ominus [[\delta_B]]$ otherwise.
6: A: compute $[[\delta]] \leftarrow [[z \div 2^l]] \ominus ([[r \div 2^l]] \oplus [[(\beta < \alpha)]])$ and send it to B.
7: B: decrypts $\delta$.
```

2.8 Notation

We denote vectors by lower-case bold letters, e.g., $\mathbf{x}$ and matrices by upper-case bold letters, e.g., $\mathbf{A}$. $\mathbb{R}^{m\times n}$ represents the set of $m \times n$ matrices with real elements and $\mathbb{S}^n_{++}$ represents the set of symmetric positive definite $n \times n$ matrices. $\mathbb{Z}$ denotes the set of integers, $\mathbb{Z}_N$ denotes the additive group of integers modulo N and $\mathbb{Z}^*_N$ denotes the multiplicative group of integers modulo N. The notation $x \leftarrow X$ means that x is uniformly drawn from a distribution X. Pr denotes the probability taken over a specified distribution. We refer to algorithms that are run interactively by multiple parties as protocols.

For a vector $\mathbf{x} \in \mathbb{R}^n$, its ℓ_1-norm is $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$, its ℓ_2-norm is $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$, and its ℓ_∞-norm is $\|\mathbf{x}\|_\infty = \max(|x_1|, \dots, |x_n|)$. The matrix-induced norm, for a matrix $\mathbf{P} \in \mathbb{S}^n_{++}$ is $\|\mathbf{x}\|_{\mathbf{P}} := \sqrt{\mathbf{x}^\intercal \mathbf{P} \mathbf{x}}$. Element-wise inequalities between vectors are denoted by $\preceq, \succeq, \prec, \succ$. For square matrices, $\mathbf{A} \succ 0$ means that $\mathbf{A}$ is positive definite and $\mathbf{A} \succeq 0$ means that it is positive semi-definite, with the converse meaning for $\prec, \preceq$.

We use the following notation for the different homomorphic encryption schemes:

- For additively homomorphic encryption schemes, in particular for the Paillier scheme, we use $\mathrm{E}(\cdot), \mathrm{D}(\cdot), \mathrm{Add}(\cdot), \mathrm{cMlt}(\cdot), \oplus, \ominus, \odot$ and $[[\cdot]]$; see Preamble 2.5.1.

- For the DGK scheme we use $[\cdot], \oplus, \ominus, \odot$; see Preamble 2.5.1.
- For somewhat homomorphic encryption schemes, in particular for the LabHE scheme, we use $\hat{\mathrm{E}}(\cdot), \hat{\mathrm{D}}(\cdot), \hat{\mathrm{Eval}}(\cdot), \hat{\mathrm{Add}}(\cdot), \mathrm{c}\hat{\mathrm{Mlt}}(\cdot), \hat{\mathrm{Mlt}}(\cdot), \hat{\oplus}, \hat{\ominus}, \hat{\odot}, \hat{\otimes}$; see Preamble 2.5.2.
- For leveled fully homomorphic encryption schemes, in particular for the CKKS scheme, we use $\mathcal{E}(\cdot), \mathcal{D}(\cdot), \mathcal{E}\mathrm{val}(\cdot), \mathcal{A}\mathrm{dd}(\cdot), \mathcal{CM}\mathrm{ult}(\cdot), \mathcal{M}\mathrm{ult}(\cdot), \mathcal{R}\mathrm{otate}(\cdot)$; see Preamble 2.5.3.

Chapter 3

Optimization Problems

3.1 Introduction

The development of large-scale distributed systems has led to the outsourcing of costly computations to cloud-computing platforms, as well as to concerns about privacy of the collected sensitive data. In control theoretic applications and machine learning, linear and quadratic optimization problems arise frequently – e.g., state estimation under minimum square error, model predictive control, support vector machines – which require privacy guarantees for the data involved in the computation.

The first part of this chapter develops a cloud-based protocol for a quadratic optimization problem involving multiple parties, each holding information it seeks to maintain private. The protocol is based on the projected gradient ascent on the Lagrange dual problem and exploits additively homomorphic encryption and secure multi-party computation techniques. Using formal cryptographic definitions of indistinguishability, the protocol is shown to achieve computational privacy, i.e., there is no computationally efficient algorithm that any involved party can employ to obtain private information beyond what can be inferred from the party's inputs and outputs only. In order to reduce the communication complexity of the proposed protocol, we introduced a variant that achieves this objective at the expense of weaker privacy guarantees. We discuss in detail the computational and communication complexity properties of both algorithms theoretically and also through implementations.

We conclude the first part with a discussion on computational privacy and other notions of privacy such as the non-unique retrieval of the private information from the protocol outputs.

In the second part of this chapter, we investigate how to securely evaluate another pervasive optimization problem, with different characteristics than quadratic programs: the least squares problem with ℓ_1-regularized regressors, called *Lasso*. Sparsity and compressed sensing have been widely used in signal processing, machine learning and control applications, especially in the big-data regime and noisy environments [52, 123]. In high-dimensional problems, it is likely that only a subset of features affects the observations. Hence, pursuing sparse representations reduces the model complexity, prevents overfitting and helps with overall interpretation. Since finding the solution with the minimal number of non-zero coefficients is NP-hard, ℓ_1-regularization has been proposed as a convex method that promotes sparsity. Perhaps one of the most used algorithms for sparse recovery has been the celebrated Lasso algorithm (least absolute shrinkage and selection operator), which accounts for both sparsity and potentially noisy data, using ℓ_1-regularization. For instance, Lasso has been used in signal reconstruction for medical imaging, wireless communication and tracking; seismology applications; portfolio optimization; text analysis [123, 236]. Many of these applications are large-scale or involve data coming from multiple data sources. With the recent widespread availability and development of cloud services, it seems an attractive and cost effective solution to outsource the computations to the cloud, when the data owner or querier lacks the computational resources and/or expertise to locally perform them. Given the privacy-sensitive nature of medical data, financial data, location data, energy measurements etc., on which such problems are computed, and how they can be used to profile users or mount attacks on critical infrastructure, the computations should not be performed in the clear at the cloud service.

The main challenges are the non-smoothness of the ℓ_1-norm, which is difficult to evaluate on encrypted data, as well as the iterative nature of the Lasso problem. We use a distributed ADMM formulation that enables us to exchange substantial local computation for little communication between multiple servers. We give an encrypted multi-party protocol for solving the distributed Lasso problem, by approximating the non-smooth part with a Chebyshev polynomial, evaluating it on encrypted data, and using a more cost effective distributed bootstrapping operation. Finally, we provide numerical results for our proposed solutions.

Compared to the first part of the chapter (Section 3.2), in the second part (Section 3.3), we consider the approximation of a non-smooth nonlinear objective function and use multiple servers to streamline the complex iterative computation, rather than a two-party computation, and a more powerful homomorphic encryption scheme to partially replace the blinded communication necessary at every iteration.

This chapter covers the work presented in [10, 13, 16].

Contributions

In the first part of the chapter, we develop a new tractable optimization protocol to privately solve constrained quadratic problems. Our protocol relies on cryptographic tools called encryption schemes. To solve the optimization problem on encrypted data, we use an additively homomorphic encryption scheme, where, in short, *addition commutes with the encryption function.* Thus, a party can process encrypted data without having direct access to the data. The novelty is how to handle in a privacy-preserving manner the constraints of the problem that introduce nonlinearities which cannot be supported by additively homomorphic encryption schemes. We show that a projected gradient method which operates on the Lagrange dual problem can alleviate this problem and can be efficiently run on encrypted data by exploiting communication between the participating parties.

The main contributions of our work are the following:

- We design and formally prove computational security guarantees for such protocols. The proof relies on applying cryptographic tools to the specific optimization algorithm that runs over multiple iterations.

- We investigate an alternative protocol which sacrifices some privacy but involves less communication overhead.

- We implement the protocols and show the computational and communication complexity produce reasonable running times.

Furthermore, we emphasize the difference between the computational security guarantee with the *non-unique retrieval* guarantee that is important in such optimization problems.

In the second part of this chapter, we use a more powerful homomorphic encryption scheme, which enables polynomial (rather than only affine) computations by the cloud over encrypted data of the client. However, encrypted Lasso brings new challenges: evaluating non-smooth functions on encrypted data, as well as continuing computations over multiple iterations and time steps, which generally requires refreshing the ciphertexts.

A conventional observation is that distributing a large optimization problem to multiple servers improves the execution time by parallelizing smaller subproblems. Apart from this, we note that *distributing the computation allows a streamlined execution of encrypted iterations.* In particular, using multiple servers allows us to perform a refresh operation at a substantially reduced cost compared to performing it only at one server. This cheaper refresh operation enables us to continue the encrypted computations over multiple iterations, as well as to use a high degree polynomial to approximate the gradient of the ℓ_1-norm. Specifically, we propose:

- an efficient distributed encrypted solution to Lasso problems using ADMM, offering computational privacy of all the data, including intermediate results;

- an optimized implementation of the above protocol using an efficient Chebyshev series evaluation for polynomial approximations and reducing the number of ciphertext levels and operations.

We also note that the quadratic program outlined in the first part of the chapter can be formulated in a similar fashion, by having the indicator function of the feasible space in the objective function. Hence, the solution we give here gives a different solution (in a different setup of the cloud) to the above problem.

Related work

Unconstrained optimization problems are commonly used in machine learning applications, e.g., for linear regression, and several works have addressed gradient methods with partially homomorphic encryption [120, 122]. However, adding constraints substantially complicates the optimization problem by restricting the feasible domain. In addition, constraints introduce nonlinear operations in the optimization algorithm, which cannot be handled inherently by the partially homomorphic cryptosystems. Examples of works that tackle private linear programming problems or machine learning applications with optimization subroutines using garbled circuits, secret sharing, and hybrid approaches between the above are [45, 58, 99, 148, 167] and the references within. Out of these, we inspire our solution from [45], which tackles linear problems in machine learning, assembling blocks for secure computation of the $\arg\max$ of multiple values and dot products, using additively and leveled homomorphic encryption.

The problem of secure constrained quadratic optimization with additively homomorphic encryption was addressed in [203], but the authors only guaranteed conditional privacy and non-unique retrieval of the private data. Specifically, in their case, in order to project the variable on the feasible space, the cloud multiplicatively blinds the encrypted variable and sends it to the requester, which decrypts the message, compares it to zero, and sends back the result of the comparison. This reveals more information than strictly necessary, and in special cases, can be used by the requester to infer information about the data of the agents.

The usage of ADMM for private distributed optimization is not novel, see e.g., [231, 234], given its convenient formulation and splitting of the objective function and variables. (For other distributed gradient based methods, see e.g., [155].) However, our usage of distributed ADMM substantially differs from previous works in the following: i) we start with centralized rather than already distributed data, so we split the centralized problem in a way that fits our privacy and low-power requirements; ii) we assume heterogeneous servers and we split the computations differently depending on who performs them; iii) the data at each server is not in the clear, which complicates the computations; iv) the servers do not learn any of the data, not even intermediate iterates and results; this requires more complex computations to privately perform nonlinear operations; v) the ℓ_1-regularization term is non-smooth and has nonlinear gradient, leading to updates of the global primal variable that are incompatible with the mentioned ADMM works.

The authors in [237] propose a distributed ADMM for a Lasso problem, using an threshold additively homomorphic encryption and SPDZ [79] for computing the nonlinear operations. In contrast to their work, we do not distribute the data in the clear to the computing servers, meaning we have less flexibility with respect to the local computations. However, we can choose to split the data in the most convenient way to have a distributed convergence speed similar to the centralized convergence, which [237] does not discuss. Another difference is that the tools they use require them to communicate for every nonlinear computation, such as multiplications and comparison operations, which require a number of communication rounds dependent on the number of bits in the messages. In our case, the servers only send two messages per iteration and the method we employ also allows us to batch vectors and perform operations in parallel for all elements of a vector.

In [96, 197], the authors propose distributed/federated training and evaluation with multi-party fully homomorphic encryption for linear, logistic and neural network models, using stochastic gradient descent. While we inspired our solution from the multi-party fully homomorphic encryption tool, their setup is different from ours: the data is either distributed in cleartext locally at the servers or other data providers perform the preprocessing; and the computations are different, leading to different optimizations: e.g., [96] uses a combination of distributed and centralized bootstrapping operations, [197] has only one recurring variable (the model) to bootstrap.

Finally, examples such as [121, 126, 176, 235] make use of differential privacy techniques in optimization algorithms. Other lines of work solve optimization problems with ad-hoc methods, such as transformation approaches, e.g. [97, 195, 214, 225]. These methods are out of the scope of this chapter, but differential privacy is an avenue for future research in privately determining parameters of the optimization problem.

3.2 Secure evaluation of quadratic programs

Organization. We formulate the problem in Section 3.2.1. In Section 3.2.3, we describe the optimization theory behind the proposed protocol and justify the choice of the gradient method. Furthermore, we present the main protocol and the subroutines that compose it and in Section 3.2.3.6 that use an additively homomorphic scheme and a masking procedure, and we show that the protocol achieves privacy of the agent's data and of the requester's result. We discuss possible relaxations of the security guarantees and investigate a more efficient protocol under these weaker conditions in Section 3.2.4.1. In Section 3.2.4, we provide a privacy analysis of the problem concerning the input-output relation. We present the complexity analysis of the protocols and show that the experimental results point out reasonable running times in Section 3.2.5. Finally, the detailed privacy proofs are given in Appendix B.

3.2.1 Problem setup

3.2.1.1 Motivating Examples

Quadratic optimization is a class of problems frequently employed in control system design and operation. As a first motivating example, consider the problem of estimating the state of a dynamical system from privacy-sensitive sensor measurements. For instance, such a problem arises in smart houses where the temperature or energy readings of the sensors are aggregated by a cloud controller and can reveal whether the owners are in the building. In particular, let the system dynamics and sensor measurements be described by:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{w}_t, \qquad \mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{v}_t, \tag{3.2.1}$$

for $t = 0, \dots, T-1$, where $\mathbf{w}_t, \mathbf{v}_t$ are process and measurement noise respectively. The system and sensor parameters $\mathbf{A} \in \mathbb{R}^{n\times n}, \mathbf{C} \in \mathbb{R}^{p\times n}$ can be thought as publicly available information while the sensor measurements $\mathbf{y}_0, \dots, \mathbf{y}_{T-1}$ are privacy-sensitive data. The untrusted cloud has to collect the measurements and output an initial state estimate $\mathbf{x}_0$ of the system to a target agent, while maintaining the privacy of the sensor data and final output. A simple state estimate may be found as the solution to the least squares problem:

$$\begin{aligned}&\min_{x_0\in\mathbb{R}^n}\ \frac{1}{2}\sum_{t=0}^{\intercal}\|\mathbf{y}_t-\mathbf{C}\mathbf{A}^\intercal\,\mathbf{x}_0\|_2^2=\frac{1}{2}\|\mathbf{y}-\mathcal{O}\,\mathbf{x}_0\|_2^2,\\ &\text{where}\quad \mathbf{y}=\begin{bmatrix}\mathbf{y}_0^\intercal & \mathbf{y}_1^\intercal & \dots & \mathbf{y}_{T-1}^\intercal\end{bmatrix}^\intercal,\ \mathcal{O}=\begin{bmatrix}\mathbf{C}^\intercal & (\mathbf{C}\mathbf{A})^\intercal & \dots & (\mathbf{C}\mathbf{A}^{T-1})^\intercal\end{bmatrix}^\intercal.\end{aligned}\tag{3.2.2}$$

$\mathcal{O}\in\mathbb{R}^{Tp\times n}$ is the system observability matrix. More general state estimation problems may also include constraints, e.g., the initial state may lie in a polyhedral set $\mathbf{D}\mathbf{x}_0\preceq\mathbf{b}$ where the shape of the polyhedron captured by the matrix $\mathbf{D}$ is publicly known but its position and size captured by the vector $\mathbf{b}$ is private information.

As a second example, consider a control problem with privacy-sensitive objectives. Suppose we are interested in steering a dynamical system (to track a private reference) starting from a private initial position while guaranteeing safety state and/or input constraints for a time horizon. The need for privacy in such problems arises when exploring vehicles are deployed in uncertain or hazardous environments or when different users compete against each other and want to hide their tactics. The appropriate controller for this problem is a model predictive controller, a type of receding horizon control that minimizes a cost on the deviation of states, measurements and control inputs from the desired references, under safety or saturation constraints. We thoroughly investigate the secure evaluation of model predictive controllers for linear systems in Chapter 6, using the solution from this chapter.

3.2.1.2 Problem statement

The above examples can be modeled as constrained quadratic optimization problems with distributed private data. We consider three types of parties involved in the problem: a number of agents $\mathcal{A}_i$, $i=1,\dots,p$, a cloud server $\mathcal{C}$ and a requester $\mathcal{R}$. The purpose of this setup is to solve an optimization problem with the data provided by the agents and the computation performed on the cloud and send the result to the requester. The architecture is presented in Figure 3.1.

Let us consider a strictly-convex quadratic optimization problem, which we assume to be feasible (there is at least one point where all constraints are satisfied):

$$
\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min}\ \frac{1}{2}\mathbf{x}^\intercal\,\mathbf{Q}_\mathcal{C}\,\mathbf{x} + \mathbf{c}_\mathcal{A}^\intercal\mathbf{x} \\
&\quad\text{s.t. } \mathbf{A}_\mathcal{C}\,\mathbf{x} \preceq \mathbf{b}_\mathcal{A},
\end{aligned}
\tag{3.2.3}
$$

where the variables and the parties to which they belong to are described as follows:

Agents $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_p)$: The agents are parties with low computational capabilities that possess the private information $\mathbf{b}_\mathcal{A}$ and $\mathbf{c}_\mathcal{A}$. The private information is decomposed across the agents as: $\mathbf{b}_\mathcal{A} = (\mathbf{b}_1, \ldots, \mathbf{b}_p)$ and $\mathbf{c}_\mathcal{A} = (\mathbf{c}_1, \ldots, \mathbf{c}_p)$, with $\mathbf{b}_i \in \mathbb{R}^{m_i}$ and $\mathbf{c}_i \in \mathbb{R}^{n_i}$ being the private data of agent i such that $\sum_{i=1}^p m_i = m$ and $\sum_{i=1}^p n_i = n$.

Cloud $\mathcal{C}$: The cloud is a party with high computational capabilities that has access to the matrices $\mathbf{Q}_\mathcal{C} \in \mathbb{S}^n_{++}$ and $\mathbf{A}_\mathcal{C} \in \mathbb{R}^{m\times n}$. When the computation is sophisticated and/or involves proprietary algorithms, $\mathbf{Q}_\mathcal{C}$ and $\mathbf{A}_\mathcal{C}$ are private data of the cloud; otherwise, $\mathbf{Q}_\mathcal{C}$ or $\mathbf{A}_\mathcal{C}$ are public.

Requester $\mathcal{R}$: The requester is a party with more computational capabilities than the agents that is interested in the optimal solution $\mathbf{x}^*$ of the problem. The requester can be either one of the data providers or a separate cloud server.

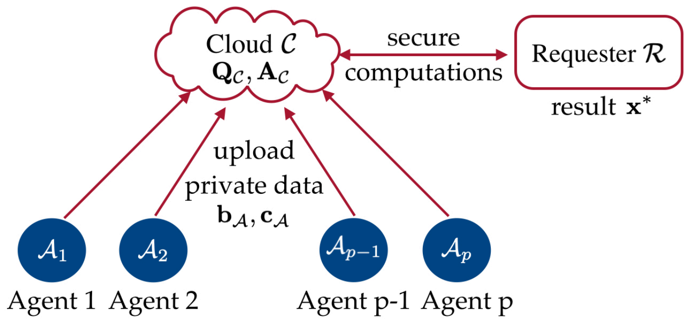


Figure 3.1: Architecture of the problem: Agents are low-resource parties that have private data that they outsource to a powerful server, called the cloud. The cloud has to solve an optimization problem on the private data of the agents and send the result to a party called the requester, which will help with the computations.

Note that in the first motivating example (3.2.2), the matrix $\mathbf{Q}_\mathcal{C}$ in (3.2.3) corresponds to the publicly known matrix $\mathcal{O}^\intercal\mathcal{O}$ and the vector $\mathbf{c}_\mathcal{A}$ corresponds to the private vector $-\mathcal{O}^\intercal\,\mathbf{y}$. For the objective to be strongly convex, we require that $\operatorname{rank}(\mathcal{O}) = n$, which also implies standard system observability. In the second example, the matrix $\mathbf{Q}_{\mathcal{C}}$ is composed from the regulating cost matrices, the cost vector $\mathbf{c}_{\mathcal{A}}$ is formed from the private initial conditions and steady-state solution for the reference tracking, mixed by the system's dynamics, and the constraints vector $\mathbf{b}_{\mathcal{A}}$ depends on the private state bounds and initial condition (see Chapter 6, equation (6.2.3)).

In most cloud applications, the service provider has to deliver the contracted service or otherwise the clients would switch to another service provider. This means that the cloud cannot alter the data it receives. Moreover, the clients' interest is to obtain the correct result from the service they pay for, hence, the agents and target node will also not alter the data. However, the parties can locally process the data they receive in any fashion they want. This means that the adversarial model we consider is semi-honest, defined in Definition 2.1.1.

The purpose of this section is to solve Problem (3.2.3) using a secure multiparty computation protocol for semi-honest parties. This protocol takes as inputs the private data of the agents, as well as the cloud's data, and involves the parties in exchanging messages and participating in some specified computations, and eventually outputs to the requester the solution of the optimization problem. This protocol should guarantee *computational security*, cf. Definition 2.2.6. More specifically, the cloud cannot obtain any information about the private inputs of the agents and the output of the requester and similarly, the requester cannot obtain any information about the private inputs of the agents and the cloud, other than what they can compute using their inputs and outputs and public information, by running a computationally efficient algorithm after the execution of the protocol.

In this section, we use the popular Paillier encryption, described in Preamble 2.5.1, but any other additively homomorphic encryption scheme can be employed. In our protocol, the requester $\mathcal{R}$ is the owner of a pair of Paillier keys. For everything that follows, $[[\cdot]]$ denotes the encryption with the requester's public key $\mathrm{pk}_{\mathcal{R}}$. In addition, we also use an additive blinding scheme, as described in Preamble 2.4, for reasons that will become apparent in the protocol, related to hiding the data from the requester. In Remark 2.5.4, we showed how this additive blinding is compatible with the AHE scheme, and combined, they offer a double encryption of the agents' data.

3.2.2 A simpler problem: secure unconstrained quadratic optimization

Consider the following example of an unconstrained optimization problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\intercal \mathbf{Q}_{\mathcal{C}}\mathbf{x} + \mathbf{c}_{\mathcal{A}}^\intercal \mathbf{x}, \tag{3.2.4}$$

where the variables have the same meaning as in Problem (3.2.3) described in Section 3.2.1. The cloud has access to the positive definite matrix $\mathbf{Q}_{\mathcal{C}}$. Notice that *ridge regression* problems $\min\limits_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{Qx}-\mathbf{c}\|_2^2 + \lambda\|\mathbf{x}\|_2^2$, for some appropriately sized quantities $\mathbf{Q},\mathbf{c}$ and penalty λ can be cast in the same form as (3.2.4).

Problem (3.2.4) can be solved by setting the optimal solution equal to value that zeroes the gradient of the objective function:

$$\nabla f(\mathbf{x}^*) = 0 \rightarrow \mathbf{x}^* = \mathbf{Q}_{\mathcal{C}}^{-1}\mathbf{c}_{\mathcal{A}}. \tag{3.2.5}$$

Alternatively, when $\mathbf{Q}_{\mathcal{C}}$ is very large or its inverse is numerically unstable, we prefer to use an iterative algorithm to reach to the optimal solution, namely, the gradient descent algorithm. An iteration of the gradient method has the form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\nabla f(\mathbf{x}_k) = (\mathbf{I} - \eta\mathbf{Q}_{\mathcal{C}})\mathbf{x}_k - \eta\mathbf{c}_{\mathcal{A}}, \tag{3.2.6}$$

where $\eta > 0$ is the step-size chosen by the cloud and $k = 0,\ldots,K-1$ for a number of iterations K. While usually we would check convergence by checking the difference between the objective function evaluated at consecutive iterations and would stop if this difference is smaller than some tolerance, this can leak information about the private data (it is problem dependent whether this leakage is significant or not). Therefore, for privacy reasons, K needs to be chosen large enough such that convergence is guaranteed with high probability and the algorithm should be run for this fixed number of iterations.

It is important to notice that *for a quadratic objective function, only linear operations* *in the private data* $\mathbf{c}_{\mathcal{A}}$ *and* $\{\mathbf{x}_k\}_{k=0,\ldots,K-1}$ *are required in order to compute the solution*, both in its closed-form (3.2.5) and in its iterative form (3.2.6). Hence, by having the agents encrypt their data with an additively homomorphic encryption scheme (e.g., the Paillier scheme described in Preamble 2.5.1) and taking advantage of the additively homomorphic property (2.5.3), the cloud can locally compute the optimal solution in its closed-form in the encrypted domain:

$$[[\mathbf{x}^*]] = \mathbf{Q}_{\mathcal{C}}^{-1} \otimes [[\mathbf{c}_{\mathcal{A}}]], \tag{3.2.7}$$

or by the gradient descent algorithm in the encrypted domain, for all $k = 0, \ldots, K-1$:

$$[[\mathbf{x}_{k+1}]] = (\mathbf{I} - \eta \mathbf{Q}_{\mathcal{C}}) \otimes [[\mathbf{x}_k]] \oplus (-\eta) \otimes [[\mathbf{c}_{\mathcal{A}}]], \tag{3.2.8}$$

and send the final point $[[\mathbf{x}^*]] = [[\mathbf{x}_K]]$ to the requester to decrypt and obtain the desired result. Such a protocol satisfies the desired security according to Definition 2.2.6 provided in Preamble 2.2 by the fact that the cloud only has access to data encrypted by a semantically secure encryption scheme.

We mention also that quadratic problems with linear *equality* constraints can be solved using the same tools presented above, assuming the invertibility of the matrix in (3.2.10):

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = {} & \frac{1}{2}\mathbf{x}^{\intercal}\mathbf{Q}_{\mathcal{C}}\mathbf{x} + \mathbf{c}_{\mathcal{A}}^{\intercal}\mathbf{x}, \\ \text{s.t. } \mathbf{A}_{\mathcal{C}}\mathbf{x} = {} & \mathbf{b}_{\mathcal{A}}. \end{aligned} \tag{3.2.9}$$

The closed-form optimal solution for (3.2.9) can be written as:

$$\begin{bmatrix} \mathbf{Q}_{\mathcal{C}} & \mathbf{A}_{\mathcal{C}}^{\intercal} \\ \mathbf{A}_{\mathcal{C}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_{\mathcal{A}} \\ \mathbf{b}_{\mathcal{A}} \end{bmatrix}, \tag{3.2.10}$$

for $\boldsymbol{\nu}$ a Lagrange multiplier, which has a unique solution if $\mathbf{Q}_{\mathcal{C}}^{-1}$ and $(\mathbf{A}_{\mathcal{C}}\mathbf{Q}_{\mathcal{C}}^{-1}\mathbf{A}_{\mathcal{C}})^{-1}$ are invertible. As before, one can write an iterative algorithm to obtain the optimal solution $\mathbf{x}^*$ without explicitly inverting the matrices.

One needs to be careful with what kind of privacy is expected from the above kind of problem. In a trivial example, if $\mathbf{A}_{\mathcal{C}} = \mathbf{I}$, then upon receiving the solution $\mathbf{x}^*$, the requester immediately learns the private data $\mathbf{b}_{\mathcal{A}}$. The kind of input-output cryptographic privacy covers this trivial example as well, stating that nothing *apart from* the knowledge inferred from the input and output is leaked by a secure protocol.

Encryption schemes are usually defined on finite fields or finite groups, while in practice, data involved in optimization problems have real values. To this end, we need to employ a lossy mapping to map real values to a finite set. One common approach is to use fixed-point representation to map from reals to a finite group of integers: truncate the real values to a fixed number of integer bits and fractional bits and then multiply by a scaling factor, a large enough integer value such that the result is an integer in the appropriate range. A subtlety that arises when evaluating iterative algorithms on encrypted data, when the underlying data uses fixed-point representation and is not originally an integer, is overflow. Truncation is not possible over (additively homomorphic) ciphertexts and dealing with consecutive multiplications implies the accumulation of the scaling factor which leads to overflow: the underlying plaintext in the resulting ciphertext will be be outside of the finite group and decryption will become incorrect. The same problems can happen with sequential additions, but the problem is clearly not as stringent as with sequential multiplications. In the literature, this is addressed by using, where possible, similarity transformations to integer matrices [65] or reset the iterate accordingly [170]. In this chapter, we avoid the overflow problem by exploiting the communication between parties (necessary in order to achieve more complex operations on encrypted data) in order to privately perform truncation.

3.2.3 Secure constrained quadratic optimization

In this section we introduce our main algorithmic results for securely solving a quadratic optimization problem with linear inequality constraints, which introduce nonlinear operations in the optimization algorithm and cannot be handled inherently by an AHE scheme. Let us first describe the optimization algorithm used for solving the minimization problem (3.2.3) on unencrypted data.

For strongly convex problems, one can resort to duality theory [47, Ch. 5] to compute the projection on the feasible set, and be able to retrieve the optimal value of the primal problem from the optimal value of the dual problem. For the quadratic optimization problem (3.2.3), its dual is also a quadratic optimization problem:

$$\begin{aligned} \boldsymbol{\mu}^* = \underset{\boldsymbol{\mu}\in\mathbb{R}^m}{\arg\max}\ & -\frac{1}{2}(\mathbf{A}_\mathcal{C}^\intercal\boldsymbol{\mu} + \mathbf{c}_\mathcal{A})^\intercal\mathbf{Q}_\mathcal{C}^{-1}(\mathbf{A}_\mathcal{C}^\intercal\boldsymbol{\mu} + \mathbf{c}_\mathcal{A}) - \boldsymbol{\mu}^\intercal\mathbf{b}_\mathcal{A} \\ & \text{s.t. } \boldsymbol{\mu} \succeq \mathbf{0}. \end{aligned} \tag{3.2.11}$$

The dual objective function is denoted by $g(\boldsymbol{\mu})$ and its gradient is equal to:

$$\nabla g(\boldsymbol{\mu}) = -\mathbf{A}_\mathcal{C}\mathbf{Q}_\mathcal{C}^{-1}(\mathbf{A}_\mathcal{C}^\intercal\boldsymbol{\mu} + \mathbf{c}_\mathcal{A}) - \mathbf{b}_\mathcal{A}. \tag{3.2.12}$$

Under standard constraint qualifications, e.g., Slater's condition [47, Ch. 5], strong duality between the primal and dual holds, which means the optimal objective in the primal problem (3.2.3) is equal to the objective in the dual problem (3.2.11). Moreover, the optimality conditions (Karush-Kuhn-Tucker) hold and are the following:

$$\mathbf{Q}_\mathcal{C}\mathbf{x}^* + \mathbf{A}_\mathcal{C}^\intercal\boldsymbol{\mu}^* + \mathbf{c}_\mathcal{A} = \mathbf{0} \tag{3.2.13}$$

$$\mathbf{A}_\mathcal{C}\mathbf{x}^* - \mathbf{b}_\mathcal{A} \preceq \mathbf{0}, \quad \boldsymbol{\mu}^* \succeq \mathbf{0} \tag{3.2.14}$$

$$\mu_i^*(\mathbf{a}_i^\intercal\mathbf{x}^* - b_i) = 0,\ i = 1, \dots, m. \tag{3.2.15}$$

For strictly convex problems, i.e., $\mathbf{Q}_\mathcal{C} \in \mathbb{S}_{++}^n$, the optimal solution of the primal problem can be obtained from (3.2.13) as

$$\mathbf{x}^* = -\mathbf{Q}_\mathcal{C}^{-1}(\mathbf{A}_\mathcal{C}^\intercal\boldsymbol{\mu}^* + \mathbf{c}_\mathcal{A}). \tag{3.2.16}$$

An algorithm for computing the optimum in problem (3.2.11), which we will show is also compatible with the AHE scheme, is the projected gradient ascent method. The projected gradient ascent is composed by iterations of the following type:

$$\boldsymbol{\mu}_{k+1} = \max\{\mathbf{0}, \boldsymbol{\mu}_k + \eta \nabla g(\boldsymbol{\mu}_k)\}, \tag{3.2.17}$$

where $\eta > 0$ is the step size and $\boldsymbol{\mu}_{k+1}$ is the projected value of $\boldsymbol{\mu}_k + \eta \nabla g(\boldsymbol{\mu}_k)$ over the non-negative orthant. For full rank of $\mathbf{A}_\mathcal{C}\mathbf{Q}_\mathcal{C}^{-1}\mathbf{A}_\mathcal{C}^\intercal$, the dual problem is strictly convex and the algorithm converges with a linear rate [173] for a fixed step size $\eta = \frac{1}{L}$, where $L = \lambda_{max}(\mathbf{A}_\mathcal{C}\mathbf{Q}_\mathcal{C}^{-1}\mathbf{A}_\mathcal{C}^\intercal)$. For non-strictly convex dual function, the gradient ascent algorithm converges in sublinear time [173].

3.2.3.1 Projected gradient ascent on encrypted data

As stated in Section 3.2.1, we aim to solve an optimization problem outsourced to the cloud on private distributed data from the agents and send the result to the requester. To protect the agents' data, we use an encryption scheme that allows the cloud to perform linear manipulations on encrypted data, as described in Preamble 2.5.1. To this end, the requester generates a pair of keys $(\mathrm{pk}_\mathcal{R}, \mathrm{sk}_\mathcal{R})$ and distributes the public key to the agents and the cloud, enabling them to encrypt their data, which only the requester will be able to decrypt, using the private key. We consider that all the data is represented on integers of l bits and comment on this further in Section 3.2.3.5.

The main difficulty in performing the projected gradient ascent on encrypted data is performing iteration (3.2.17). We have already seen in the example in Section 3.2.2 that the update of the iterate in the direction of the gradient ascent can be computed locally by the cloud directly on the encrypted data (3.2.8). However a first challenge lies in performing the comparison with zero. Due to the probabilistic nature and modular arithmetic of the Paillier encryption scheme, *comparing ciphertexts does not give any information about the order between the underlying messages* and comparison on encrypted data cannot be performed locally by the cloud. Notwithstanding, we employ an interactive protocol between the cloud, which holds the encrypted data, and the requester, which is the owner of the private key for the Paillier cryptosystem, that achieves the comparison securely. Moreover, after the comparison is performed, the update of the encrypted iterate (3.2.17) has to be done in a private way, so that the result of the maximum operation is not revealed to any of the parties involved (the cloud and the requester). These two steps are the main computational bottleneck in the protocol we propose, as both require secure communication between the cloud and the requester.

We can privately achieve the two steps mentioned above in three stages. First, the cloud has to randomize the order of the two encrypted variables it wants to compare (Protocol 3.2.1). Second, the cloud and requester engage in an interactive comparison protocol that takes as inputs the two randomized variables and outputs the result of the comparison to the requester (Protocol 2.7.2 in Preamble 2.7). Third, the update of the dual iterate is achieved through an interactive protocol between the cloud and requester, which takes as inputs the two randomized variables and the result of the comparison and outputs to the cloud the updated iterate (Protocol 3.2.2 and Preamble 2.6). Throughout this section, by comparison we mean element-wise comparison, since the variable $\boldsymbol{\mu}$ is a vector.

3.2.3.2 Secure comparison protocol

In order to privately compute (3.2.17), i.e., hide the result from *all* the parties involved, we want to keep the result of the comparison of the updated iterate $\boldsymbol{\mu}_k + \nabla g(\boldsymbol{\mu}_k)$ with zero unknown to both the cloud and the requester. The comparison protocol will reveal the result of the comparison between the two inputs to the requester $\mathcal{R}$. However, if we introduce an additional step where $\mathcal{C}$ randomizes the order of the two values that it wants to compare, then $\mathcal{R}$ does not learn any information by knowing the result of the comparison.

PROTOCOL 3.2.1: Randomization step

Input: $\mathcal{C}$: $[[\bar{\boldsymbol{\mu}}]], [[\mathbf{0}]]$, where $\bar{\boldsymbol{\mu}} := \boldsymbol{\mu} + \eta\nabla g(\boldsymbol{\mu})$

Output: $\mathcal{C}$: $[[\mathbf{a}]], [[\mathbf{b}]]$

```
1: $\mathcal{C}$: choose a random permutation $\pi$ on two elements
2: $\mathcal{C}$: output $[[\mathbf{a}]], [[\mathbf{b}]] \leftarrow \pi([[\mathbf{0}]], [[\bar{\boldsymbol{\mu}}]])$, where the permutation is applied component-wise
```

The comparison protocol that will be used in our optimization protocol is as follows. Let $\mathcal{C}$ have two encrypted values under the Paillier scheme $[[a]]$ and $[[b]]$ that it obtained after running Protocol 3.2.1, and let $\mathcal{R}$ have the decryption key. Furthermore, let $\mathcal{R}$ also have the decryption key of the DGK homomorphic encryption scheme. At the end of the protocol (outlined in Protocol 2.7.2 in Preamble 2.7), $\mathcal{R}$ will have the result of the comparison in the form of one bit δ such that $(\delta = 1) \Leftrightarrow (a \leq b)$. Let l denote the number of bits of the unencrypted inputs a, b.

We will use blinding by random numbers to secure the communication from the cloud to the requester. To guarantee the security of the DGK scheme, we choose a key-size that makes factoring hard and set the randomization in the encryption primitive to be of length greater than $2t$ bits, for $t = 160$. See more details in [77, 78].

3.2.3.3 Secure update protocol

Moreover, we need to ensure that when the cloud $\mathcal{C}$ updates the value of the dual iterate at iteration $k+1$ in equation (3.2.17), it does not know the new value. The solution is to make the cloud blind the values of $[[\mathbf{a}]]$ and $[[\mathbf{b}]]$ and send them to the requester in this order, where the latter selects the value accordingly to the comparison result and then sends it back to the cloud. However, there are two important issues that have to be addressed in order for the update step to not leak information about the sign of the iterate: the blinding should be additive and effectuated with different random values, and the ciphertexts should be refreshed. The reasons are the following: if the blinding is multiplicative, by decrypting the product, the requester knows which one of the values is zero. Moreover, if the two values are additively blinded with the same random value, the requester can subtract them and reveal at least if the value is zero. Re-randomization of the encryptions is necessary so that the cloud cannot simply compare $[[\mathbf{a}]]$ and $[[\mathbf{b}]]$ with the received value. This can be done by adding an encryption of zero or by decryption followed by encryption. Protocol 3.2.2 is the solution to the update problem. In particular, it is the oblivious transfer procedure OT' in Preamble 2.6 applied element-wise for vectors $[[\mathbf{a}]], [[\mathbf{b}]]$ and selection bits $\boldsymbol{\delta}$.

PROTOCOL 3.2.2: Secure update of the dual variable

```
Input: $\mathcal{C}$: $[[\mathbf{a}]], [[\mathbf{b}]]$; $\mathcal{R}$: $\boldsymbol{\delta}$ such that $(\delta_i = 1) \Leftrightarrow (a_i \leq b_i)$
Output: $\mathcal{C}$: $[[\boldsymbol{\mu}]]$
 1: for each $i = 1, \dots, m$ in parallel do
 2:     $\mathcal{C}$: choose two random numbers $r, s$
 3:     $\mathcal{C}$: $[[\bar{a}]] \leftarrow [[a_i]] \oplus [[r]], [[\bar{b}]] \leftarrow [[b_i]] \oplus [[s]]$
 4:     $\mathcal{C}$: send $[[\bar{a}]]$ and $[[\bar{b}]]$ to $\mathcal{R}$
 5:     if $\delta_i = 0$ then $\mathcal{R}$: $[[v]] \leftarrow \widetilde{[[\bar{a}]]} = [[\bar{a}]] + [[0]]$
 6:     else $\mathcal{R}$: $[[v]] \leftarrow \widetilde{[[\bar{b}]]} = [[\bar{b}]] + [[0]]$        ▷ Refresh the ciphertext
 7:     end if
 8:     $\mathcal{R}$: send $[[v]]$ and $[[\delta_i]]$ to $\mathcal{C}$
 9:     $\mathcal{C}$: $[[\mu_i]] \leftarrow [[v]] \oplus r \otimes [[\delta_i]] \oplus [[-r]] \oplus (-s) \otimes [[t]]$        ▷ $\mu_i \leftarrow v + r(\delta_i - 1) - s\delta_i$
10: end for
```

3.2.3.4 Protocol for solving strictly-convex quadratic problems

Having defined these protocols, we can now build a protocol that represents one iteration (3.2.17) of the dual projected gradient ascent method.

PROTOCOL 3.2.3: Secure iteration of the dual projected gradient ascent method

```
Input: $\mathcal{C}$: $\mathbf{A}_\mathcal{C} \in \mathbb{R}^{m \times n}, \mathbf{Q}_\mathcal{C} \in \mathbb{S}^n_{++}, [[\mathbf{b}_\mathcal{A}]], [[\mathbf{c}_\mathcal{A}]], \eta > 0, [[\boldsymbol{\mu}_k]]$; $\mathcal{R}$: $\mathrm{sk}_\mathcal{R}$
Output: $\mathcal{C}$: $[[\boldsymbol{\mu}_{k+1}]]$
1: $\mathcal{C}$: $[[\nabla g(\boldsymbol{\mu}_k)]] \leftarrow (-\mathbf{A}_\mathcal{C}\mathbf{Q}_\mathcal{C}^{-1}\mathbf{A}_\mathcal{C}^\intercal) \otimes [[\boldsymbol{\mu}_k]] \oplus (-\mathbf{A}_\mathcal{C}\mathbf{Q}_\mathcal{C}^{-1}) \otimes [[\mathbf{c}_\mathcal{A}]] \oplus (-1) \otimes [[\mathbf{b}_\mathcal{A}]]$    ▷ Compute the
   encrypted gradient as in (3.2.12)
2: $\mathcal{C}$: $[[\bar{\boldsymbol{\mu}}_k]] \leftarrow [[\boldsymbol{\mu}_k]] \oplus \eta \otimes [[\nabla g(\boldsymbol{\mu}_k)]]$        ▷ Update the value in the ascent direction
3: $\mathcal{C}, \mathcal{R}$ truncate $[[\bar{\boldsymbol{\mu}}_k]]$ to $l$ bits
4: $\mathcal{C}$ execute Protocol 3.2.1: $\mathcal{C}$ gets $[[\mathbf{a}_k]], [[\mathbf{b}_k]]$        ▷ Randomly assign $[[\bar{\boldsymbol{\mu}}_k]], [[\mathbf{0}]]$ to $[[\mathbf{a}_k]], [[\mathbf{b}_k]]$
5: $\mathcal{C}, \mathcal{R}$ execute Protocol 2.7.2 element-wise on inputs $[[\mathbf{a}_k]], [[\mathbf{b}_k]]$: $\mathcal{R}$ gets $\boldsymbol{\delta}_k$ ▷ Secure comparison
6: $\mathcal{C}, \mathcal{R}$ execute Protocol 3.2.2: $\mathcal{C}$ obtains $[[\boldsymbol{\mu}_{k+1}]]$    ▷ Secure update ensuring $\boldsymbol{\mu}_{k+1} = \max\{\bar{\boldsymbol{\mu}}_k, \mathbf{0}\}$
```

Line 3 ensures that the updated iterate has the required number of bits for the comparison protocol. This step is achieved by an exchange between the cloud and requester: the cloud additively blinds the iterate by a random number, sends it to the requester, which decrypts and truncates the sum and sends it back, where the cloud then subtracts the truncated random number.

The random numbers used for blinding the sensitive values (in Protocols 2.7.2 and 3.2.2) are sampled uniformly from the integers in $\mathbb{Z}_N$ of $l+\lambda$ bits, where λ is the statistical security parameter, chosen such that brute-forcing the solution is intractable. In order to guarantee correctness of the comparison protocol, no overflow must take place, so we must impose $log_2 N > l + \lambda + 1$.

The proof of security in the semi-honest model follows similar steps as in the argmax protocol in [45] and we will address it in Appendix B.1.

Proposition 3.2.1. *Protocol 3.2.3 is secure in the semi-honest model, cf. Definition 2.2.5.*

Using the building blocks described above, we can finally assemble the protocol that privately solves the constrained quadratic optimization problem (3.2.3) with private data and sends the optimal solution to the requester. The public key $\text{pk}_{\mathcal{R}}$ and bit-length l are known by all the parties, hence we omit them from the inputs.

Protocol 3.2.4: Privacy preserving algorithm for solving strictly-convex quadratic optimization problems

Input: $\mathcal{A}_{i=1,\dots,p}$: $\mathbf{b}_{\mathcal{A}} = \{b_j\}_{j=1,\dots,m}, \mathbf{c}_{\mathcal{A}} = \{c_j\}_{j=1,\dots,n}$; $\mathcal{C}$: $\mathbf{A}_{\mathcal{C}} \in \mathbb{R}^{m\times n}, \mathbf{Q}_{\mathcal{C}} \in \mathbb{S}^n_{++}, K$; $\mathcal{R}$: $\text{sk}_{\mathcal{R}}, K$

Output: $\mathcal{R}$: $\mathbf{x}^*$

```
1: for i=1,...,p do
2:     $\mathcal{A}_i$ : encrypt the private information $msg_i \leftarrow ([[\mathbf{b}_i]], [[\mathbf{c}_i]])$
3:     $\mathcal{A}_i$ : send the encrypted messages to $\mathcal{C}$
4: end for
5: $\mathcal{C}$: Construct the vectors $[[\mathbf{b}_{\mathcal{A}}]]$ and $[[\mathbf{c}_{\mathcal{A}}]]$ from the messages
6: $\mathcal{C}$: $\eta \leftarrow 1/\lambda_{max}(\mathbf{A}_{\mathcal{C}}\mathbf{Q}_{\mathcal{C}}^{-1}\mathbf{A}_{\mathcal{C}}^{\intercal})$
7: $\mathcal{C}$: Choose a random positive initial value $\boldsymbol{\mu}_0$ for the dual variable and encrypt it: $[[\boldsymbol{\mu}_0]]$
```

```
 8: for each $k = 0, \dots, K-1$ do
 9:     $\mathcal{C}, \mathcal{R}$ execute Protocol 3.2.3: $\mathcal{C}$ gets $[[\boldsymbol{\mu}_{k+1}]]$    ▷ $\mathcal{C}, \mathcal{R}$ securely effectuate an iteration of the dual projected gradient ascent
10: end for
11: $\mathcal{C}$: $[[\mathbf{x}^*]] \leftarrow (-\mathbf{Q}_\mathcal{C}^{-1}\mathbf{A}_\mathcal{C}^\intercal) \otimes [[\boldsymbol{\mu}_K]] \oplus (-\mathbf{Q}_\mathcal{C}^{-1}) \otimes [[\mathbf{c}_\mathcal{A}]]$ and send it to $\mathcal{R}$    ▷ Compute the primal optimum from the optimal dual solution as in (3.2.13)
12: $\mathcal{R}$: Decrypt $[[\mathbf{x}^*]]$ and output $\mathbf{x}^*$
```

3.2.3.5 Fixed-point arithmetic

The optimization problem (3.2.3) is defined on real variables, whereas the Paillier encryption scheme is defined on integers. To address this issue, we adopt a fixed-point arithmetic setting, where we allow for a number to have a fixed number of fractional bits. First, we consider numbers that have the magnitude between $-2^{l_i-1} < x < 2^{l_i-1}$. Second, we consider a value having l_i bits for the integer part and l_f bits for the fractional part. Therefore, by multiplying the real values by 2^{l_f} and truncating the result, we obtain integers. We choose $l = l_i + l_f$ large enough such that the loss in accuracy is negligible and assume that there is no overflow. For ease of exposition, we consider this data processing done implicitly in the protocols described.

The errors in the solution caused by the fixed-point arithmetic operations necessary for the encryption can be analyzed with the same tools as in [11, 129, 193]. The round-off errors can be regarded as states in a stable dynamical system with bounded disturbances, and hence, have a bounded norm that offers a guide on how to choose the number of fractional bits l_f for the fixed-point representation. On the other hand, the overflow and quantization errors depend on the magnitude of the dual iterates. We considered feasible and bounded problems–the dual problem (3.2.11) has a finite solution–therefore, one can select the number of integer bits l_i in the representation such that no overflow occurs.

3.2.3.6 Privacy of quadratic optimization protocol

We first provide the statements of the results and the main ideas of the proofs here, and follow-up with the detailed arguments in Appendix B.

Theorem 3.2.2. *Protocol 3.2.4 is secure cf. Definition 2.2.5 for non-colluding parties.*

The intuition for the proof is as follows. Consider an iteration of the gradient ascent in Protocol 3.2.4. Firstly, in the Paillier cryptosystem, two ciphertexts are computationally indistinguishable to a party that does not have access to the decryption key. Secondly, the exchanges between the cloud and the requester are additively blinded using a different random number uniformly sampled from a large enough range (at least λ bits more over the values that are desired to be blinded, where the size of λ is chosen appropriately, as discussed in Preamble 2.4. This means that the blinded message is statistically indistinguishable from a random number sampled from the same distribution. Thirdly, the ciphertexts are refreshed (a different encryption of the same value), after each exchange, so a party that does not have access to the decryption key cannot infer information about the encrypted values by simply comparing the ciphertexts. Then, none of the parties can infer the magnitude or the sign of the private variables. However, we need to show that privacy is not broken by running an iteration multiple times. We prove that storing the exchanged messages does not give any new information on the private data, using similar arguments. Formally, using Definition 2.2.5, we construct a probabilistic polynomial-time simulator that randomly generates messages from the inputs and outputs such that its view and the view of the adversary, on the same inputs, are computationally indistinguishable. The correctness of Protocol 3.2.4 is immediate and follows from the correctness of the dual gradient ascent algorithm and the correctness of the comparison protocol. The proof is given in Appendix B.1.

Let us now consider collusions between the parties in the setup and prove privacy of Protocol 3.2.4 under coalitions. Definition 2.2.6 states that even under collusions, the protocol securely computes a functionality, which in this case, is solving a quadratic optimization problem. No further information is revealed than what can inferred from the coalition's inputs and outputs. However, if all agents and the cloud collude then they have access to all the information to solve the problem, in which case the above result is rather vacuous. Similarly, if the cloud colludes with the requester, then it can gain access to all the private data of the agents. Hence, we consider coalitions that involve either all the agents or the cloud and the requester to be prohibited (and unrealistic) and only consider coalitions between a strict subset of the agents and the cloud or between a strict subset of the agents and the requester. We give a complementary analysis on the information gained from the inputs and outputs of coalitions in Section 3.2.4.

Theorem 3.2.3. *Protocol 3.2.4 is secure cf. Definition 2.2.6 against coalitions.*

The proof of Theorem 3.2.3 can be derived from the proof of Theorem 3.2.2 because the agents only contribute with their inputs to the view of the coalition, and not with new messages. The proof is given in Appendix B.2.

Remark 3.2.4. When the matrices $\mathbf{Q}_\mathcal{C}$ and $\mathbf{A}_\mathcal{C}$ are private, hence, sent to the cloud encrypted, an additively homomorphic encryption scheme does not suffice anymore. To account for this and give a private solution to problem (3.2.3), we can update Protocol (3.2.4) to use a level-1 somewhat homomorphic encryption scheme, like LabHE or leveled fully HE (see Preamble 2.5.2 and Preamble 2.5.3).

3.2.4 Privacy discussion

3.2.4.1 Alternative quadratic optimization protocol

As explained in Section 3.2.3.2, the major computational and communication overhead of the above protocol is the secure comparison protocol, required to project dual variables to non-negative numbers. In this section, we describe a computationally less involved alternative approach, which bypasses the need for the secure comparison protocol, at the expense of revealing more information. This approach is developed in more detail in [13, 203].

Specifically, consider a step of the dual gradient algorithm where the cloud maintains an unprojected gradient ascent step $\bar{\boldsymbol{\mu}}_k = \boldsymbol{\mu}_k - \eta[\mathbf{A}_\mathcal{C}\mathbf{Q}_\mathcal{C}^{-1}(\mathbf{A}_\mathcal{C}^\intercal\boldsymbol{\mu}_k + \mathbf{c}_\mathcal{A}) + \mathbf{b}_\mathcal{A}]$ encrypted using the public key of the requester. Suppose the cloud multiplies the elements in this vector with random scalar values r_k uniformly distributed over the positive integers and sends the products to the requester. The latter can decrypt the message using its private key and gain access not to the actual unprojected iterate but to the randomly scaled version of it, which reveals the sign. It can then project it to the non-negative orthant in an unencrypted fashion: $\max\{0, (r_k)_i(\bar{\mu}_k)_i\}$ for $i = 1, \ldots, m$, and finally encrypt the result using its public key and return it to the cloud. The cloud can divide the result with the previously selected values r_k to compute in an encrypted fashion the elements $[[(\mu_{k+1})_i]] = (1/(r_k)_i) \otimes [[\max\{0, (r_k)_i(\bar{\mu}_k)_i\}]]$ which is equivalent to the encrypted actual projected step $[[\max\{0, (\bar{\mu}_k)_i\}]]$ for $i = 1, \ldots, m$, because division with a positive number commutes with the max operator.

This alternative solution bypasses the complexity of the secure comparison part. On the other hand, it reveals more information to the requester than the secure comparison approach. In particular it reveals a scaled version of the unprojected dual variable which in turn does not reveal the magnitude of this value but reveals whether this is positive, negative, or zero.

Implementation-wise, we want the values of r_k to be large enough to blind the magnitude of the elements of $\bar{\mu}_k$, so we will randomly sample r_k from the values of $\gamma + l$ bits, where γ is the security size for the multiplicative blinding. Furthermore, we need to be able to represent $1/(r_k)_i$ on the message space and to have enough precision as to obtain the correct value of $(\mu_{k+1})_i$. Hence, we will perform implicit multiplications and divisions by $2^{l'_f}$, for $l'_f > l + \gamma$. This change shows that, although significantly less communication is required to perform the projection operation, all the other operations are more expensive, because we work with larger numbers.

3.2.4.2 Privacy analysis of input-output relation

In this section we discuss privacy aspects that differ from the computational notions of Definitions 2.2.5 and 2.2.6. Even if a protocol does not leak any information, the known information in a coalition (e.g., the output and some of the inputs) can be used by the coalition to infer the rest of the private inputs. More specifically, in our optimization problem, the private variables and the optimal solution are coupled via the optimality conditions (3.2.13)-(3.2.15), which are public knowledge, irrespective of the protocol, and may be used for the purpose described above.

Consider the following definition that concerns the retrieval of private data from adversarial/known data.

Definition 3.2.5. (Non-unique retrieval) Let p be the private inputs of a problem and let an algorithm $A(p)$ solve that problem. Let $\mathcal{K}$ be the adversarial knowledge of the problem, which can contain public information, some private information (including some parts of the input) and the output of algorithm A for the adversary, denoted by $A^{\mathcal{K}}(p)$. We say p cannot be uniquely retrieved by the adversary if there exists a set $\mathcal{U}$, such that $p \in \mathcal{U}$, $|\mathcal{U}| \geq 2$ and:

$$\forall p' \in \mathcal{U} : A^{\mathcal{K}}(p) = A^{\mathcal{K}}(p').$$

Definition 3.2.5 can be modified to be stronger by requiring the set $\mathcal{U}$ to have an infinite number of elements.

Meeting the requirements of Definition 2.2.5 and Definition 2.2.6 is equivalent to not revealing any other information than what is already known to any of the parties, i.e. inputs, prescribed outputs, if any, and previously known side information. This is a stronger requirement than guaranteeing that an adversary cannot uniquely retrieve the data of the honest parties, i.e. Definition 3.2.5. Revealing sensitive information does not always lead to a unique retrieval of the private data. Nevertheless, any piece of information revealed by the execution of the protocol, that cannot be obtained only from its inputs and outputs, leads to the violation of Definitions 2.2.5, 2.2.6, even if the private data cannot be singled out with this information.

In what follows, beyond the computational security analysis of the previous sections, we carry out an algebraic analysis on a black-box protocol that given the agent's private data $\mathbf{b}_{\mathcal{A}}, \mathbf{c}_{\mathcal{A}}$ and the cloud's matrices $\mathbf{Q}_{\mathcal{C}}, \mathbf{A}_{\mathcal{C}}$, outputs the solution $\mathbf{x}^*$ of Problem (3.2.3) to the requester. We provide conditions such that a coalition cannot uniquely determine unknown private inputs from the output and a set of inputs, in the sense of Definition 3.2.5. In particular, this analysis applies to Protocol 3.2.4 which, assuming it runs for sufficient iterations, outputs the desired result $\mathbf{x}^*$ to the requester. We perform this algebraic analysis in the space of real numbers, which can be further expanded to fixed-point arithmetics for large enough precision.

Suppose without loss of generality that a coalition between $\bar{p}$ agents $(1 \leq \bar{p} < p)$ has access to the elements $b_1, \ldots, b_{\bar{m}}$ with $0 \leq \bar{m} \leq m$, and $c_1, \ldots, c_{\bar{n}}$ with $0 \leq \bar{n} \leq n$. Then let us define the decomposition of the matrix $\mathbf{A}_{\mathcal{C}}$ as:

$$\mathbf{A}_{\mathcal{C}} = \left[\begin{array}{c|c} \multicolumn{2}{c}{\mathbf{A}_1} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} \end{array}\right] \tag{3.2.18}$$

where the matrices $\mathbf{A}_1 \in \mathbb{R}^{\bar{m} \times n}$, $\mathbf{A}_{21} \in \mathbb{R}^{(m-\bar{m}) \times \bar{n}}$, $\mathbf{A}_{22} \in \mathbb{R}^{(m-\bar{m}) \times (n-\bar{n})}$.

Proposition 3.2.6. *Consider a protocol solving Problem* (3.2.3) *and a coalition between the requester and agents with access to $\bar{m}$ of the values of $\mathbf{b}_{\mathcal{A}}$ and $\bar{n}$ of the values of $\mathbf{c}_{\mathcal{A}}$. Suppose the cost and constraint matrices $\mathbf{A}_{\mathcal{C}}$, $\mathbf{Q}_{\mathcal{C}}$ are public. Then:*

(1) if $\bar{m} < m$ and there exists a vector $\boldsymbol{\gamma} \in \mathbb{R}^{m-\bar{m}}$ such that $\boldsymbol{\gamma} \neq 0, \boldsymbol{\gamma} \succeq \mathbf{0}$ and $\mathbf{A}_{21}^{\intercal}\boldsymbol{\gamma} = \mathbf{0}$, then the coalition cannot uniquely retrieve the value of $\mathbf{b}_{\mathcal{A}}$;

(2) if additionally $\bar{n} < n$ and $\mathbf{A}_{22}^{\intercal}\boldsymbol{\gamma} \neq \mathbf{0}$ then the coalition cannot uniquely retrieve the value of $\mathbf{c}_{\mathcal{A}}$.

The proof is based on the fact that the variables $\mathbf{b}_{\mathcal{A}}$, $\mathbf{c}_{\mathcal{A}}$, $\mathbf{x}^*$, $\boldsymbol{\mu}^*$ satisfy the optimality conditions (3.2.13)-(3.2.15). Specifically, these are conditions that the unknown variables $b_{\bar{m}+1}, \ldots, b_m$ and $c_{\bar{n}+1}, \ldots, c_n$, as well as the optimal dual variables $\boldsymbol{\mu}^*$, must satisfy given all the rest of the known variables. Hence, if there are multiple such solutions to the KKT conditions the coalition cannot uniquely determine the private variables. The proof is given in Section B.3.

Proposition 3.2.7. *Consider a protocol solving Problem* (3.2.3) *and a coalition between the cloud and agents with access to $\bar{m}$ of the values of $\mathbf{b}_{\mathcal{A}}$ and $\bar{n}$ of the values of $\mathbf{c}_{\mathcal{A}}$. Then,* *a coalition that satisfies $\bar{m} < m$ and $\bar{n} < n$ cannot uniquely retrieve the values of $\mathbf{b}_{\mathcal{A}}, \mathbf{c}_{\mathcal{A}}$ and $\mathbf{x}^*$.*

The proof follows from the fact that $\mathbf{Q}_{\mathcal{C}}$ is a positive definite matrix, hence, the solution $\mathbf{x}^*$ is finite, and $\mathbf{A}_{\mathcal{C}}$ does not have any columns or rows of zeros. A coalition between the cloud and the $\bar{p} < p$ agents cannot solve (3.2.3) since it lacks all the data to define it ($\bar{m} < m$ and $\bar{n} < n$), so it cannot uniquely retrieve $\mathbf{x}^*$ and the rest of the agents' private data.

Propositions 3.2.6 and 3.2.7 give general sufficient conditions for the desired non-unique retrieval property to hold and are independent of the exact problem instance values. If the conditions stated are not satisfied, then the previous result in Theorem 3.2.3 still holds. However, the additional non-unique retrieval property may fail because the inputs and outputs of the coalition are sufficient to leak information about the rest of the private inputs. The above analysis can also be extended to the case where some of the matrices $\mathbf{Q}_{\mathcal{C}}, \mathbf{A}_{\mathcal{C}}$ are private information.

Let us now use Definition 3.2.5 to analyze the alternative protocol in Section 3.2.4.1 and the effect the release of more information to the requester has on the privacy of the inputs of the honest agents. We perform the analysis at a setup where the protocol runs for a sufficient number of iterations and hence the dual variable $\boldsymbol{\mu}_k$ has converged to the true optimal value $\boldsymbol{\mu}^*$ and the algorithm has also converged to the true optimal primal value $\mathbf{x}^*$. Suppose the requester has access to the sign of the unprojected optimal dual variable $\boldsymbol{\mu}^* + \nabla g(\boldsymbol{\mu}^*)$ – note that this is the case when we employ the alternative solution. In combination with the solution $\mathbf{x}^*$, this information can be further employed by the requester to infer private information of the agents.

Proposition 3.2.8. *Consider a protocol solving Problem (3.2.3) where the requester has access to the solution $\mathbf{x}^*$ and also the sign of the unprojected optimal dual variables $\boldsymbol{\delta} := sign(\boldsymbol{\mu}^* + \nabla g(\boldsymbol{\mu}^*)) \in \{+1, -1, 0\}^m$. Suppose further the matrix $\mathbf{A}_{\mathcal{C}}$ is publicly available. Then the private values $\mathbf{b}_{\mathcal{A}}$ cannot be uniquely retrieved by the requester if and only if $\delta_i < 0$ for some $i \in \{1, \ldots, m\}$.*

Here $\delta_i < 0$ implies that the corresponding optimal dual value is zero, $\mu_i^* = 0$. This means that the corresponding constraint in problem (3.2.3) is inactive at the optimal solution $\mathbf{x}^*$ [47, Ch. 5] and the requester cannot uniquely determine the ith element of the corresponding bound $\mathbf{b}_{\mathcal{A}}$. In the opposite case, if all constraints (3.2.14) are either active or redundant at the optimal solution ($\boldsymbol{\mu}^* \succeq \mathbf{0}$), this is revealed to the requester because in that case $\delta_i \geq 0$, and the private value $\mathbf{b}_{\mathcal{A}}$ is uniquely determined by $\mathbf{b}_{\mathcal{A}} = \mathbf{A}_{\mathcal{C}}\mathbf{x}^*$.

Proposition 3.2.9. *Consider the setup of Proposition 3.2.8 and the matrix $\mathbf{Q}_{\mathcal{C}}$ is publicly available. The private values $\mathbf{c}_{\mathcal{A}}$ cannot be uniquely retrieved from the outputs $\mathbf{x}^*$ and δ of the requester if and only if $\delta_i > 0$ for some $i \in \{1, \ldots, m\}$.*

The case $\delta_i > 0$ corresponds now to the case where the corresponding constraint is active at the optimal solution $\mathbf{x}^*$. When this fails, all constraints are inactive at $\mathbf{x}^*$ and they do not play a role. Hence, we have an unconstrained quadratic problem in (3.2.3), i.e., the optimal solution satisfies the first order condition $\mathbf{Q}_{\mathcal{C}}\mathbf{x}^* + \mathbf{c}_{\mathcal{A}} = \mathbf{0}$, which reveals the value of $\mathbf{c}_{\mathcal{A}}$ to the requester. To guarantee privacy with respect to both private values $\mathbf{b}_{\mathcal{A}}, \mathbf{c}_{\mathcal{A}}$ we need both an inactive constraint ($\delta_i < 0$) as well as an active constraint ($\delta_j > 0$) for some $i, j \in \{1, \ldots, m\}$, leaving some uncertainty in the estimations performed by the requester. Similar analysis may be performed for collusions, e.g., of the requester and some agents.

3.2.5 Implementation

The efficiency of a secure multi-party protocol is measured in the complexity of the computations performed by each party, along with the rounds of communications between the parties. While the former is relevant from the perspective of the level of computational power required, the latter are relevant when communication is slow.

In the setup we considered, the agents are low-power platforms, hence, they are only required to effectuate one encryption and one communication round, but the cloud and the requester are platforms with high computational capabilities, e.g., servers.

Let σ be the bit-length of the modulus N. Then, the size of a Paillier ciphertext is 2σ, regardless of the size of the plaintext in $\mathbb{Z}_N$. A Paillier encryption takes one online exponentiation and one multiplication modulo N^2, and a decryption takes one exponentiation modulo N^2 and one multiplication modulo N. The addition of two ciphertexts takes one modular multiplication modulo N^2. A multiplication of a ciphertext with a plaintext of l bits is achieved as a modular exponentiation modulo N^2. A multiplication of two elements in $\mathbb{Z}^*_{N^2}$ can be achieved in $O((2\sigma)^{1.7})$. A modular exponentiation in N^2 with an l-bit exponent can be computed in $O(l(2\sigma)^2)$ and can be sped up via the Chinese Remainder Theorem when the factorization of N is known. A DGK encryption takes one modular exponentiation and one modular multiplication in $\mathbb{Z}^*_{N_{\mathrm{DGK}}}$, and a DGK decryption–checking if the encrypted message is 0 or not–takes an exponentiation modulo p_{DGK}.

We implemented the protocol proposed in Section 3.2.3.4 in Python 3 and ran it on a 2.2 GHz Intel Core i7 processor. For random instances of the data in Problem (3.2.3), with $\sigma = 1024$ and $l = 32$ bits, and 0 ms delay for communication, we obtain the average running times depicted with the dashed gray lines in Figure 3.2 and Figure 3.3.

Figure 3.2 depicts the online execution time of Protocol 3.2.4 with the iterations as in Protocol 3.2.3. As expected, because we work in the dual space, the time required for running the iterations varies very little with the number of variables, but increases with the number of constraints. However, in the case of Figure 3.3, which depicts the online execution time of Protocol 3.2.4 with the alternative iterations as in Section 3.2.4.1, the time varies more with the number of variables n and not as much with m as Protocol 3.2.3 does. The reason is that we work with substantially larger numbers than in the previous case, due to the large factor with which we have to multiply in order to be able to de-mask the dual iterate, which amplifies the time difference between problems of different dimensions.

The trade-off between the privacy and communication can be seen when we artificially add a communication delay of 20 ms between the cloud and the requester, to simulate the delays that can occur on a communication network. It can be observed in Figures 3.2 and 3.3 that the communication delay has a smaller effect on the less private protocol than on the fully private Protocol 3.2.3.

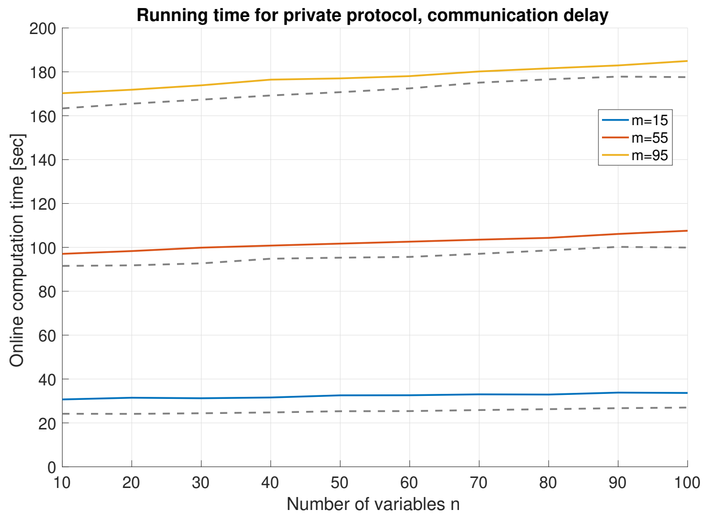


Figure 3.2: Average running time of Protocol 3.2.4 for problem instances with the number of variables on the abscissa and the number of constraints in the legend, with a 20 ms communication delay (colored solid) and undelayed (gray dashed). The simulation is run for 30 iterations, a 1024 bit key and 32 bit messages (16 bit precision). The statistical parameter for additive blinding is 100 bits.

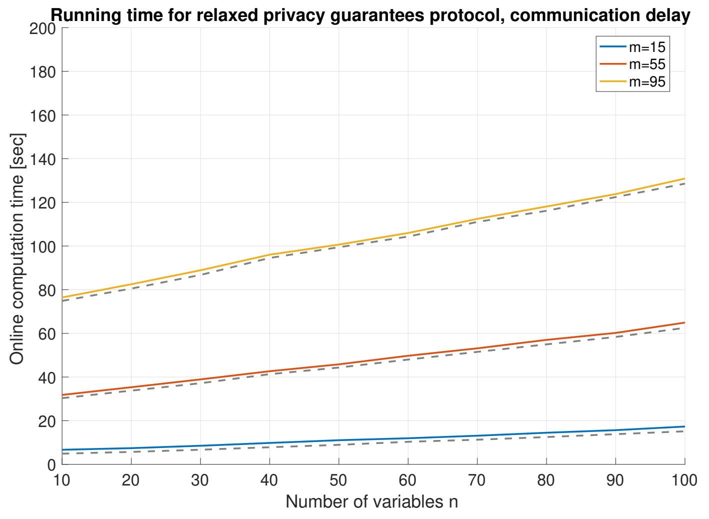


Figure 3.3: Average running time of Protocol 3.2.4 with the alternative iterations as in Section 3.2.4.1 for problem instances with the number of variables on the abscissa and the number of constraints in the legend, with a 20 ms communication delay (colored solid) and undelayed (gray dashed). The simulation is run for 30 iterations, a 1024 bit key and 32 bit messages (16 bit precision). The statistical parameter for multiplicative blinding is 40 bits.

3.3 Secure evaluation of Lasso

Organization. We outline the Lasso problem and the unsecure solution using ADMM in Section 3.3.1 and we present the privacy requirements for the secure solution. In Section 3.3.2, we describe the multi-party homomorphic encryption scheme used and justify the choice of distributed servers and distributed bootstrapping, as well as the method chosen for polynomial approximation. We then detail the challenges for achieving secure Lasso and their corresponding solutions, and assemble the encrypted protocol. Finally, Section 3.3.4 shows our implementation results and observations for Lasso problems of varying dimensions. The proof of privacy for the encrypted protocol is given in Appendix B.5.

3.3.1 Problem formulation

For a covariate matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, a vector of outcomes $\mathbf{b} \in \mathbb{R}^m$, the variable $\mathbf{x} \in \mathbb{R}^n$ and a penalty parameter $\lambda > 0$, the Lasso problem in its Lagrangian form is given by:

$$\min_{\mathbf{x}} \ \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_1. \tag{3.3.1}$$

For dependent covariates, there is no closed-form solution to (3.3.1) and many iterative optimization algorithms have been proposed in the literature [123, Ch. 5]. For example, Lasso problems can be solved using proximal methods or augmented Lagrangian methods, such as the Alternating Direction Method of Multipliers (ADMM) [48], [123, Ch. 5]. Splitting the objective function in the ADMM way, we get:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \ & \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{z}\|_1 \\ \text{s.t.} \ & \mathbf{x} - \mathbf{z} = \mathbf{0}. \end{aligned} \tag{3.3.2}$$

Let $S_\alpha(\mathbf{x}) = (\mathbf{x} - \alpha\mathbf{1})_+ - (-\mathbf{x} - \alpha\mathbf{1})_+$ denote the soft thresholding operator. The ADMM algorithm for (3.3.2) is:

$$\begin{aligned}
\mathbf{x}^{k+1} &= \arg\min_{\mathbf{x}} \left(\frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2 + \frac{\rho}{2}\|\mathbf{x}-\mathbf{z}^k+\mathbf{w}^k\|_2^2\right) \\
&= (\mathbf{A}^\intercal\mathbf{A}+\rho\mathbf{I})^{-1}\left(\mathbf{A}^\intercal\mathbf{b}+\rho(\mathbf{z}^k-\mathbf{w}^k)\right) \\
\mathbf{z}^{k+1} &= \arg\min_{\mathbf{z}} \left(\lambda\|\mathbf{z}\|_1 + \frac{\rho}{2}\|\mathbf{x}^{k+1}-\mathbf{z}+\mathbf{w}^k\|_2^2\right) = S_{\lambda/\rho}(\mathbf{x}^{k+1}+\mathbf{w}^k) \\
\mathbf{w}^{k+1} &= \mathbf{w}^k+\mathbf{x}^{k+1}-\mathbf{z}^{k+1}.
\end{aligned} \tag{3.3.3}$$

While for general optimization problems, the ADMM might converge slowly, for the Lasso problem it is known to have a fast convergence of a few (tens of) iterations for a large range of parameter $\rho > 0$ [48]. ADMM is a fast choice in the cases where a very high precision of the optimal result is not required, which is the case in noisy control problems, like the one we investigate in Chapter 7.3.

Goals and privacy requirements. A client outsources problem (3.3.1) to a cloud service that has to compute the optimal solution based on the data from the client.

We consider the cloud service to be *semi-honest*, meaning it does not deviate from the client's specifications, but can process the data it receives to extract private information for its own profit. The cloud service can be a conglomerate of K servers (see Figure 3.4), possibly belonging to different organizations, offering the guarantee that not all K servers collude.

Under this adversarial model, we require *client data confidentiality*, i.e., an adversary corrupting at most $K-1$ of the servers should not be able to infer anything about the client's inputs and outputs, which consist of the values of the matrix $\mathbf{A}$ and the vector $\mathbf{b}$, any intermediate values, and solution $\mathbf{x}^*$. The penalty λ and parameter ρ can be chosen by

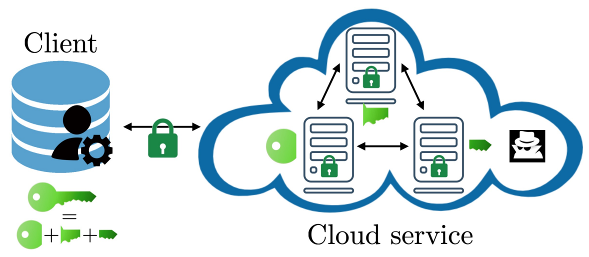


Figure 3.4: Schematic diagram of the problem, with a client outsourcing its encrypted data to a cloud service, that is authorized to compute on the data, but not to decrypt it.

the cloud service or chosen by the client, but are public. The formal privacy definition can be particularized from the multi-party privacy Definition 2.2.6 and is given by Definition 3.3.1.

Definition 3.3.1. (Client privacy w.r.t. semi-honest behavior of servers) Let x_C be the private input of a client and $x_{S,i}$ be the input of server i, for $i = 1, \ldots, n$. The client wants the servers to evaluate a functionality f and return the result $f(x_C, x_{S,1}, \ldots, x_{S,n})$. Denote the inputs by $\bar{x} = (x_C, x_{S,1}, \ldots, x_{S,n})$. Let $f_{S,i}(\bar{x})$ denote the corresponding result at server i. For $I = \{i_1, \ldots, i_t\} \subset [n] := \{1, \ldots, n\}$, with $|I| < n$, we let $f_{S,I}(\bar{x})$ denote the subsequence $f_{S,i_1}(\bar{x}), \ldots, f_{S,i_t}(\bar{x})$, which models a coalition of a number of servers. Let Π be a n-party protocol for computing f. Denote the view of server i on input $(\bar{x})$ by $V^{\Pi}_{S,i}(\bar{x})$ and denote the view of a coalition between servers by $V^{\Pi}_{I}(\bar{x}) = (I, V^{\Pi}_{S,i_1}(\bar{x}), \ldots, V^{\Pi}_{S,i_t}(\bar{x}))$. For a deterministic functionality f, we say that Π **privately computes** f if there exist **simulators** S, such that, for every $I \subset [n]$, it holds that, for any inputs $\bar{x}_t = (x_{S,i_1}, \ldots, x_{S,i_t})$:

$$\{S(I, (\bar{x}_t), f_I(\bar{x}_t))\}_{\bar{x} \in (\{0,1\}^*)^n} \overset{c}{\equiv} \{V^{\Pi}_{I}(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^n}.$$

3.3.2 Background

3.3.2.1 Distributed Bootstrapping for Homomorphic Encryption (HE)

If done locally at a server with no access to the private key, bootstrapping is a very expensive operation, consuming around 10 levels and introducing more noise (from encrypted approximation of nonpolynomial functions); see [60, 63] for details. Apart from the computationally intensive bootstrapping procedure, all the prior and posterior operations are impacted, since ciphertexts are required to have an extra 10 levels, leading to very large scheme parameters and ciphertexts size, making centralized bootstrapping undesirable.

Instead of performing bootstrapping locally, a server can ask the client to refresh a ciphertext on level 0. However, this implies more computation, communication and availability from the client, which is often prohibitive. A preferable solution is to use two or more servers for the computation and the refreshing. However, corrupting only two servers can be attainable by an adversary. Increasing the number of servers decreases the probability of an adversary corrupting them all. *Distributed bootstrapping trades substantial computation power to one round of communication and does not introduce as much noise as the centralized bootstrapping*, as described below.

In multi-party HE schemes, the private key is additively secret shared between a number of servers, meaning that no proper subset of the servers can decrypt. An important assumption is that an adversary cannot corrupt all servers at once, hence the private key is never recovered. In [169], a multi-party HE scheme is described, where servers can carry out the homomorphic computations locally and only need to interact for decryption and bootstrapping. In our scenario, the decryption will be performed at the client so we are only interested in distributed bootstrapping. Intuitively, distributed bootstrapping requires each server to use its local secret share of the private key to perform a partial decryption, mask this result and send it to the other servers. Summing up all the partial decryptions results in a refreshed ciphertext with the desired number of levels that can be correctly decrypted to the original message. The masking needs to provide statistical privacy of the message, so we require the mask to be > 80 bits larger than the size of messages and that no overflow occurs (the statistical security parameter is generally smaller than the computational security parameter). This means that distributed bootstrapping consumes around 3 levels; see more details in [96, 169].

In the rest of this chapter, we work with the multi-party version [96] of the leveled HE scheme CKKS [62]. Specifically, we use the version of the CKKS scheme [62], optimized to run on machine word size of 64-bit integer arithmetic [64, 117] instead of multiprecision integer arithmetic. We will denote by $\mathcal{E}_{\mathrm{v}0}(\mathbf{x})$ the encryption of the vector $\mathbf{x}$ followed by trailing zeros and by $\mathcal{E}_{\mathrm{v}*}(\mathbf{x})$ the encryption of the vector $\mathbf{x}$ followed by junk elements (elements whose value we do not care about).

3.3.2.2 Polynomial approximation and Chebyshev series

As described above, homomorphic encryption can evaluate polynomials on encrypted values. However, other operations such as trigonometric functions, divisions or comparisons are not supported. Therefore, we prefer to evaluate a polynomial approximation of the nonpolynomial functions. We choose to work with the Chebyshev polynomial series rather than the more common Taylor power series due to better precision and smaller approximation error. Specifically, the Chebyshev series polynomial interpolation is a near-minimax approximation of a continuous function on the interval $[-1, 1]$ [160].

However, polynomial approximation is not a panacea: for non-smooth functions, it gives a reasonable error only on relatively small intervals or using high degree polynomials. We choose to use this method, rather than other encrypted computation tools that can exactly evaluate non-smooth functions at the cost of more communication, knowing that we are dealing with noisy systems, where the small approximation errors are absorbed by noise.

3.3.3 Encrypted distributed Lasso

The setting we consider is the following: the client encrypts its data $\mathbf{A}, \mathbf{b}$ and secret shares its private key to a number of servers. The servers are responsible to compute and return the solution of problem (3.3.2) to the the client.

3.3.3.1 Challenge: Evaluating nonpolynomial functions

In the steps (3.3.3) of the ADMM algorithm for problem (3.3.2), the soft thresholding function is nonpolynomial, yet we need to evaluate it on encrypted data when computing $\mathbf{z}^{k+1}$. We deal with this challenge by approximating the soft thresholding function using a polynomial on a fixed interval via the Chebyshev series (we hardcode the coefficients for this function). If the interval is not $[-1, 1]$, we first apply a linear transformation to bring the inputs to this interval.

In the context of encrypted evaluation, a high polynomial degree increases the number of levels necessary for the computations, as well as the number of homomorphic multiplications between ciphertexts, which are expensive operations (compared to plaintext-ciphertext multiplications or additions). While we cannot consume fewer than $\lceil \log n \rceil$ levels to evaluate a polynomial of degree n, we can reduce the $O(n)$ homomorphic multiplications from the naive evaluation. Specifically, we implement the Paterson-Stockmayer algorithm [182], which reduces the number of homomorphic multiplications to $\lceil \sqrt{2n} + \log n \rceil + \mathcal{O}(1)$ by recursively evaluating polynomials of smaller degree. We modify the Paterson-Stockmayer algorithm that works with power series to work with Chebyshev series. The benefit of this algorithm compared to the naive evaluation is visible after degree 5 and grows with the degree. As an example, to evaluate a non-monic polynomial of degree 25, we require 5 levels and 11 homomorphic multiplications between ciphertexts.

Remark 3.3.2. The "stability" of the ADMM iterations for the Lasso problem allows the value $\mathbf{x}^{k+1} + \mathbf{w}^k$ to stay within a fixed interval, given in Lemma 3.3.3. In practice, we choose this interval from prior simulation.

Lemma 3.3.3. *Define* $\mathbf{M} := \mathbf{A}^\intercal\mathbf{A} + \rho\mathbf{I}$, $\mathbf{n} := \mathbf{M}^{-1}\mathbf{A}^\intercal\mathbf{b}$ *and* $c := \sqrt{n}\lambda/\rho\|2\rho\mathbf{M}^{-1} - \mathbf{I}\|_2 + \|\mathbf{n}\|_2$*. Since* $\sigma := \|\rho\mathbf{M}^{-1}\|_2$ *is in* $(0, 1]$*, we have the following bounds for the quantity* $\|\mathbf{x}^{k+1} + \mathbf{w}^k\|_\infty$ *in* (3.3.3)*, for all* $k = 1, \dots, K_{\text{iter}}$*:*

$$\begin{aligned} \|\mathbf{x}^{k+1} + \mathbf{w}^k\|_\infty &\le \sigma^k\|\mathbf{n}\|_2 + \frac{1-\sigma^k}{1-\sigma}c, && \text{if } \sigma < 1 \\ \|\mathbf{x}^{k+1} + \mathbf{w}^k\|_\infty &\le \|\mathbf{n}\|_2 + kc, && \text{if } \sigma = 1. \end{aligned} \tag{3.3.4}$$

The proof is given in Section B.4.

3.3.3.2 Challenge: Evaluating iterations

Depending on the precision we choose, the polynomial approximation can have a high degree, implying the need of bootstrapping in order to continue operations in the subsequent iterations. We resolve this challenge by making use of multiple servers in order to realize a cheaper bootstrapping compared to a centralized bootstrapping and a less burdensome solution than requesting action from the client.

To this end, we turn to the distributed version of ADMM [48], such that we use the servers both to ease the computation of the optimal solution and to ensure privacy through encrypted computations. We split the matrix $\mathbf{A}$ and vector $\mathbf{b}$ into K parts, each to be held by a server, and rewrite (3.3.2) as:

$$\begin{aligned}
&\min_{\mathbf{x}_1,\dots,\mathbf{x}_K,\mathbf{z}} \ \frac{1}{2}\sum_{i=1}^{K} \|\mathbf{A}_i\mathbf{x}_i - \mathbf{b}_i\|_2^2 + \lambda\|\mathbf{z}\|_1 \\
&\text{s.t.}\ \ \mathbf{x}_i - \mathbf{z} = \mathbf{0}, \quad i = 1,2,\dots,K.
\end{aligned} \tag{3.3.5}$$

The ADMM algorithm for problem (3.3.5) is, for $i = 1,\dots,K$:

$$\begin{aligned}
\mathbf{x}_i^{k+1} &= (\mathbf{A}_i^\intercal\mathbf{A}_i + \rho\mathbf{I})^{-1}\left(\mathbf{A}_i^\intercal\mathbf{b}_i + \rho(\mathbf{z}^k - \mathbf{w}_i^k)\right) \\
\mathbf{z}^{k+1} &= \frac{1}{K}S_{\lambda/\rho}\left(\sum_{i=1}^{K}\mathbf{x}_i^{k+1} + \sum_{i=1}^{K}\mathbf{w}_i^k\right) \\
\mathbf{w}_i^{k+1} &= \mathbf{w}_i^k + \mathbf{x}_i^{k+1} - \mathbf{z}^{k+1}.
\end{aligned} \tag{3.3.6}$$

Each server is given ciphertexts corresponding to $\mathbf{A}_i, \mathbf{b}_i$. We assume that there is a preprocessing step where servers can precompute convenient ciphertexts that will be used often in the online iterations, such as $1/\rho\mathbf{A}_i^\intercal\mathbf{b}_i$ and $\rho\left(\mathbf{A}_i^\intercal\mathbf{A}_i + \rho\mathbf{I}\right)^{-1}$. As in the unencrypted case, the servers can use the matrix inversion lemma to compute an inversion of a smaller matrix, which saves in offline computation. Online, each server locally computes the encryptions of $\mathbf{x}_i$ and $\mathbf{w}_i$, then communicates to the other servers the local sum $\mathbf{x}_i^{k+1} + \mathbf{w}_i^k$, such that all servers are then able to compute $\mathbf{z}^{k+1}$. So far, the only communication necessary is the same as in the unencrypted ADMM.

3.3.3.3 Challenge: Realizing the fewest bootstrapping operations

Distributed bootstrapping requires all parties to start by holding the same ciphertext and all parties to obtain that refreshed ciphertext. Bootstrapping the ciphertext encrypting $\mathbf{z}^{k+1}$ seems attractive, because it is global and its evaluation involves the most sequential multiplications. However, this is not enough: $\mathbf{w}_i^{k+1}$ loses levels through $\mathbf{x}_i^{k+1}$, which is the result of a multiplication; so we would need to bootstrap also before computing $\mathbf{z}^{k+1}$, not just after.

Instead, we do the following trick. Each server already has to compute and send a ciphertext encrypting $\mathbf{x}_i^{k+1} + \mathbf{w}_i^k$ to the other servers in order to compute the global iterate

$\mathbf{z}^{k+1}$. This means that each server can then construct a packed ciphertext $\mathbf{c}^{k+1}$ encrypting $\left[(\mathbf{x}_1^{k+1} + \mathbf{w}_1^k)^\intercal (\mathbf{x}_2^{k+1} + \mathbf{w}_2^k)^\intercal \ \ldots \ (\mathbf{x}_K^{k+1} + \mathbf{w}_K^k)^\intercal\right]$ and distributedly bootstrap it. Afterwards, each server can extract the refreshed ciphertext containing its local value $\mathbf{x}_i^{k+1} + \mathbf{w}_i^k$, as well as a refreshed ciphertext containing $\sum_{i=1}^K \mathbf{x}_i^{k+1} + \mathbf{w}_i^k$ by repeatedly rotating and summing the refreshed ciphertext $\mathbf{c}^{k+1}$ (this takes $O(K)$ operations). From this value, each server can locally compute its encrypted iterates $\mathbf{w}_i^{k+1}$ and $\mathbf{x}_i^{k+1}$, while doing only one bootstrapping operation per iteration rather than two.

Apart from $\mathbf{x}_i^{k+1} + \mathbf{w}_i^k$, the servers send one more message to complete the bootstrapping operation, so there are two rounds of communication per iteration, one at the smallest admissible level (dictated by bootstrapping) and the other at the full number of levels required, computed below.

Assume that offline quantities are refreshed. Define l_B to be the number of levels for a statistically secure distributed bootstrapping and l_P to be the number of levels necessary for the evaluation of $S_{\lambda/\rho}(\cdot)$ at a desired precision: this is the degree of the approximation polynomial plus one, coming from the linear transformation to the interval $[-1, 1]$ (we merge the multiplication by $1/K$ in the Chebyshev coefficients). Hence, the fresh ciphertexts need to have $L = l_B + l_P + 1$ levels, if we bootstrap once per iteration. Because l_P is usually higher than 5, bootstrapping once every few iterations leads to larger parameters and ciphertexts.

3.3.3.4 Encrypted protocol

Protocol 3.3.1 describes the steps for privately solving the distributed Lasso problem.

We use an optimized diagonal method [115] for encrypted matrix-vector multiplication. Consider a matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{p} \in \mathbb{R}^n$. Denote the diagonals of $\mathbf{S}$ by $\mathbf{d}_i$, for $i = 0, \ldots, n-1$. Let $n_1 := \lceil \sqrt{n/2} \rceil$ and $n_2 := n/n_1$. The corresponding result $\mathbf{q} = \mathbf{S}\mathbf{p}$ is:

$$\mathbf{q} = \sum_{i=0}^{n-1} \mathbf{d}_i \odot \rho(\mathbf{p}, i) \tag{3.3.7}$$

$$= \sum_{j=0}^{n_2-1} \rho\left(\sum_{k=0}^{n_1-1} \rho(\mathbf{d}_{j\cdot n_1+k}, -j\cdot n_1) \odot \rho(\mathbf{p}, k); j\cdot n_1 \right). \tag{3.3.8}$$

With (3.3.8), we need $n_1 + n_2 = O(\sqrt{n})$ homomorphic rotations, given $\rho(\mathbf{d}_{j\cdot n_1+k}, -j \cdot n_1)$, instead of n if we use (3.3.7). In both cases we require n homomorphic multiplications.

For a rectangular matrix, we need extended diagonals but the method is the same. The header of the function that achieves this is MultDiag($\mathbf{S}, \mathbf{p}$), and we pass the matrix $\mathbf{S}$ as separate ciphertexts encoding diagonals rotated accordingly and the vector $\mathbf{p}$ encoded in a ciphertext with trailing zeros. (In line 9 in Protocol 3.3.1, a masking is performed in order to satisfy the latter requirement.) We implement MultDiag such that it returns a ciphertext that encodes the result $\mathbf{q}$ with trailing zeros. This method can be parallelized. MultDiag is performed locally at the servers (lines 3 and 11).

The header ApproxSoftT($\mathbf{p}$) represents the implementation of the Chebyshev interpolation element-wise for $\mathbf{p}$, for a given set of coefficients that specify the degree of the approximation and an interval for which the coefficients are valid. Internally, the input is normalized to the interval $[-1, 1]$ (this also masks the junk elements), such that the output is a ciphertext encoding the result with trailing zeros. ApproxSoftT is performed locally (line 8 in Protocol 3.3.1).

DBoot($\mathbf{p}$) is a distributed protocol, where all servers start with the ciphertext of the vector $\mathbf{p}$ and all servers obtain a ciphertext that contains the refreshed vector $\mathbf{p}$ having a predetermined number of levels (line 6 in Protocol 3.3.1). It implies one round of communication between all servers, as described in Section 3.3.2.1.

There is also an offline protocol that computes the input as listed in Protocol 3.3.1. We mention that the client distributes the rows of $\mathbf{A}$ and $\mathbf{b}$ to the servers and the shares of the private key. The servers compute $\mathbf{m}_i$ by multiplication and $\mathbf{M}_i$ by the matrix inversion lemma and a high degree polynomial approximation for the inversion function, and use rotation and masking in order to obtain the diagonal representation needed. The servers collaborate to bootstrap the ciphertexts, in order to start Protocol 1 on the desired level.

Theorem 3.3.4. *Protocol 3.3.1 achieves client data confidentiality with respect to semi-honest servers, assuming at least one of the servers is honest.*

The proof is given in Section B.5.

Protocol 3.3.1: Distributed encrypted protocol for (3.3.5) with *equal servers* and *equal data split*

Input: Public parameters: public key pk, number of servers K, number of maximum iterations K_{iter}. C: secret key sk. $S_1, \dots, S_K$: encryption of $\mathbf{M}_i = \rho(\mathbf{A}_i^\intercal \mathbf{A}_i + \rho \mathbf{I})^{-1}$, encryption of $\mathbf{m}_i = \frac{1}{\rho}(\mathbf{A}_i^\intercal \mathbf{b}_i)$, share of the secret key sk$_i$, the Chebyshev coefficients for evaluating $S_{\lambda/\rho}(\cdot)$ on a given interval.

Output: C: $\mathbf{x}^*$.

```
1: $S_{i=1,\dots,K}$: set initial values $\mathcal{E}_{\text{v}0}(\mathbf{x}_i^0)$, $\mathcal{E}_{\text{v}0}(\mathbf{w}_i^0)$, $\mathcal{E}_{\text{v}0}(\mathbf{z}^0)$ (the value of $\mathbf{z}^k$ is previously agreed upon);
2: for $k = 0, \dots, K_{\text{iter}} - 1$ do
3:     $S_{i=1,\dots,K}$: $\mathcal{E}_{\text{v}0}(\mathbf{x}_i^k) = \text{MultDiag}(\mathbf{M}_i, \mathbf{m}_i + \mathbf{z}^k - \mathbf{w}_i^k)$;
4:     $S_{i=1,\dots,K}$: compute and send to other servers the rotation of the sum $\mathcal{E}_{\text{v}0}(\mathbf{v}_i) := \rho(\mathbf{x}_i^{k+1} + \mathbf{w}_i^k, -(i-1)n)$;
5:     $S_{i=1,\dots,K}$: assemble $\mathcal{E}_{\text{v}*}(\mathbf{v}) := \mathcal{E}_{\text{v}*}([\mathbf{v}_1\, \mathbf{v}_2 \dots \mathbf{v}_K])$ by summing own ciphertext and all received shifted ciphertexts;
6:     $S_{i=1,\dots,K}$: perform part in the distributed bootstrapping to get $\mathcal{E}_{\text{v}*}(\mathbf{v}^b) := \text{DBoot}(\mathcal{E}_{\text{v}*}(\mathbf{v}))$;
7:     $S_{i=1,\dots,K}$: extract its refreshed sum of local iterates $\mathcal{E}_{\text{v}*}(\mathbf{v}_i^b) = \rho(\mathcal{E}_{\text{v}*}(\mathbf{v}^b), (i-1)n)$;
8:     $S_{i=1,\dots,K}$: rotate and sum $\mathcal{E}_{\text{v}*}(\mathbf{v}^b)$ to get $\mathcal{E}_{\text{v}*}(\sum_{i=1}^K \mathbf{x}_i^{k+1} + \mathbf{w}_i^k)$, then compute $\mathcal{E}_{\text{v}0}(\mathbf{z}^k) = \text{ApproxSoftT}(\frac{1}{K}\sum_{i=1}^K \mathbf{x}_i^{k+1} + \mathbf{w}_i^k, \lambda/\rho)$;
9:     $S_{i=1,\dots,K}$: $\mathcal{E}_{\text{v}0}(\mathbf{w}_i^{k+1}) = [\mathbf{1}_S^\intercal\, \mathbf{0}^\intercal]^\intercal \odot \mathcal{E}_{\text{v}*}(\mathbf{v}_i^b) - \mathcal{E}_{\text{v}0}(\mathbf{z}^k)$;
10: end for
11: $S_1$: compute $\mathcal{E}_{\text{v}0}(\mathbf{x}_1^{K_{\text{iter}}}) = \text{MultDiag}(\mathbf{M}_1, \mathbf{m}_1 + \mathbf{z}^{K_{\text{iter}}-1} - \mathbf{w}_1^{K_{\text{iter}}-1})$ and send it to the client $C$;
12: $C$: decrypt and extract $\mathbf{x}^*$.
```

Remark 3.3.5. This distributed setup can be used for the quadratic problem described in Section 3.2, by using polynomial approximation for the maximum function in the dual problem (3.2.11). However, unless special conditions on the matrices are given, the projected gradient iterates do not have the same "stability" property as in Remark 3.3.2. This might require larger intervals and worse approximation error or very high approximation degree. Hence, we preferred to exploit the architecture and the capabilities of the requester in

Section 3.2, and to obtain more precise solutions.

3.3.4 Numerical results

We evaluate Protocol 3.3.1 on Ubuntu 18.04 on a commodity laptop with 8 GB of RAM and Intel Core i7, implemented using the PALISADE library [178], for three servers using 8 threads. We set the parameters such that we get a security level of 128 bits, i.e., we use a ciphertext modulus of 436 bits and a ring dimension of 2^{14}. We obtain 6 decimal places precision for the results.

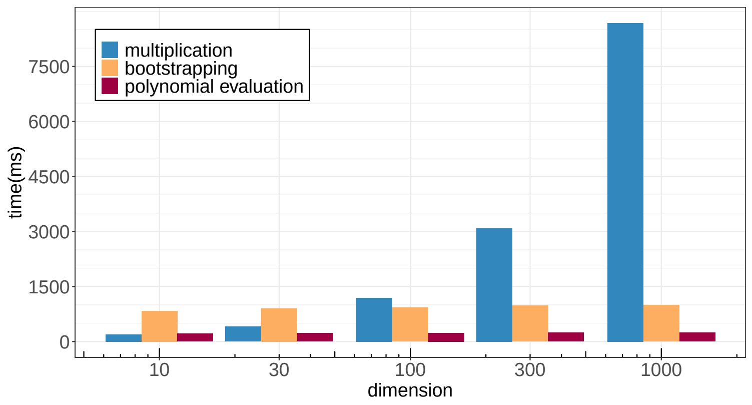


Figure 3.5: Timing for steps in one online iteration for one server for solving Lasso problems of various dimensions via encrypted distributed ADMM with three servers (Protocol 3.3.1). The legend shows the operation that takes the most time in the step.

In Figure 3.5, we show how the time for one ADMM iteration varies with the dimension of the problem, i.e., number of columns of matrix **A** in (3.3.2). The blue bar shows the time for lines 3 and 4 in Protocol 3.3.1, effectively consisting of the matrix-vector multiplication. The yellow bar shows the time for lines 5 and 6, representing the preparation for bootstrapping and the bootstrapping itself. Finally, the red bar represents lines 7–9 of Protocol 3.3.1, consisting mostly of the polynomial evaluation. We want to stress that the bootstrapping and polynomial evaluation are made independent from the dimension of the problem through packing, which represents a great advantage when increasing the dimension. On the other hand, for large dimensions, the encrypted matrix multiplication takes most of the computational and memory effort, and other methods that decrease storage and number of operations at the cost of more levels might be preferable.

Chapter 4

Weighted Aggregation

4.1 Introduction

As large amounts of data are circulated both from users to a cloud server and between users, there is a critical need for privately aggregating the shared data. Of particular interest is the general problem of weighted sum aggregation, that we explore in this chapter, in which an aggregating party needs to collect and sum contributions from a number of agents– the contributions consist of some local data weighted by some other relevant quantities. There is a wealth of examples spanning various research areas that require the computation of weighted aggregates: **(a)** Decentralized and cooperative linear control for multi-agent systems [69, 151, 223]; **(b)** Graph neural networks [136] and collaborative inference [111, 206]; **(c)** Average consensus [90, 229]; **(d)** Federated learning [165, 228], aggregation of linear inference results; **(e)** Energy price aggregation and management [68, 222], vehicle tolls collection [26, 146].

Each of the above examples can pose different privacy requirements on the local data of the agents, as well as on the corresponding weights. For example, in the context of federated learning, the model is locally trained by the agents and the aggregating server needs to compute the mean model without obtaining the local models. In some price collection instances, the prices can depend on private information known at the aggregator and can vary dynamically, so the aggregator knows the price weights, while the agents do not. Finally, there are cases where a system operator has invested resources into computing the control gains for a distributed system and wants to keep them private from both the agents and the aggregator, who needs to compute a linear control without knowing neither the local agent's states nor the gains. Similar privacy requirements are in place for secure inference, where a service provider has trained a proprietary model on its own data and wants to keep it private while allowing it to be deployed.

Related work

In the context of **(a)**, linear distributed control with homomorphically encrypted gains was addressed by [7, 12, 199], with [7] touching also on **(b)**. We will elaborate and improve on these works in Section 4.3.2. Concerning **(c)**, there is a body of research that targets the privacy of the local data of the agents achieving consensus, using partially homomorphic encryption or differential privacy: see [112, 113, 175, 192] and the references within. For **(d)** and **(e)**, works such as [2, 41, 89, 142, 147] provide private solutions for private sum aggregation, touching on a large base of cryptographic tools, such as secret sharing, threshold homomorphic encryption, differential privacy.

Contributions

This chapter considers the problem of *private weighted sum aggregation* with secret weights, where an aggregator wants to compute the weighted sum of the local data of some agents. Given the wide spread of private weighted sum aggregation problems with different privacy constraints, our first contribution is to review their solutions and give a unified formulation. We intend for this chapter to serve as a guide for choosing an efficient particular solution based on knowledge distribution and privacy demands. Our second contribution is to offer a private solution for the general case of weighted aggregation, where weights are hidden from all parties. Our third contribution is to extend these solutions to multi-dimensional data rather than scalar data and give valuable optimizations. Our fourth contribution is to implement and extensively demonstrate the runtime and communication improvements of up to 80% for the problem with hidden weights.

Organization. In Section 4.2, we outline the three types of problems and the desired security requirements. We first review existing solutions for *private sum aggregation* (weights are known by the agents, but not by the aggregator) in Section 4.3.1.1. Most of the previously mentioned literature falls into this category. Second, we describe *private weighted sum aggregation with centralized weights* (weights are known at the aggregator, but not at the agents), which can be solved from the lens of functional encryption for inner products in Section 4.3.1.2. Third, we give a solution for the more general case of *private weighted sum aggregation with hidden weights* (neither agents nor aggregator know the weights; they are generated by a system operator, who wants to keep them private) and improve it compared to previous work in terms of security: larger collusion threshold, communication: fewer messages exchanged, and runtime: fewer operations, in the multi-dimensional case in Section 4.3.2. Finally, we propose a private weighted sum aggregation scheme that uses the same secret keys for all the time steps of the computation, thus minimizing the number of communication rounds, and keeps the threshold of colluding agents at the cardinality of all but one participating agents in Section 4.5. The capabilities of this final scheme could also allow aggregation of nonlinear functions of private weights.

Our solutions achieve *aggregator obliviousness*, even under collusion between the participants. In order to make the schemes communication efficient, we use packing, which compresses a vector of messages in one plaintext, respectively one ciphertext. We show how to astutely perform the operations on the packed ciphertexts to reduce the computational and communication cost. Different solutions are presented for the different schemes, exploiting the characteristics of the underlying schemes, and are given in Section 4.4 and Section 4.6.

This chapter covers work presented in [7, 8, 12].

Special notation for this chapter

For a positive integer n, let $[n] := \{1, 2, \dots, n\}$. A quantity $(\cdot)_i$ refers to agent i and a quantity $(\cdot)_a$ refers to the aggregator. By $\mathbf{x}^{[j]}$, we refer to the j-th element of vector $\mathbf{x}$ and by $\mathbf{W}^{[jl]}$, to the element of matrix $\mathbf{W}$ on the j-th row and l-th column. κ is the security parameter. We denote the Paillier encryption primitive by E$(\cdot)$ and the decryption primitive by D$(\cdot)$. We denote negligible functions, defined in Definition 2.2.1 by $\eta(\cdot)$. $\phi(N)$ denotes Euler's totient function; for $N = pq$, with p, q primes, $\phi(N) = (p-1)(q-1)$. A value $x \in \mathbb{Q}_{l_i,l_f}$ represents a rational value $x = x_i.x_f$ with l_i bits for the integer part and l_f bits for the fractional part.

4.2 Problem statement

We investigate an aggregation problem of weighted contributions, depicted schematically in Figure 7.1. We consider a system with M agents and one aggregator. Each agent $i \in [M]$ has some data $\mathbf{x}_i(t) \in \mathbb{R}^{n_i}$ at time t and the aggregator wants to compute an aggregate of the data in the system $\mathbf{x}_a(t) \in \mathbb{R}^{n_a}$, where $\mathbf{W}_i \in \mathbb{R}^{n_a \times n_i}$ are constant weights designated for the local data of agent i:

$$\mathbf{x}_a(t) = \sum_{i=1}^{M} \mathbf{W}_i \mathbf{x}_i(t). \tag{4.2.1}$$

At every time step, each agent $i \in [M]$ has access to its local data $\mathbf{x}_i(t)$, either by direct measurement (e.g., location, energy consumption) or by computation (e.g., gradient of the model, local prediction). We consider three types of privacy requirements for private weighted sum aggregation.

Private Weighted Sum Aggregation with hidden weights

This case requires the strongest privacy guarantees:

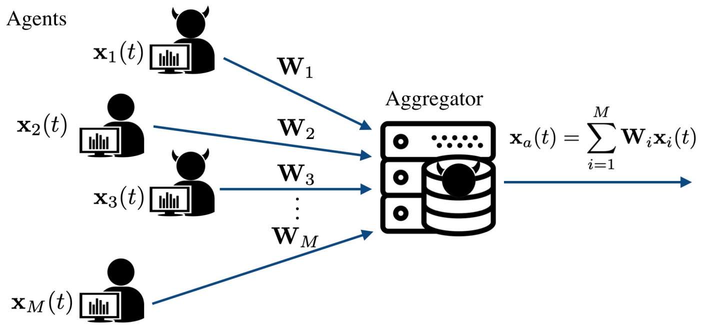


Figure 4.1: Diagram of the private weighted sum aggregation. Some of the participants can be corrupted and disclose their private data.

(a) Agent i should not infer anything about the other agents' local data $\mathbf{x}_j(t)$, $j \in [M]\backslash\{i\}$ or about the aggregator's result $\mathbf{x}_a(t)$ or about the weights $\mathbf{W}_i$, $i \in [M]$, including partial information such as $\mathbf{W}_i\mathbf{x}_i(t)$.

(b) The aggregator should only be able to compute $\mathbf{x}_a(t)$ and should not infer anything else about the agents' local data $\mathbf{x}_i(t)$ or the weights $\mathbf{W}_i$, $i \in [M]$, including partial information such as $\mathbf{W}_i\mathbf{x}_i(t)$.

Private Sum Aggregation

In this case, we replace (a) by:

(a) Agent i knows its corresponding weight $\mathbf{W}_i$ and should not infer anything about the other agents' local data and weights $\mathbf{x}_j(t), \mathbf{W}_j, j \in [M] \setminus \{i\}$, including partial information such as $\mathbf{W}_j\mathbf{x}_j(t)$, or about the aggregator's result $\mathbf{x}_a(t)$.

Private Weighted Sum Aggregation with centralized weights

In this case, we replace (b) by:

(b) The aggregator knows the weights $\mathbf{W}_i$ and should only be able to compute $\mathbf{x}_a(t)$ and should not infer anything else about the agents' local data $\mathbf{x}_i(t)$, $i \in [M]$, including partial information such as $\mathbf{W}_i\mathbf{x}_i(t)$.

These privacy requirements should hold even under collusion between the aggregator and at most $M-2$ agents, or between $M-1$ agents, i.e., a coalition should not be able to infer the private data of the remaining honest participants.

We consider computationally bounded adversaries that are *semi-honest*, defined in Definition 2.1.1. Such a model is chosen because the aggregator is interested in obtaining the correct result of the computation, and for instance, in applications involving pricing, the agents would be fined in they cheat.

The goal here is to protect the privacy of the inputs and intermediary computations, but reveal the output to the aggregator. As a side note, all the presented algorithms can support differential privacy, in case the output should also be protected.

In describing the rest of this section and the schemes in Sections 4.3.1.1, 4.3.1.2, 4.3.2 and 4.5, we focus on scalar data $w_i, x_i, x_a \in \mathbb{Z}_{\geq 0}$, for $i \in [M]$. After illustrating the functionalities, we provide methods for dealing with multi-dimensional rational data in Sections 4.4.1, 4.4 and 4.6.

Aggregator obliviousness for pWSAh

We give a formal description of the privacy requirements from Section 4.2 as a cryptographic game between an adversary and a challenger, where the adversary $\mathcal{A}$ can corrupt agents and the aggregator. The weights $w_{i\in[M]}$ are constant over the time steps, so the adversary is forced to specify constant weights; in particular, the adversary will specify two sets of weights: $w^{\mathcal{A},0}_{i\in[M]}$ and $w^{\mathcal{A},1}_{i\in[M]}$. The security game pWSAO (private Weighted Sum Aggregator Obliviousness) is as follows:

Setup. The challenger runs the Setup algorithm and gives the public parameters prm to the adversary.

Queries. The adversary can submit compromise queries and encryption queries that are answered by the challenger. In the case of compromise queries, the adversary submits an index $i \in [M]$ to the challenger and receives sk_i, which means the adversary corrupts agent i. The set of the corrupted agents is denoted by $\mathcal{C}$. In the case of encryption queries, the adversary is allowed one query per time step t and per agent $i \in [M]$. The adversary submits $(i, t, w_i^{\mathcal{A}}, x_i(t))$, where $w_i^{\mathcal{A}} = \{w_i^{\mathcal{A},0}, w_i^{\mathcal{A},1}\}$, and the challenger first runs $\mathrm{sw}_i^{\mathcal{A},0} = \mathrm{InitW}(\mathrm{prm}, i, w_i^{\mathcal{A},0})$, $\mathrm{sw}_i^{\mathcal{A},1} = \mathrm{InitW}(\mathrm{prm}, i, w_i^{\mathcal{A},1})$ and returns $\mathrm{Enc}(\mathrm{prm}, \mathrm{sw}_i^{\mathcal{A},0}, \mathrm{sk}_i, t, x_i(t))$ and $\mathrm{Enc}(\mathrm{prm}, \mathrm{sw}_i^{\mathcal{A},1}, \mathrm{sk}_i, t, x_i(t))$. The set of participants for which an encryption query was made by the adversary at time t is denoted by $\mathcal{E}(t)$.

Challenge. The adversary chooses a specific time step t^*. Let $\mathcal{U}^*$ denote the set of participants that were not compromised at the end of the game and for which no encryption query was made at time t^*, i.e., $\mathcal{U}^* = ([M] \cup \{a\}) \setminus (\mathcal{C} \cup \mathcal{E}(t^*))$. The adversary specifies a subset of participants $\mathcal{S}^* \subseteq \mathcal{U}^*$. At this time t^*, for each agent $i \in \mathcal{S}^* \setminus \{a\}$, the adversary chooses two plaintext series $x_i^0(t^*)$ and $x_i^1(t^*)$, along with $w_i^{\mathcal{A},0}$ and $w_i^{\mathcal{A},1}$, and sends them to the challenger. If $\mathcal{S}^* = \mathcal{U}^*$ and $a \notin \mathcal{S}^*$, i.e., the aggregator has been compromised, then, the values submitted by the adversary have to satisfy $\sum_{i\in\mathcal{S}^*} w_i^{\mathcal{A},0} x_i^0(t^*) = \sum_{i\in\mathcal{S}^*} w_i^{\mathcal{A},1} x_i^1(t^*)$. The challenger flips a random bit $b \in \{0,1\}$ and computes $\text{Enc}(\text{prm}, \text{InitW}(\text{prm}, i, w_i^{\mathcal{A},b}), \text{sk}_i, t, x_i^b(t^*))$, $\forall i \in \mathcal{S}^*$. The challenger then returns the ciphertexts to the adversary.

Guess. The adversary outputs a guess $b' \in \{0,1\}$ on whether b is 0 or 1. The advantage of the adversary is defined as:

$$\mathbf{Adv}^{\text{pWSAO}}(\mathcal{A}) := \left|\Pr[b' = b] - \frac{1}{2}\right|.$$

The adversary wins the game if it correctly guesses b.

Definition 4.2.1. A scheme pWSAh = (Setup, Enc, InitW, AggrDec) achieves *weighted sum aggregator obliviousness* if no probabilistic polynomial-time adversary has more than negligible advantage in winning this security game:

$$\mathbf{Adv}^{\text{pWSAO}}(\mathcal{A}) \leq \eta(\kappa).$$

4.3 Private weighted sum aggregation

4.3.1 Review of private sum aggregation

4.3.1.1 Private sum aggregation

Private sum aggregation (pSA), introduced in [187, 202], enables an untrusted aggregator to compute the sum of the private data contributed by agents, without learning their individual contributions. Additionally, it allows noise mechanisms to ensure that the aggregate is differentially private. Improvements in terms of efficiency and functionality of pSA have been proposed in [28, 33, 41, 57, 131, 208] and the references within. The formal definition of *aggregator obliviousness* that pSA schemes have to satisfy (informally described in Section 4.2) was introduced in [202] and is given as a cryptographic game between an adversary and a challenger, similar to the game we describe in Section 4.2.

When the weights w_i are known to the agents, equation (4.2.1) can be computed privately with a pSA scheme. Specifically, in every time step, denoted by $t \in \mathbb{Z}_{\geq 0}$, each agent $i \in [M]$ holds a private value $x_i(t)$ and w_i. Define $v_i(t) := w_i x_i(t)$. The aggregator wants to compute the aggregate statistics over the private values: $x_a(t) = \sum_{i \in [M]} v_i(t)$.

Let l denote the maximum number of bits of $x_i(t), w_i$, $\forall i \in [M]$. An assumption that we make for the rest of the chapter is:

Assumption 4.3.1. For each time step t, after discretization, $x_i(t), w_i, w_i x_i(t), x_a(t) < N$, $\forall i \in [M]$, i.e., there is no overflow for N specified in each scheme.

The most intuitive pSA scheme involves secret sharing. Each participant will be given by a trusted dealer at the onset of scheme a secret share of zero for each time step. Each agent will use this share to mask its local data, like a one-time pad–see Preamble 2.4. The aggregator will then sum all the contributions it receives and obtain the desired sum, as the shares of zero will cancel out. The idea of using shares of zero to additively mask private values in aggregation problems was explored, e.g., in [41, 54, 149, 208].

A private sum aggregation scheme should consist of the following algorithms, $\mathrm{pSA}_1 = (\mathrm{Setup}, \mathrm{Enc}, \mathrm{AggrDec})$:

- $\mathrm{Setup}(1^\kappa, M, w_{i\in[M]}, T)$: take as input the security parameter κ, the number of agents M and time period T, and output public parameters prm, secret information for each agent sk_i, $i \in [M]$ and for the aggregator sk_a. This happens as follows: let $N = \max(\kappa, 2l + M)$, then, for each time step $t \in [T]$, generate $M+1$ shares of zero:

$$\sum_{i \in [M] \cup \{a\}} s_i(t) = 0, \quad s_i(t) \in \mathbb{Z}_N.$$

  Set $\mathrm{prm} = (\kappa, M)$, $\mathrm{sk}_i = (s_i(t), w_i)$, $\mathrm{sk}_a = (s_a(t))$.

- $\mathrm{Enc}(\mathrm{prm}, \mathrm{sk}_i, t, x_i(t))$: take as input the public parameters, agent i's secret information, the time step and the local private value. Set $v_i(t) = w_i x_i(t)$ and compute:

$$c_i(t) = v_i(t) + s_i(t) \in \mathbb{Z}_N.$$

- $\text{AggrDec}(\text{prm}, \text{sk}_a, t, \{c_i(t)\}_{i\in[M]})$: take as input the public parameters, the aggregator's secret information, the time step and the ciphertexts of the agents for that time step. The aggregator obtains $x_a(t) = \sum_{i\in[M]} v_i(t)$, as follows:

$$x_a(t) = s_a(t) + \sum_{i\in[M]} c_i(t) \bmod N.$$

The correctness of pSA_1 is based on the correct generation of the random shares of zero in Setup, that cancel out after aggregation. Aggregator obliviousness is based on the perfect security of masking the private data in Enc by a one-time pad.

The scheme pSA_1 requires a different set of shares of zero for every time step (otherwise, partial information such as differences between private contributions at different time steps is leaked), which can involve elaborate communication, as we will see in Section 4.3.3. On the other hand, the $\text{pSA}_2 = (\text{Setup}, \text{Enc}, \text{AggrDec})$ scheme from [131], that we describe next, only requires an initial set of shares of zero. This scheme is based on the Paillier cryptosystem [177], see Preamble 2.5.1.

- $\text{Setup}(1^\kappa, M, \{w_i\}_{i\in[M]}, T)$: generate p, q to be two equal-size primes and set $N = pq$ with $\gcd(\phi(N), N) = 1$, $\lfloor \log_2 N \rfloor = \kappa$. Define a hash function that acts as a random oracle $H : \mathbb{Z} \to \mathbb{Z}^*_{N^2}$. Generate $M + 1$ shares of zero:

$$s_a := -\sum_{i\in[M]} s_i, \quad s_i \in \mathbb{Z}^*_{N^2}.$$

  Set $\text{prm} = (\kappa, N, H)$, $\text{sk}_i = (s_i, w_i)$, $\text{sk}_a = (s_a)$.

- $\text{Enc}(\text{prm}, \text{sk}_i, t, x_i(t))$: set $v_i(t) = w_i x_i(t)$ and output:

$$c_i(t) = (1 + N)^{v_i(t)} H(t)^{s_i} \bmod N^2.$$

- $\text{AggrDec}(\text{prm}, \text{sk}_a, t, \{c_i(t)\}_{i\in[M]})$: take as input the public parameters, the aggregator's secret information, the time step and the ciphertexts of the agents for that time step. The aggregator obtains $x_a(t) = \sum_{i\in[M]} v_i(t)$, as follows:

$$V(t) = H(t)^{s_a} \prod_{i\in[M]} c_i(t) \bmod N^2, \qquad x_a(t) = (V(t) - 1)/N.$$

The correctness of this scheme follows from the generation of the secret shares and from (A.1.1) in Appendix A.1. The aggregator obliviousness property is proved in [131]. Furthermore, [131] shows that the security of the scheme is not impacted when the hash function H takes values in $\mathbb{Z}_{N^2}$, not in $\mathbb{Z}^*_{N^2}$.

In the pSA$_2$ scheme, the aggregator can decrypt the sum *without* having access to $\phi(N)$, as opposed to what would be needed for Paillier decryption, see Preamble 2.5.1. This is crucial for the proof of aggregator obliviousness of pSA$_2$ and explains why we cannot use the same scheme to obtain the private weighted sum aggregation scheme with hidden weights when the weights are encrypted with the Paillier cryptosystem.

4.3.1.2 Private weighted sum aggregation with centralized weights

When the aggregator knows the weight w_i corresponding to each of the agents (and they are not identical), but the agents do not know them, we cannot reuse the above private sum aggregation schemes. We operate under the assumption that the constant weights are not chosen in an adversarial way and there is an auditor that checks them beforehand. This way, we ensure that, the weights are not chosen to single out only one piece of local data.

There are two lines of work that investigate this problem. First, in [39], the authors propose a distributed scenario for aggregation in a graph of agents using a threshold cryptosystem. This implies that after receiving contributions from the agents, the aggregator would ask for help in decrypting the aggregate value. The second line of work removes the extra communication required for decryption. Functional encryption [43] is a generalization of homomorphic encryption and allows a party to compute a functionality over the encrypted data of another party and obtain the desired solution without decryption. One of the few current practical implementations is the functionality of inner products, where one party holds one input and the other party holds the other [3]. Here, we formulate our problem of private weighted sum aggregation with weights known by the aggregator in terms of functional encryption for inner product: $x_a(t) = \langle [w_1, \dots, w_M], [x_1(t), \dots, x_M(t)] \rangle$.

The proposed scheme assumes constant weights, as in Section 4.2. The definition of aggregator obliviousness for this case can be written as a cryptographic game formalizing the requirements in Section 4.2. Stronger privacy definitions, from the perspective of functional encryption, can be found in [3].

We modify pSA_2 to get a private weighted sum with centralized weights scheme $\mathrm{pWSAc} = (\mathrm{Setup}, \mathrm{Enc}, \mathrm{AggrDec})$:

- $\mathrm{Setup}(1^\kappa, \{w_i\}_{i\in[M]}, T)$: given the security parameter κ, generate two equal-size prime numbers p, q and set $N = pq$ such that $\lfloor \log_2 N \rfloor = \kappa$ and $\gcd(\phi(N), N) = 1$. The public key is $\mathrm{pk} = (N)$. Sample M values $s_i \in \mathbb{Z}^*_{N^2}$ and set:

$$s_a = -\sum_{i\in[M]} w_i s_i. \tag{4.3.1}$$

  Choose a hash function that acts as a random oracle $H : \mathbb{Z} \to \mathbb{Z}^*_{N^2}$ (see [3]). Finally, set $\mathrm{prm} = (\kappa, N, H)$, $\mathrm{sk}_i = (s_i)$ and $\mathrm{sk}_a = (\{w_i\}_{i\in[M]}, s_a)$.

- $\mathrm{Enc}(\mathrm{prm}, \mathrm{sk}_i, t, x_i(t))$: For $x_i(t) \in \mathbb{Z}_N$, compute:

$$c_i(t) = (1+N)^{x_i(t)} \cdot H(t)^{s_i} \bmod N^2.$$

- $\mathrm{AggrDec}(\mathrm{prm}, \mathrm{sk}_a, t, \{c_i(t)\}_{i\in[M]}, \{w_i\}_{i\in[M]})$: compute

$$V(t) = H(t)^{s_a} \cdot \prod_{i\in[M]} c_i(t)^{w_i} \bmod N^2, \qquad x_a(t) = (V(t) - 1)/N.$$

Correctness follows after expanding $V(t)$: Using the binomial theorem modulo N^2, we can write $c_i(t) = (1+N)^{x_i(t)} H(t)^{s_i} \bmod N^2$. Then:

$$V(t) = H(t)^{s_a} \cdot (1+N)^{\sum_{i\in[M]} w_i x_i(t)} \cdot H(t)^{\sum_{i\in[M]} w_i s_i}.$$

Using $s_a = -\sum_{i\in[M]} w_i s_i$, we obtain that $(V(t)-1)/N = \sum_{i\in[M]} w_i x_i(t) = x_a(t)$, as needed. Aggregator obliviousness follows from the proof in [131], where the secret of the aggregator is now $s_a = -\sum_{i\in[M]} w_i s_i$ instead of $-\sum_{i\in[M]} s_i$, and the aggregator raises the ciphertexts of the participants to the respective power w_i. A different proof can be found in [3].

Notice that the keys are independent of the time period T. The Setup step can be performed as follows by a third-party dealer that does not need to know the weights of the aggregator. The dealer generates M random secrets s_i and sends one to each agent. The aggregator generates the public and secret key of an additively homomorphic encryption scheme, e.g., Paillier [177], encrypts the weights w_i, for $i \in [M]$ and sends them to the dealer. Then, the dealer computes (4.3.1) as:

$$\mathrm{E}(s_a) = \prod_{i\in[M]} \mathrm{E}(w_i)^{-s_i},$$

and sends it to the aggregator, which then simply decrypts s_a. If using the Paillier scheme, the modulus corresponding to the aggregator's scheme has to satisfy $N_a > N^2$.

4.3.2 Private weighted sum aggregation with hidden weights

A private weighted sum aggregation scheme for weights unknown to all participants is composed of algorithms $\mathrm{pWSAh} = (\mathrm{Setup}, \mathrm{InitW}, \mathrm{Enc}, \mathrm{AggrDec})$. The formal security definition of aggregator obliviousness is given in Definition 4.2.1 in Section 4.2 as a cryptographic game. This game mimics the real execution of the scheme, but with a more powerful adversary that can choose both the local data of the corrupted participants and the local data of uncorrupted participants. If even in this case, the adversary is not able to break the privacy of the scheme, then the scheme is private also when multiple participants collude and share their private information, but cannot set the private data of the honest participants.

In pWSAh, the weight w_i should be private from all participants, so one solution is to encrypt it. Then, agent i has to be able to send an encryption of the masked product $w_i x_i(t)$ to the aggregator, and the latter has to be able to compute and decrypt the result. This suggests the outline in Figure 4.2:

- w_i should be encrypted with an additively homomorphic encryption that the aggregator knows how to decrypt;

- the layer of encryption introduced in Enc should be compatible with the inner additively homomorphic layer;

- the aggregator should not be able to decrypt the individual contributions it receives from the agents, despite having the secret key of the homomorphic encryption scheme.

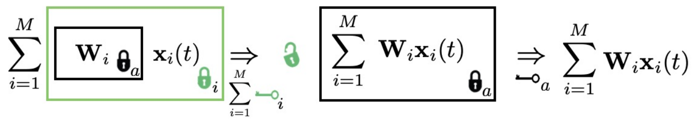


Figure 4.2: Diagram of the pWSAh functionality and privacy requirements.

To achieve the solution, we use a combination of the two schemes described in Section 4.3.1.1. For the outer layer of encryption, we use one-time pads as in pSA$_1$, which are compatible with the additively homomorphic property. For the inner layer of encryption, we use an asymmetric additive homomorphic encryption scheme. We instantiate it with the Paillier cryptosystem [177], due to its simplicity and popularity. More details about this cryptosystem can be found in Preamble 2.5.1.

Hence, the steps of the algorithms in pWSAh are:

- Setup$(1^\kappa, M, T)$: given the security parameter κ, get a pair of Paillier keys (pk, sk): generate two equal-size prime numbers p, q and set $N = pq$ such that $\lfloor \log_2 N \rfloor = \kappa$ and $\gcd(\phi(N), N) = 1$. Set: $\text{pk} = (N)$, $\text{sk} = \big(\phi(N), \phi(N)^{-1} \bmod N\big)$. For every $t \in [T]$, generate $M + 1$ shares of zero:

$$s_a(t) := -\sum_{i \in [M]} s_i(t), \quad s_i(t) \in \mathbb{Z}_N^*.$$

  Finally, set $\text{prm} = (\kappa, \text{pk})$, $\text{sk}_i = (s_i(t \in [T]))$ and $\text{sk}_a = (\text{sk}, s_a(t \in [T]))$.

- InitW$(\text{prm}, M, \{w_i\}_{i \in [M]})$: given the public key of the Paillier scheme pk, encrypt w_i

for $i \in [M]$ and return $\mathrm{sw}_i = \mathrm{E}(w_i) = (1+N)^{w_i} r^N \bmod N^2$, for r randomly sampled from $\mathbb{Z}_N$ and such that $\gcd(r, N) = 1$.

- $\mathrm{Enc}(\mathrm{prm}, \mathrm{sw}_i, \mathrm{sk}_i, t, x_i(t))$: for $x_i(t) \in \mathbb{Z}_N$, compute:

$$c_i(t) = \mathrm{E}(w_i)^{x_i(t)} \cdot \mathrm{E}(s_i(t)) = \mathrm{E}(w_i x_i(t) + s_i(t)).$$

- $\mathrm{AggrDec}(\mathrm{prm}, \mathrm{sk}_a, t, \{c_i(t)\}_{i\in[M]})$: compute $V(t) = \prod_{i\in[M]} c_i(t) \bmod N^2$ and then set:

$$x_a(t) = \big(\mathrm{D}(V(t)) + s_a(t)\big) \bmod N.$$

Correctness: $\mathrm{D}(V(t)) = \sum_{i\in[M]} w_i x_i(t) + s_i(t)$ follows from the correct execution of Paillier operations in Enc. Then, $\mathrm{D}(V(t)) + s_a(t) = \sum_{i\in[M]} w_i x_i(t) \bmod N = x_a(t)$ from the generation of shares of zero.

Theorem 4.3.2. *The* pWSAh *scheme achieves weighted sum aggregator obliviousness w.r.t. Definition 4.2.1.*

The proof is given in Section C.1.

Remark 4.3.3. Unlike in pSA$_2$, in pWSAh the aggregator has to know the secret key of the cryptosystem that encrypts the weights. If we would use pSA$_2$, which has a single share of zero per agent for all time steps, an adversary that corrupts the aggregator and selects equal contributions at different time steps for an agent in the pWSAO game (described in Section 4.2) could learn that agent's secret share.

The above scheme is appealing due to its simplicity, but involves demanding communication, because secret shares of zero are required at every time step t for every participant, as motivated by Remark 4.3.3. The Setup is executed by an incorruptible trusted third-party, called dealer. This dealer cannot be online at every time step to distribute the shares because, otherwise, this party could act as a trusted aggregator. A more reasonable assumption is that, prior to the online computations, the dealer computes the shares for T time steps and sends them to the agents, who have to store them. Alternatively, we also offer a solution to generate the secret shares of zero in a distributed way, without the need of a trusted third-party.

4.3.3 Decentralized generation of zero shares

4.3.3.1 One communication round, lower collusion threshold

We first describe a solution with one round of communication but lower collusion threshold. This solution is appealing in the case where the agents are connected by a dense graph. Specifically, at each time step, each agent would generate and send shares to the agents in its neighborhood (including the aggregator), then sum up the shares it received from its neighbors. This guarantees that all participants will hold a share of zero. However, if the communication graph between the agents is sparse, the collusion threshold drops from $M-1$ participants to the minimum number of neighbors that an agents has.

The scheme for distributedly generating shares of zero for the update computed at agent i for time t has the following steps:

1. At time $t-1$, each agent $j \in \mathcal{N}_i \cup i$ sends shares of zero $\sigma^i_{jl}(t) \in \mathbb{Z}^*_N$ to itself and to the agents in the intersection of its neighbors and the neighbors of agent i:

$$\sum_{l \in (\mathcal{N}_j \cup j) \cap (\mathcal{N}_i \cup i)} \sigma^i_{jl}(t) = 0 \bmod N. \tag{4.3.2}$$

2. At time t, each agent $j \in \mathcal{N}_i \cup i$ sums its own share and the shares it received meant for the aggregation at i:

$$s_{ij}(t) := \sum_{l \in (\mathcal{N}_j \cup j) \cap (\mathcal{N}_i \cup i)} \sigma^i_{lj}(t). \tag{4.3.3}$$

From (4.3.2), it is clear that: $\sum_{j \in \mathcal{N}_i \cup i} \sum_{l \in (\mathcal{N}_j \cup j) \cap \mathcal{N}_i} \sigma^i_{jl}(t) = 0 \bmod N$. Then, we confirm that we obtained shares of zero for agent i and $j \in \mathcal{N}_i$:

$$\sum_{j \in \mathcal{N}_i \cup i} s_{ij}(t) = 0 \bmod N.$$

The idea is inspired from the dining cryptographers problem [59].

Furthermore, the bandwidth overhead can be reduced if, instead of sending the full secret $\sigma \in \mathbb{Z}_N^*$, the agents send only a smaller seed $\tau \in \mathbb{Z}_w$, $w << N$ for a pseudorandom generator function (e.g. a hash function) $H : \mathbb{Z}_w \to \mathbb{Z}_N^*$, that is publicly known. The above scheme can be modified as follows:

1. At time $t-1$, each agent $j \in \mathcal{N}_i \cup i$ generates random seeds for each agent $l \in \mathcal{N}_j \cap (\mathcal{N}_i \cup i)$: $\tau^i_{jl}(t) \in \mathbb{Z}_w$, and computes for itself:

$$\sigma^i_{jj}(t) = - \sum_{l \in \mathcal{N}_j \cap (\mathcal{N}_i \cup i)} H\big(\tau^i_{jl}(t)\big) \bmod N.$$

2. At time t, each agent $j \in \mathcal{N}_i \cup i$ sums the outputs of the hash function on the seeds it received from its neighbors that participated in the computation and its own share:

$$s_{ij}(t) := \sum_{l \in \mathcal{N}_j \cap (\mathcal{N}_i \cup i)} H\big(\tau^i_{lj}(t)\big) + \sigma^i_{jj}(t).$$

For the centralized solution of creating shares of zero considered in Section 4.3.2, the privacy of one agent $j \in \mathcal{N}_i \cup i$ is guaranteed as long as the number of colluding agents is strictly less than $|\mathcal{N}_i \cup i| - 1$ (otherwise the share of agent j can be computed from the shares of the colluding agents). For the decentralized solution, the privacy of one agent $j \in \mathcal{N}_i \cup i$ is guaranteed as long as the number of colluding agents is strictly less than $(\mathcal{N}_j \cup j) \cap (\mathcal{N}_i \cup i)$. Hence, this scheme is more robust the more neighbors an agent has.

In summary, the above solution has the advantage that the agents communicate their shares in a decentralized fashion, but the disadvantage that the number threshold of colluding parties that can guess a secret share is reduced from the cardinality of the neighbors of the aggregating agent to the cardinality of the smallest intersection of the neighborhood of the aggregator's neighbors and the aggregator's neighborhood.

4.3.3.2 Two communication rounds, same collusion threshold

When agents are not sufficiently connected, there are ways of remediating the problem, each with different trade-offs. If we are able to enforce new communication links between the agents, we can create dummy connections such that each agent reaches a desired vertex degree. This keeps the same number of communication rounds as before, but it is debatable whether the cost of adding new communication links is reasonable. For instance, if the connections are based on proximity, such a solution is expensive. On the other hand, if we relax the number of communication rounds such that each agent obtains a valid share of zero in two rounds, we can retrieve the initial collusion threshold of $M-1$ participants. Instead of sending the shares to each other, the agents will use the aggregator as an intermediate relay to get to the agents that they are not directly connected to. Specifically, each agent $i \in [M]$ will generate $M+1$ shares and encrypt them with a key known the agent $l \in ([M] \setminus i) \cup a$ (with, e.g., a symmetric encryption like AES). The aggregator will forward the corresponding shares to its neighbors $l \neq i$.

1. At time $t-2$, each agent and the aggregator $i \in [M] \cup a$ creates shares of zero $\sigma_{il}(t) \in \mathbb{Z}_N^*$ for itself and for the rest of the agents:

$$\sum_{l \in [M] \cup a} \sigma_{il}(t) = 0 \bmod N. \tag{4.3.4}$$

   It encrypts them with a key known to agent $l \in [M] \cup a \setminus i$ and sends $\mathrm{AES}(\mathrm{key}_l, \sigma_{il}(t))$ to the aggregator.

2. At time $t-1$, the aggregator batches the M shares for agent $i \in [M]$ and sends them.

3. At time t, each agent $l \in [M] \cup a$ sums its own share and the shares it received and

decrypted from the aggregator:

$$s_l(t) := \sum_{i\in[M]\cup a} \sigma_{il}(t) \bmod N. \tag{4.3.5}$$

In order to reduce the load on the aggregator, an agent i can communicate directly to agents $l \in \mathcal{N}_i \cap [M]$, where $\mathcal{N}_i$ is the set of neighbors of agent i and only sends the encrypted shares to the aggregator for the agents $l \notin \mathcal{N}_i \cap [M]$.

A technical discussion on sampling the shares of zero is given in Appendix C.2.

4.4 Schemes for multi-dimensional data

4.4.1 Packed Paillier scheme

Assume we have a vector $\mathbf{y} = [\mathbf{y}^{[1]}, \mathbf{y}^{[2]}, \ldots, \mathbf{y}^{[m]}]$, with $\mathbf{y}^{[i]} \in [0,\ 2^l) \cap \mathbb{Z}_{\geq 0}$. We can take advantage of the fact that $N >> 2^l$, where N is the Paillier modulus by packing m items of l bits into a plaintext in $\mathbb{Z}_N$ in the following way:

$$p_y = \sum_{i=1}^{m} \mathbf{y}^{[i]} 2^{l(i-1)} = [\mathbf{y}^{[1]}|\mathbf{y}^{[2]}|\ldots|\mathbf{y}^{[m]}].$$

Here, we depict the least significant bits on the left, to show the elements in the order that they appear in the vector. If we need to perform additional operations on p_y after packing, we have to make sure we retrieve the correct elements. Hence, we need to take into account possible overflows from one "slot" of the ciphertext the another. We do this by padding with enough zeroes, where $\delta > l$ and will be determined based on the computations performed on p_y afterwards:

$$p_y = \sum_{i=1}^{m} \mathbf{y}^{[i]} 2^{\delta(i-1)} = [\underbrace{\mathbf{y}^{[1]}}_{l}\underbrace{0...0}_{\delta-l}|\underbrace{\mathbf{y}^{[2]}}_{l}\underbrace{0...0}_{\delta-l}|\ldots|\underbrace{\mathbf{y}^{[m]}}_{l}\underbrace{0...0}_{\delta-l}]. \tag{4.4.1}$$

Note that we can perform (4.4.1) as long as $m\delta < N$.

In (4.4.1), we require positive integers. For a value $y \in \mathbb{Q}_{l_i,l_f}$, we first construct an integer $\overline{y}$, and then obtain a positive integer $\tilde{y}$, for $\gamma > l := l_i + l_f$ that we will specify later:

$$\overline{y} := y2^{l_f} \Rightarrow \overline{y} \in [-2^{l-1}, 2^{l-1}) \cap \mathbb{Z} \tag{4.4.2}$$

$$\tilde{y} := \overline{y} + 2^{\gamma} \Rightarrow \tilde{y} \in [0, 2^{\gamma+1}) \cap \mathbb{Z}_{\geq 0}. \tag{4.4.3}$$

In Sections 4.4.2.2 and 4.4.2.4, where we do not use packing, instead of (4.4.3) we use (assuming $2^{l-1} < N/2$):

$$\tilde{y} := \begin{cases} \bar{y} & \text{if } \ \bar{y} \geq 0 \\ \bar{y} + N & \text{if } \ \bar{y} < 0 \end{cases} \quad \Rightarrow \quad y \in \mathbb{Z}_N. \tag{4.4.4}$$

Batching multiple entries into one Paillier ciphertext was first proposed in [100]. Notice that after packing multiple plaintexts into one Paillier ciphertext as in (4.4.1), we can still perform the homomorphisms corresponding to element-wise addition and scalar multiplication. In the following, we investigate a more efficient way to compute an encrypted matrix-plaintext vector multiplication, by using only packing, element-wise addition and scalar multiplication. For a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, denote the jth column by $\mathbf{w}^j$, for $j \in [n]$. Then, in order to obtain the product $\mathbf{v} := \mathbf{W}\mathbf{x}$, we multiply each column $\mathbf{w}^j$ by the corresponding element in the vector $\mathbf{x}^{[j]}$ and then sum over all the obtained vectors:

$$\mathbf{v} = \sum_{j=1}^{n} \mathbf{w}^j \mathbf{x}^{[j]}. \tag{4.4.5}$$

Figure 4.3 depicts this method.

4.4.2 Multi-dimensional private weighted sum aggregation

4.4.2.1 Multi-dimensional pSA

In the case of pSA_1, the messages have small sizes and communication is less of a problem even for multi-dimensional data. However, in the case of pSA_2, messages (ciphertexts) are larger and we propose a better method than sending a different message for each element of

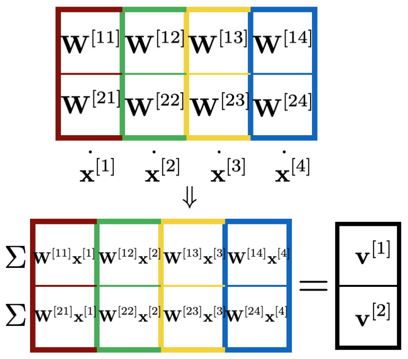

$\mathbf{W}^{[11]}$
$\mathbf{W}^{[12]}$
$\mathbf{W}^{[13]}$
$\mathbf{W}^{[14]}$
$\mathbf{W}^{[21]}$
$\mathbf{W}^{[22]}$
$\mathbf{W}^{[23]}$
$\mathbf{W}^{[24]}$
$\mathbf{x}^{[1]}$
$\mathbf{x}^{[2]}$
$\mathbf{x}^{[3]}$
$\mathbf{x}^{[4]}$
$\sum$
$\mathbf{W}^{[11]}\mathbf{x}^{[1]}$
$\mathbf{W}^{[12]}\mathbf{x}^{[2]}$
$\mathbf{W}^{[13]}\mathbf{x}^{[3]}$
$\mathbf{W}^{[14]}\mathbf{x}^{[4]}$
$\mathbf{W}^{[21]}\mathbf{x}^{[1]}$
$\mathbf{W}^{[22]}\mathbf{x}^{[2]}$
$\mathbf{W}^{[23]}\mathbf{x}^{[3]}$
$\mathbf{W}^{[24]}\mathbf{x}^{[4]}$
$\mathbf{v}^{[1]}$
$\mathbf{v}^{[2]}$

Figure 4.3: Diagram of column-packed matrix-vector multiplication. The entries with the same outer coloring are packed in the same ciphertext.

the resulting vector, by batching the elements in a single ciphertext. The aggregator wants to obtain $\mathbf{x}_a(t) = \sum_{i\in[M]} \mathbf{W}_i\mathbf{x}_i(t) =: \sum_{i\in[M]} \mathbf{v}_i(t)$. In pSA$_2$, the dealer generates the secret shares the same way as previously, but each agent $i \in [M]$ computes $\mathbf{v}_i^{[j]}(t)$ and uses (4.4.2) and (4.4.3) to obtain $\tilde{\mathbf{v}}_i^{[j]}(t)$, then computes:

$$p_i(t) = \sum_{j\in[n_i]} \tilde{\mathbf{v}}_i^{[j]}(t)2^{\delta(j-1)}, \quad c_i(t) = (1+N)^{p_i(t)}H(t)^{s_i} \bmod N^2.$$

The aggregator computes $V(t)$ as before, and retrieves:

$$\tilde{\mathbf{x}}_a^{[k]} = V(t)//2^{(n_a-k)\delta} \bmod 2^{(k-1)\delta}, \quad k \in \{2,\dots,n_a-1\}$$

and $\tilde{\mathbf{x}}_a^{[1]} = V(t) \bmod 2^\delta$, $\tilde{\mathbf{x}}_a^{[n_a]} = V(t)//2^{(n_a-1)\delta}$, where by // we mean the quotient operation. From the elements of $\tilde{\mathbf{x}}_a(t)$, the aggregator needs to subtract $2^\gamma M$ and divide by 2^{2l_f} each element, in order to obtain $\mathbf{x}_a(t)$.

Choosing $\gamma = 2l+1$ and $\delta = 2l+2+\lceil\log_2 M\rceil$ ensures the correctness of the decryption, as no overflow occurs.

4.4.2.2 Multi-dimensional pWSAc

Here, we cannot use packing to reduce the number of ciphertexts because we would require rotations and element-wise multiplications, which cannot be performed on packed Paillier ciphertexts. Compared to the pWSAh scheme, pWSAc has the advantage that only one set of secret shares are needed for all time steps, hence, communication due to secret generation and sharing only happens once.

In the multi-dimensional case, the algorithms change from the ones in Section 4.3.1.1 as described next. The participants prepare their data using (4.4.2) and (4.4.4). In Setup, $n_a M$ secrets $\mathbf{s}_i \in (\mathbb{Z}_N^*)^{n_a}$ are generated and:

$$\mathbf{s}_a^{[k]} = -\sum_{i\in[M]}\sum_{j\in[n_i]} \mathbf{W}_i^{[kj]}\mathbf{s}_i^{[j]}, \quad k \in [n_a].$$

In Enc, each agent $i \in [M]$ constructs n_a ciphertexts:

$$\mathbf{c}_i^{[j]}(t) = (1 + \tilde{\mathbf{x}}_i^{[j]}(t)N) \cdot H(t)^{\mathbf{s}_i^{[j]}} \bmod N^2, \quad j \in [n_i].$$

Finally, in AggrDec, the aggregator computes for $k \in [n_a]$:

$$\mathbf{V}^{[k]}(t) = H(t)^{\mathbf{s}_a^{[k]}} \cdot \prod_{i\in[M]}\prod_{j\in[n_i]} \mathbf{c}_i^{[j]}(t)^{\tilde{\mathbf{W}}_i^{[kj]}} \bmod N^2, \qquad \tilde{\mathbf{x}}_a^{[k]}(t) = \left(\mathbf{V}^{[k]}(t) - 1\right)/N.$$

The aggregator retrieves the elements of $\mathbf{x}_a(t)$ from $\tilde{\mathbf{x}}_a(t)$ by subtracting N from the elements greater than $N/2$ and dividing all of them by 2^{2l_f}.

4.4.2.3 Multi-dimensional pWSAh

We consider values on l bits, with $\log_2 N >> l$, for N ensuring semantic security of the Paillier scheme. Sampling a random value from a large $\mathbb{Z}_N$ is expensive, but also redundant, since it masks a much smaller message. To this end, we prefer to sample $s_i(t) \in (0, 2^{\lambda+2l}), \forall i \in [M]$, for λ the statistical security parameter and to set $s_a(t) := -\sum_{i\in[M]} s_i(t)$ in pWSAh. Masking by $s_i(t)$ will guarantee λ-statistical security rather than perfect security, see Preamble 2.4. From here on, we use this more efficient approach.

4.4.2.4 Naive multi-dimensional scheme

This solution was proposed in [12]. The algorithms change compared to Section 4.3.2 as follows. In Setup, for every $t \in [T]$, $M \cdot n_a$ shares of zero $\mathbf{s}_i^{[k]}(t) \in (0, 2^{\lambda+2l})$ are generated for $i \in [M], k \in [n_a]$:

$$\mathbf{s}_a^{[k]}(t) = -\sum_{i\in[M]}\sum_{k\in[n_a]} \mathbf{s}_i^{[k]}(t).$$

In InitW, the weights $\mathbf{W}_i$, $i \in [M]$ are processed as in (4.4.2) and (4.4.4) and encrypted element-wise: $\mathrm{E}(\mathbf{W}_i^{[kj]}) = (1+N)^{\tilde{\mathbf{W}}_i^{[kj]}} r^N \bmod N^2$, for r randomly sampled from $\mathbb{Z}_N$ and satisfying $\gcd(r, N) = 1$. In Enc, each agent $i \in [M]$ computes:

$$\mathbf{c}_i^{[k]}(t) = \prod_{j\in[n_i]} \mathrm{E}(\tilde{\mathbf{W}}_i^{[kj]})^{\tilde{\mathbf{x}}_i^{[j]}(t)} \cdot \mathrm{E}(\mathbf{s}_i^{[k]}(t)) = \mathrm{E}\left(\sum_{j\in[n_i]} \mathbf{W}_i^{[kj]}\mathbf{x}_i^{[j]}(t) + \mathbf{s}_i^{[k]}(t)\right), \forall k \in [n_a].$$

Finally, in AggrDec, the aggregator computes, for $k \in [n_a]$:

$$\mathbf{V}^{[k]}(t) = \prod_{i\in[M]} \mathbf{c}_i^{[k]}(t) \bmod N^2, \qquad \tilde{\mathbf{x}}_a^{[k]}(t) = \mathrm{D}(\mathbf{V}^{[k]}(t)) + \mathbf{s}_a^{[k]}(t).$$

4.4.2.5 Packed multi-dimensional scheme

We reduce the number of ciphertexts and the corresponding number of operations by using packing and the more efficient encrypted matrix-plaintext vector multiplication described in Section 4.4.1.

Assume at the moment that we can pack at least n_a values in one ciphertext. The steps we take are: 1) Pre-process the values to be positive and integer; 2) Pack and encrypt the columns of the matrix $\mathbf{W}_i$ and obtain n_i ciphertexts; 3) Perform a scalar multiplication of one encrypted column c with the scalar $\mathbf{x}_i^{[c]}(t)$; 4) Sum the n_i ciphertexts to get the encryption of $\mathbf{W}_i\mathbf{x}_i(t)$; 5) Add the share of zero and mask the intermediate results; 6) Sum the M ciphertexts to obtain the encryption of $\sum_{i\in[M]} \mathbf{W}_i\mathbf{x}_i(t)$; 7) Decrypt, unmask and unpack the result.

Figure 4.4 indicates possible values for the number of rows, columns and agents depending on the plaintext size and statistical security. Denote the maximum number of values we can pack by $m < N/\delta$. If $n_a > m$, we split the columns into $\lceil n_a/m \rceil$ Paillier ciphertexts and follow the same operations as before, and concatenate the resulting vectors after decryption.

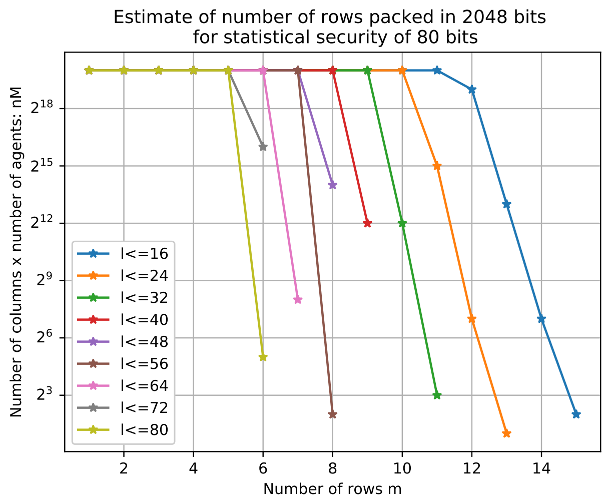


Figure 4.4: The number of rows m that can be packed in a plaintext of $N = 2048$ bits, as a function of the number of columns n, number of agents M, message size l and statistical security $\lambda = 80$ bits.

Let $\text{pWSAh}^* = (\text{Setup}, \text{InitW}, \text{Enc}, \text{AggrDec})$ be a packed multi-dimensional scheme for private weighted sum aggregation with hidden weights, where steps 1) and 2) are performed by the dealer as part of InitW and the shares of zero for step 5) are generated as part of Setup, steps 1), 3)–5) are performed by each agent $i \in [M]$ in Enc and steps 6) and 7) are performed by the aggregator in AggrDec.

The steps and how to choose bit sizes in order to guarantee both correctness and privacy, as well as the proof of the following theorem, are detailed in Appendix C.3.

Theorem 4.4.1. *The packed multi-dimensional scheme* pWSAh* *is correct and achieves aggregator obliviousness.*

4.4.3 Comparison between naive and packed method

In the naive version of the multi-dimensional pWSAh, each agent receives $n_a n_i$ ciphertexts for $\mathbf{W}_i$ at the initialization of the protocol, then computes $n_a n_i$ ciphertext–scalar multiplications (modular exponentiation), $n_a(n_i - 1)$ ciphertext additions (modular multiplications) and sends to the aggregator n_a ciphertexts. In the decentralized way of generating shares, each agent will have to send out $(2l + \lambda)n_a$ bits of randomness to each of the M neighbors per time step.

Denote by m the number of elements we can pack in a Paillier ciphertext. In the packed version pWSAh*, each agent receives $\lceil n_a/m \rceil n_i$ ciphertexts for $\mathbf{W}_i$ at the initialization of the protocol, then computes $\lceil n_a/m \rceil n_i$ ciphertext–scalar multiplications, $\lceil n_a/m \rceil (n_i - 1)$ ciphertext additions and sends to the aggregator $\lceil n_a/m \rceil$ ciphertexts. In the decentralized way of generating shares, each agent will have to send out $(2l + 1 + \lceil n \rceil + \lceil M \rceil)\lceil n_a/m \rceil$ bits of randomness to each of the M neighbors per time step.

4.5 LWE-based private weighted aggregation with hidden weights

To avoid the trust and communication issues introduced by successive symmetric keys, as described in the previous section, we use a public-key additively homomorphic cryptosystem (as the inner encryption scheme) to encapsulate the message in a Learning with Errors ciphertext (the outer encryption scheme). A pSA scheme based on this idea was proposed in [28] using the Augmented Learning with Errors concept introduced in [88]. We show how to modify the scheme in [28] such that we obtain a private weighted sum aggregation scheme and exploit the structure of the problem in order to reduce the number of ciphertexts communicated and number of operations performed.

The LWE problem essentially amounts to distinguishing random linear equations perturbed by small amounts of noise from uniform linear equations. Let κ denote the security parameter, $q = q(\kappa)$ a positive prime, $l = \lceil \log q \rceil$ and λ a positive integer, such that $\lambda/l \in \mathbb{Z}$. The *Augmented Learning with Errors* (A-LWE) problem [88] encodes a message in the error from an LWE term. An A-LWE term consists of $(\mathbf{A}, \mathbf{b}^\intercal)$, with $\mathbf{b}^\intercal = \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal \in \mathbb{Z}_q^\lambda$, for a public matrix $\mathbf{A} \in \mathbb{Z}_q^{\kappa \times \lambda}$, a secret key $\mathbf{s} \in \mathbb{Z}_q^\kappa$ and error term $\mathbf{e} \in \mathbb{Z}_q^\lambda$, which is sampled from a distribution related to the message we want to encode.

Specifically, let μ be a message and $f(\cdot)$ a function with the property that its output is indistinguishable from random (e.g., an encryption mechanism from a semantically secure scheme). For $\mathbf{y} = f(\mu) \in \mathbb{Z}_q^{\lambda/l}$ and for a public matrix $\mathbf{G} \in \mathbb{Z}_q^{\lambda/l \times \lambda}$, the error term $\mathbf{e}$ is sampled from a special error distribution, defined by $D_{\Lambda_{\mathbf{y}}^{\perp}(\mathbf{G}),\sigma}$, and satisfies $\mathbf{Ge} \equiv \mathbf{y} \bmod q$. The special error distribution $D_{\Lambda_{\mathbf{y}}^{\perp}(\mathbf{G}),\sigma}$ is a discrete Gaussian distribution with standard deviation σ over a lattice determined by the matrix $\mathbf{G}$ and vector $\mathbf{y}$, see Definition 4.5.1.

Informally, given an A-LWE term $(\mathbf{A}, \mathbf{b}^{\intercal})$, one cannot retrieve the secret key $\mathbf{s}$ and the message μ encoded in the error term $\mathbf{e}$, but, given $\mathbf{s}$, one can efficiently recover μ from $\mathbf{e}$, see Definition 4.5.2.

For the inner layer of encryption, we require a semantically secure public-key additively homomorphic encryption that allows plaintext-ciphertext multiplication and the operator on the ciphertext space corresponding to addition is also addition. We will call such a scheme Packed Additively Homomorphic Encryption (PAHE) for reasons described in Section 4.6. For the moment, we give a bare-bones description, just to specify the compatibility with the A-LWE outer ciphertext: $\mathcal{S}\text{etup}() \to \text{prm}$; $\mathcal{K}\text{ey}\mathcal{G}\text{en}(\text{prm}) \to (\mathbf{pk}, \mathbf{sk})$; $\mathcal{E}_{\mathbf{pk}}(\mu) \to \mathbf{c}$; $\mathcal{D}_{\mathbf{sk}}(\mathbf{c}) \to \mu$; and $\mathcal{E}\text{val}$, composed of: $\mathcal{A}\text{dd}(\mathbf{c}_1, \mathbf{c}_2) \leftarrow \mathbf{c} \equiv \mathbf{c}_1 + \mathbf{c}_2$; $\mathcal{CM}\text{ult}(p_1, \mathbf{c}_2) \leftarrow \mathbf{c} \equiv p_1 \cdot \mathbf{c}_2$.

There is an encoding step in the encryption primitive that transforms the given message into an appropriate plaintext and a decoding step in the decryption primitive that transforms the obtained plaintext into a message from the desired domain. There exist transformations between the ciphertext space, which is a ring of polynomials, and $\mathbb{Z}_q^{\lambda/l}$. So, for simplicity, we say $\mathbf{c} \in \mathbb{Z}_q^{\lambda/l}$.

Definition 4.5.1. [A-LWE distribution] Let κ, q, p, λ be integers and $l := \lceil \log q \rceil$. Let $\mu \in \mathbb{Z}_p$ be a plaintext and $f : \mathbb{Z}_p \to \mathbb{Z}_q^{\lambda/l}$ be a function with output indistinguishable from random. Define $\mathbf{g}^{\intercal} := \begin{bmatrix} 1 & 2 & \dots & 2^{l-1} \end{bmatrix} \in \mathbb{Z}_q^l$ and $\mathbf{G} := \mathbf{I}_{\lambda/l} \otimes \mathbf{g}^{\intercal} \in \mathbb{Z}_q^{\lambda/l \times \lambda}$. Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{\kappa}$ and $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{\kappa \times \lambda}$. A sample from the Augmented Learning with Errors distribution $L_{\kappa,\lambda,q}^{\text{A-LWE}}(\mu)$ over $\mathbb{Z}_q^{\kappa \times \lambda} \times \mathbb{Z}_q^{\lambda}$ is obtained as follows: Compute $\mathbf{y} := f(\mu) \in \mathbb{Z}_q^{\lambda/l}$; Sample $\mathbf{e} \leftarrow D_{\Lambda_{\mathbf{y}}^{\perp}(\mathbf{G}),\sigma} \in \mathbb{Z}_q^{\lambda}$; Return $(\mathbf{A}, \mathbf{b}^{\intercal})$, with $\mathbf{b}^{\intercal} := \mathbf{s}^{\intercal}\mathbf{A} + \mathbf{e}^{\intercal}$.

Let $\rho : \mathbb{R} \to (0, 1]$ with $\rho_\sigma(x) = \exp(-x^2/\sigma^2)$. The discrete Gaussian distribution over the integers $D_{\mathbb{Z},\sigma}$ samples $x \in \mathbb{Z}$ with probability $\rho_\sigma(x)/(\sum_{y \in \mathbb{Z}} \rho_\sigma(y))$. An efficient sampling algorithm of the error term $\mathbf{e}$ from the discrete Gaussian distribution $D_{\Lambda^{\perp}_{\mathbf{y}}(\mathbf{G}),\sigma}$ for a general q, is given by [101]. We remark that we can also use another basis instead of 2 for $\mathbf{g}$.

Definition 4.5.2. [Decisional A-LWE problem] Let κ, q, p, λ be integers and let f be a function with output indistinguishable from random. The decisional A-LWE problem asks to distinguish in polynomial time poly(κ) between samples $(\mathbf{A}, \mathbf{b}^{\intercal}) \leftarrow L^{\text{A-LWE}}_{\kappa,\lambda,q}(\mu)$ and uniform random samples $(\bar{\mathbf{A}}, \bar{\mathbf{b}}^{\intercal}) \overset{\$}{\leftarrow} \mathbb{Z}_q^{\kappa\times\lambda} \times \mathbb{Z}_q^{\lambda}$.

If the distribution of $\mathbf{y}$ is computationally indistinguishable from the uniform distribution on the same domain, i.e., PAHE is semantically secure, then the decisional A-LWE problem is hard [28, 88].

Solution of pWSAh problem

The idea of the scheme is as follows. The aggregator generates a pair of PAHE keys. A third party (responsible also for choosing the weights) encrypts the weights with the public key and generates a set of $M+1$ random keys that sum to zero, then distributes them accordingly to the agents and aggregator. Each agent computes the product of the encrypted weight with its local data, which is possible due to the homomorphic properties of the PAHE cryptosystem. Then, it samples the error term according to its local PAHE ciphertext, creates an A-LWE ciphertext with its local key and sends it to the aggregator. The aggregator sums all the A-LWE ciphertexts and obtains the sum of the error terms, which is an encoding of the sum of the PAHE ciphertexts. Using its PAHE secret key, the aggregator proceeds to decrypt and obtain the desired weighted sum of the data of the agents in the network.

- Setup$(1^{\kappa}, M, \lambda, q, \sigma, w_{1,\dots,M}, T)$: Generate the public parameters $\mathbf{A}_t \overset{\$}{\leftarrow} \mathbb{Z}_q^{\kappa\times\lambda}$, for time steps $t = 1, \dots, T$. Generate $(\mathbf{pk}, \mathbf{sk}) \leftarrow \mathcal{K}ey\mathcal{G}en$. For all agents $i \in [M]$, draw $\mathbf{s}_i \overset{\$}{\leftarrow} \mathbb{Z}_q^{\kappa}$ and let $\mathbf{s}_a = -\sum_{i\in[M]} \mathbf{s}_i$. Broadcast public parameters prm $= \big(\mathbf{A}_{1,\dots,T}, q, \kappa, \sigma, \lambda, M, \mathbf{pk}\big)$. To each agent $i \in [M]$, send their secret key $\mathbf{s}_i$. Send $\big(\mathbf{s}_a, \mathbf{sk}\big)$ to the aggregator. Each agent and the aggregator computes the vector $\mathbf{g}^{\intercal} = \begin{bmatrix} 1 & 2 & \dots & 2^{l-1}\end{bmatrix} \in \mathbb{Z}_q^{l}$. The aggregator also computes $\mathbf{G} = \mathbf{I}_{\lambda/l} \otimes \mathbf{g}^{\intercal} \in \mathbb{Z}_q^{\lambda/l\times\lambda}$.

- $\text{InitW}(\text{prm}, \{w_i\}_{i\in[M]})$: given the public key of the PAHE scheme $\mathbf{pk}$, encrypt w_i for $i \in [M]$ and return $\mathbf{sw}_i = \mathcal{E}_{\mathbf{pk}}(w_i)$.

- $\text{Enc}(\mathbf{A}_t, \mathbf{g}^\intercal, \mathbf{pk}, \sigma, \mathbf{s}_i, x_i(t), \mathbf{sw}_i)$: Each agent i computes the chiphertext $\mathbf{y}_i(t) = \mathcal{CM}\text{ult}(\mathbf{sw}_i, x_i(t)) \in \mathbb{Z}_q^{\lambda/l}$ and samples the noise term $\mathbf{e}_i(t) \leftarrow D_{\Lambda^\perp_{\mathbf{y}_i(t)}(\mathbf{G}),\sigma} \in \mathbb{Z}_q^\lambda$. Finally, it computes $\mathbf{c}_i(t) = \mathbf{s}_i^\intercal \mathbf{A}_t + \mathbf{e}_i(t)^\intercal \in \mathbb{Z}_q^\lambda$ and sends the ciphertext to the aggregator.

- $\text{AggrDec}(\mathbf{A}_t, \mathbf{G}, \mathbf{s}_a, \mathbf{sk}, \mathbf{c}_{1,\ldots,M}(t))$: The aggregator sums the ciphertexts from all the agents $\mathbf{c}(t) = \sum_{i\in[M]} \mathbf{c}_i(t)$. It then computes the aggregated error term $\mathbf{e}(t) = \mathbf{c}(t) + \mathbf{s}_a^\intercal \mathbf{A}_t$. Finally, the aggregated sum of the agents' data is computed as $\mathbf{x}_a(t) = \mathcal{D}_{\mathbf{sk}}(\mathbf{G}\mathbf{e}(t) \bmod q)$.

Theorem 4.5.3. *The* pWSA *scheme achieves weighted sum aggregator obliviousness w.r.t. Definition 4.2.1.*

The proof for Theorem 4.5.3 and the correctness of the scheme are given in Section C.4.

Compared to the solution in Section 4.3.2, only one set of secret shares of zero $\mathbf{s}_a = -\sum_{i\in[M]} \mathbf{s}_i$ have to be generated for all the time steps. In this case, it is reasonable to expect a trusted third party to generate and distribute them offline or even for the agents and aggregator to embark in an offline secure multi-party computation algorithm to obtain them.

For the security of the scheme over multiple time steps, we need to use different matrices $\mathbf{A}$ for each time step t, otherwise the aggregator can obtain differences of messages at two time steps. These matrices $\mathbf{A}_t$ are public and can be broadcasted or posted on a board of messages, where each participant has access to. For better efficiency, only smaller random seeds can be sent and stored, e.g., agree on a function that outputs a pseudorandom matrix and feed in the time steps and a smaller seed. Only one seed $\mathfrak{s}_0$ should be sent at time zero, then the agents can construct the seed for time t in a counter block cipher mode $\mathfrak{s}_t = \mathfrak{s}_0 + t$.

4.6 Packed weighted aggregation with A-LWE

In this section, we show how to extend in an efficient way the scalar weighted sum aggregation scheme to multi-dimensional data. Compared to the solution in 4.4.2.3, this scheme uses ciphertexts that allow packing a significantly larger number of values.

We require the packed additively homomorphic encryption scheme (PAHE) scheme to be semantically secure and to satisfy the requirement from Section 4.5 of ciphertext summation. This scheme should also allow packing and single instruction multiple data (SIMD) operations. We draw inspiration from the packed additively homomorphic encryption scheme used in [132]. PAHE can be instantiated by schemes in e.g., [51, 62]; see Preamble 2.5.3. The underlying hardness problem is Ring Learning with Errors (R-LWE in Appendix A.2). The PAHE construction is parameterized by the following constants: the ring dimension $2N$, the plaintext modulus p, the ciphertext modulus q and the standard deviation σ of a discrete Gaussian distribution. We can pack up to N values in one ciphertext using the Chinese Remainder Theorem. Packing can be thought of as the ciphertext having N independent data slots. Using the notation in Section 4.5, $N = \lambda/(2l)$.

Recall the abstraction of the PAHE primitives: $\mathcal{S}$etup, $\mathcal{K}$ey$\mathcal{G}$en, $\mathcal{E}$, $\mathcal{D}$, $\mathcal{E}$val. The operations that can be evaluated during $\mathcal{E}$val are single instruction multiple data, which means that they can operate on the whole encoded vector. Specifically, these operations are SIMD$\mathcal{A}$dd, SIMD$\mathcal{CM}$ult and $\mathcal{R}$otate, i.e. element-wise addition, element-wise multiplication by a plaintext vector and slots permutations that can achieve rotations:

- SIMD$\mathcal{A}$dd$(\mathbf{c}_1, \mathbf{c}_2) \to \mathbf{c}$, such that, if $\mathbf{c}_i = \mathcal{E}_{\mathbf{pk}}(\boldsymbol{\mu}_i)$, $i = 1, 2$, then $\mathcal{D}_{\mathbf{sk}}(\mathbf{c}) = \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2$.
- SIMD$\mathcal{CM}$ult$(\mathbf{c}, \boldsymbol{\nu}) \to \mathbf{c}'$, such that, if $\mathbf{c} = \mathcal{E}_{\mathbf{pk}}(\boldsymbol{\mu})$, then $\mathcal{D}_{\mathbf{sk}}(\mathbf{c}') = \boldsymbol{\mu} \circ \boldsymbol{\nu}$, where $\circ$ denotes element-wise multiplication.
- $\mathcal{R}$otate$_{\mathbf{rk}^{(\pi)}}(\mathbf{c}, \pi) \to \mathbf{c}'$, such that, for a permutation π and $\mathbf{c} = \mathcal{E}_{\mathbf{pk}}(\boldsymbol{\mu})$, we have $\mathcal{D}_{\mathbf{sk}}(\mathbf{c}') = [\boldsymbol{\mu}_{\pi(1)}, \dots, \boldsymbol{\mu}_{\pi(n)}]$, where $\mathbf{rk}$ is the associated rotation key.

The encryption $\mathcal{E}$ endows the ciphertexts with a fresh small noise η_0. The operations in $\mathcal{E}$val also introduce an amount of noise in the ciphertext, which can overflow and prevent the correct decryption. Denote by η the noise level in a ciphertext $\mathbf{c}$. A ciphertext resulted from SIMD$\mathcal{A}$dd has noise $\eta_1 + \eta_2$. A ciphertext resulted from SIMD$\mathcal{CM}$ult has noise bounded by $\eta\eta_\times$, where $\eta_\times \leq p\sqrt{N}$. A ciphertext resulted from $\mathcal{R}$otate has noise $\eta + \eta_\pi$, where η_π is the noise of the permutation operation. The multiplication introduces the largest noise. In terms of computation cost, the addition is the cheapest, while the rotation is the most expensive. The parameters of the scheme need to be chosen such that the noise does not overflow.

4.6.1 Efficient homomorphic matrix-vector multiplication

In this section, we investigate efficient methods for performing the multiplication of an encrypted matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ and plaintext $\mathbf{x} \in \mathbb{R}^n$, i.e., computing the corresponding ciphertext version of $\mathbf{y} = \mathbf{W}\mathbf{x}$. For our target applications of weighted aggregation, the goals of the computation are, in order of importance:

- minimize the size of the encrypted output that contains $\mathbf{y}$, since this ciphertext has to be encoded in the noise term and then sent over to the aggregator;
- minimize the noise growth, in order to reduce the parameter size, which in turn also minimizes both the computational cost and size of the communicated message;
- minimize the computational cost;
- minimize the input ciphertext that packs $\mathbf{W}$.

Sequential SIMD$\mathcal{CM}$ult operations should be avoided because of the exponential growth in the noise. For the input vector packing, we assume that $m, n < N$. For the input matrix packing, we assume that $nm < N$ or $n^2 < N$. We pad the rest of the slots up to N with zeros. If $mn > N$ (or $n^2 > N$), then the number of input ciphertexts will be $\lceil mn/N \rceil$ (respectively $\lceil n^2/N \rceil$). We show in Figure 4.5 the schematic representations of the five methods considered for computing a matrix-vector multiplication.

4.6.1.1 Naive method with each row packed in one ciphertext.

The output of a matrix-vector multiplication is given by:

$$\mathbf{y}_j = \mathbf{W}_j\mathbf{x} = \sum_{k=1}^{n} \mathbf{W}_{jk} \circ \mathbf{x}_k,\ j = 1, \ldots, m. \tag{4.6.1}$$

If each row of the matrix $\mathbf{W}$ is packed and encrypted in a ciphertext, and $\mathbf{x}$ is packed in a plaintext, then the naive version involves m SIMD$\mathcal{CM}$ult operations. For each resulting vector, we then need $n-1$ $\mathcal{R}$otate and $n-1$ SIMD$\mathcal{A}$dd operations, which creates a ciphertext j whose first slot contains $\mathbf{y}_j$. Using a tree structure to perform these operations, we can reduce their number to $\lceil \log n \rceil$ $\mathcal{R}$otate and $\lceil \log n \rceil$ SIMD$\mathcal{A}$dd operations. This creates m output ciphertexts. In order to obtain only one output ciphertext, we need m SIMD$\mathcal{CM}$ult to mask the ciphertexts by $[1|0|0|\ldots]$, then $m-1$ $\mathcal{R}$otate and $m-1$ SIMD$\mathcal{A}$dd operations. This sequence of operations introduces a lot of noise, because of the two sequential SIMD$\mathcal{CM}$ult operations.

4.6.1.2 Method with each column packed in one ciphertext.

Denote by $\mathbf{C}_j$ the j'th column of the matrix $\mathbf{W}$, $j = 1, \ldots, n$.

$$\mathbf{y} = \sum_{j=1}^{n} \mathbf{C}_j\mathbf{x}_j. \tag{4.6.2}$$

If each $\mathbf{C}_j$ is packed and encrypted in a ciphertext, and we pack m copies of $\mathbf{x}_j$ in a plaintext, the product can be achieved by n SIMD$\mathcal{CM}$ult, $n-1$ SIMD$\mathcal{A}$dd operations, and no permutation, while outputting a single ciphertext.

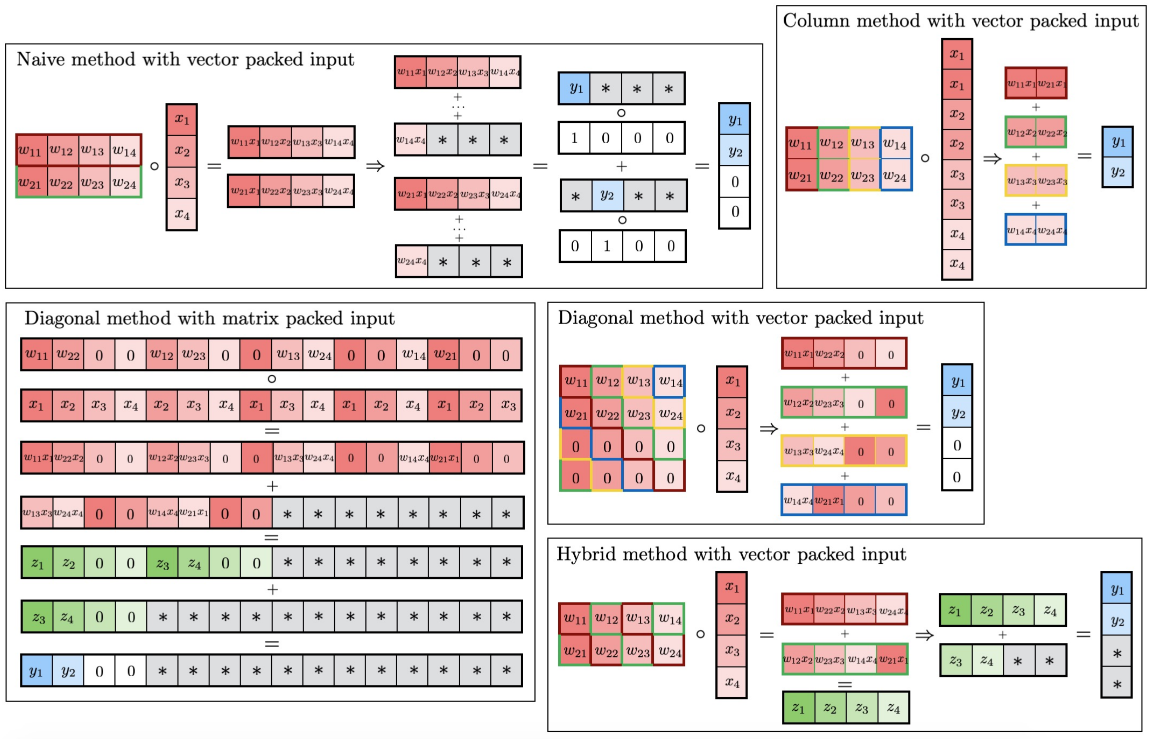
4.6.1.3 Method with each diagonal packed in one ciphertext.

In the applications mentioned in the Introduction, the matrix $\mathbf{W}$ usually satisfies $m \leq n$. We pad the matrix with zeros such that it becomes square $\bar{\mathbf{W}} \in \mathbb{R}^{n \times n}$. We can pack and encrypt every diagonal of the matrix, denoted by $\mathbf{d}_j, j = 1, \ldots, n$ as a separate ciphertext.

Let $\rho(\mathbf{x}, j)$ be the rotation of $\mathbf{x}$ to the left by j elements. Then, we have:

$$\mathbf{y} = \sum_{j=1}^{n} \mathbf{d}_j \circ \rho(\mathbf{x}, j-1). \tag{4.6.3}$$

If we rotate and pack the plaintext vector $\mathbf{x}$, the product can be achieved by n SIMD$\mathcal{CM}$ult, $n-1$ SIMD$\mathcal{A}$dd operations, and no permutation, while outputting a single ciphertext, the same as in the column method.



Naive method with vector packed input
Column method with vector packed input
Diagonal method with matrix packed input
Diagonal method with vector packed input
Hybrid method with vector packed input

Figure 4.5: Diagrams of the various matrix-vector multiplication schemes considered. The entries with the same outer coloring are packed in the same ciphertext. The inner color is selected to aid with visualizing the rotations and the corresponding element-wise slot multiplications.

4.6.1.4 Method with hybrid diagonal packing in a ciphertext.

We can combine the naive and the diagonal method by packing into a ciphertext an "extended diagonal" of the rectangular matrix $\mathbf{W}$, which we denote $\bar{\mathbf{d}}_j$ for $j = 1, \ldots, m$. Then, the product can be written as:

$$\mathbf{y} = \sum_{i=1}^{\lceil n/m \rceil} \rho\Big(\sum_{j=1}^{m} \bar{\mathbf{d}}_j \circ \rho(\mathbf{x}, j-1), m(i-1)\Big). \tag{4.6.4}$$

This requires m SIMD$\mathcal{CM}$ult operations, $\lceil \log n/m \rceil$ $\mathcal{R}$otate and $\lceil \log n/m \rceil + m - 1$ SIMD$\mathcal{A}$dd.

4.6.1.5 Naive method with all matrix packed in one ciphertext.

Consider that $\mathbf{W}$ is packed and encrypted in one ciphertext, row by row. Then we pack m copies of $\mathbf{x}$ in a plaintext and perform one SIMD$\mathcal{CM}$ult operation to get:

$$[\mathbf{W}_1 \circ \mathbf{x} | \mathbf{W}_2 \circ \mathbf{x} | \ldots | \mathbf{W}_m \circ \mathbf{x} | \ldots].$$

We only need to perform $\lceil \log n \rceil$ $\mathcal{R}$otate and $\lceil \log n \rceil$ SIMD$\mathcal{A}$dd operations to obtain: $[\mathbf{y}_1 | * | \ldots | * | \mathbf{y}_2 | * | \ldots | * | \mathbf{y}_m | * | \ldots]$. Notice that this yields only one output ciphertext, and no further masking is required as long as the aggregator knows which slots to retrieve when decrypting.

4.6.1.6 Method with all diagonals packed in one ciphertext.

Consider that the n diagonals of $\bar{\mathbf{W}} \in \mathbb{R}^{n \times n}$ are packed and encrypted in one ciphertext. Then we can also pack the n rotated versions of $\mathbf{x}$ in a plaintext, and then perform one SIMD$\mathcal{CM}$ult operation to get:

$$[\mathbf{d}_1 \circ \mathbf{x} | \mathbf{d}_2 \circ \rho(\mathbf{x}, 1) | \ldots | \mathbf{d}_n \circ \rho(\mathbf{x}, n-1) | \ldots].$$

As before, we only need to perform $\lceil \log n \rceil$ $\mathcal{R}$otate and $\lceil \log n \rceil$ SIMD$\mathcal{A}$dd operations to obtain: $[\mathbf{y}_1 | \mathbf{y}_2 | \mathbf{y}_m | * | * | \ldots]$.

Regardless of how we pack the matrix into one ciphertext (column and hybrid packing too), the computational cost and noise are the same. The advantage of using the diagonal/ column input matrix packing is that the elements of the output vector will be in the first m slots of the output ciphertext, rather than spread one every n slots.

Table 4.1 summarizes the number of operations, noise gain and number of input and output ciphertexts of the methods we analyzed. Note that η_0 grows with the number of inputs packed inside a ciphertext. These methods point out a trade-off between memory and computation. The agents have to perform as many multiplications as input ciphertexts (with the exception of the naive method). At the same time, the maximum number of input ciphertexts (n) required in the diagonal/column methods does not require any permutation and has the least amount of noise.

Method	$\mathcal{R}$otate	SIMD $\mathcal{CM}$ult	SIMD$\mathcal{A}$dd	Noise	# In ctx	# Out ctx
Naive	$m\lceil\log n\rceil + m-1$	$2m$	$m\lceil\log n\rceil + m-1$	$m\eta_\times(n\eta_0\eta_\times + (n-1)\eta_\pi) + (m-1)\eta_\pi$	m	1
Diagonal/ Column	0	n	$n-1$	$n\eta_0\eta_\times$	n	1
Hybrid	$\lceil\log n/m\rceil$	m	$\lceil\log n/m\rceil + m-1$	$n\eta_0\eta_\times + \lceil\log n/m-1\rceil\eta_\pi$	m	1
Matrix packed	$\lceil\log n\rceil$	1	$\lceil\log n\rceil$	$n\eta_0\eta_\times + (n-1)\eta_\pi$	1	1

Table 4.1: Table with costs of different methods for computing a ciphertext matrix-plaintext vector multiplication. η_0 represents the noise of the corresponding fresh ciphertext. In the method column, the naive, diagonal/column and hybrid are input vector packed.

Making a decision between the available methods should take into account the agents' capabilities and the sizes of n and m. Note that the weights are constant and transmitted only once at the protocol's initialization, hence the communication overhead for the input transmission is not decisive. For a square matrix $\mathbf{W}$, the diagonal method is the same as the hybrid one and is the default option, as is the column method. For very large n and $n >> m$, the hybrid method is preferable. For large n, m with mn comparable to N, the input matrix packing is preferable.

Remark 4.6.1. When the number of items packed in a ciphertext is less than N and rotations are performed, some slots in the output ciphertext will reveal partial sums to the decryptor. An inexpensive solution (one SIMD$\mathcal{A}$dd) to this issue is to add noise to the slots that are not of interest, in order to prevent information leakage at the decryption. The diagonal and column methods with input vector packing do not require the noise treatment.

4.6.2 Multi-dimensional A-LWE based solution

Fot the pWSAh scheme, the Setup phase unfolds as in Section 4.5: the parameters and keys are generated with respect to the PAHE crypto-system and the weight matrices $\mathbf{W}_i$ are packed and encrypted corresponding to the chosen method from Table 4.1.

In the Enc phase, each agent packs its local vector $\mathbf{x}_i(t)$ in plaintext corresponding to the chosen method from Table 4.1 and performs that instead of CMult. Before sampling $\mathbf{e}_i(t)$ from $\mathbf{y}_i(t)$, the agents add noise as indicated in Remark 4.6.1.

In the AggrDec phase, the aggregator takes the same steps as in Section 4.5. Unlike the packed matrix-vector multiplication at the agent's side, the multiplication $\mathbf{Ge}(t)$ is done as is. Although the size of the matrix can be large, it is very sparse: $\mathbf{G} = \mathbf{I}_{\lambda/l} \otimes \mathbf{g}^\intercal$, where $\mathbf{g}^\intercal = [1, 2, \dots, 2^{l-1}]$, so we only need to multiply chunks of size l. These multiplications can be efficiently obtained by bit shifting.

Correctness: The correctness of this modified scheme is immediate, given correctly selected parameters such that the noise does not overflow.

Theorem 4.6.2. *The pWSA scheme for multi-dimensional problems achieves weighted sum aggregator obliviousness w.r.t. Definition 4.2.1.*

Proof. The proof follows the same steps as the proof of Theorem 4.5.3 and also incorporates the noise added at the agents' side such that there is no information leakage from the partial sums obtained by the aggregator. □

Chapter 5

Linear Control

In this chapter, we explore the secure evaluation of linear controllers, one of the most widely used type of controllers. Nevertheless, even securely evaluating the most elementary control policies brings three challenges. First, we want to preserve the privacy of *both* signals, such as states, measurements, control inputs, references, estimates, and system parameters, such as dynamical model parameters, control gains, associated costs. Second, we are looking at dynamical processes, that operate over multiple time steps; this requires the secure evaluation protocol to be designed such that privacy and correctness are preserved at every time step (there are no overflows, decryptions are correct). Third, linear controllers are often preferred due to their simplicity for fast systems with small sampling times and low-power platforms. Hence, the secure evaluation protocol has to operate in real-time and return the desired computation before the sampling period ends, and its computational overhead has to be compatible with the available system hardware.

In this chapter, we showcase two different linear controllers in two different settings: a controller composed by a Kalman filter for estimation and a quadratic optimal control for tracking performance, computed between a cloud controller and an actuator, based on multiple sensors' data; and a decentralized generic linear controller, computed between a set of agents that are interconnected through communication links.

Specifically, in the first part of the chapter, we consider the problem of implementing a

Linear Quadratic Gaussian (LQG) controller on a system, while maintaining the privacy of the measurements, state estimates, control inputs and system model. The component subsystems and actuator outsource the LQG computation to a cloud controller and encrypt their signals and parameters. The encryption scheme used is Labeled Homomorphic Encryption, which supports the evaluation of degree-2 polynomials on encrypted data, by attaching a unique label to each piece of data and using the fact that the outsourced computation is known by the actuator. We write the state estimate update and control computation as multivariate polynomials in the encrypted data and propose an extension to the Labeled Homomorphic Encryption scheme that achieves the evaluation of low-degree (larger than two) polynomials on encrypted data. We showcase the numerical results of the proposed protocol for a temperature control application that indicates competitive online times.

In the second part of the chapter, we propose a secure multi-party computation scheme that ensures the private computation of the linear control updates of each agent in a distributed system, without leaking any other information about the states and controls of their neighbors or themselves. To this end, we make use of additively homomorphic encryption and private sum aggregation schemes. We analyze the conditions such that a dishonest agent cannot observe the rest of the network. Finally, we present implementations of the proposed schemes and showcase their efficiency.

This chapter covers the work presented in [6, 7, 12].

5.1 Secure evaluation of LQG control

5.1.1 Introduction

In the setting of distributed systems with large number of sensors that collect data over large periods of times, e.g., FitBit data, medical monitoring data or parameters in a plant, the data from the sensors is aggregated, stored and processed at a powerful server, generically called cloud. Requesting parties submit the processing algorithms they want to perform on the data stored at the cloud. Although cloud computing solves the storage and computation problems, it also raises issues about trust and privacy of the data and results [46]. Users agree to participate in the computation if their data is guaranteed to remain concealed from both the cloud and requester. Similarly, a requesting party desires to keep private the parameters of its processing algorithms, as well as the result of the computation.

Usually, in cloud-outsourced control and estimation applications, the controller or planner knows the algorithms that are outsourced to the cloud, for instance, Kalman Filter, Linear Quadratic (Gaussian) Regulator or Model Predictive Controller. An encrypted Linear Quadratic Gaussian (LQG) controller can be useful for any application that requires distributed noisy private data from sensors to be aggregated and steered by a potentially untrustworthy cloud. Examples of such applications include: power generation regulation, robots tracking targets in a dangerous environment, temperature regulation in smart buildings, packet routing in a private computer network. Since these applications involve multiple entities, the matrices in the model will depend on local private data. This requires both sensor data and system parameters to be private. In the rest of the section, we will focus on the problem of secure implementation of an LQG controller on private data.

General frameworks under the umbrella of Secure Multiparty Computation (SMPC) [74] have been proposed to solve the problem of private computations with data collected from multiple parties. While their generality is desirable in some cases, it is also advantageous to exploit the particularities of the specific architecture and the computation that is performed. Specifically, one of the solutions in the literature called Labeled Homomorphic Encryption [27], which we will explore in this section, achieves accelerated complex operations on encrypted data by making use of the knowledge of the algorithm the cloud computes by the requester party.

5.1.1.1 Related work

There are several recent approaches that explore privacy-preserving filters and controllers, with the goal of concealing the private information from untrusted parties. Linear encrypted controllers and filters are presented in [92, 199] using additively homomorphic encryption– the signals are encrypted but the gains are public; using multiplicatively homomorphic encryption in [139]–both the gains and signals are encrypted, however the decryptor is able to find out not just the final result, but also products of scalars before summation, which can leak at least which signals or gain entries are zero; and using fully homomorphic encryption in [134]–both the gains and signals are encrypted and only the final result is revealed, but the scheme is computationally prohibitive, requiring multiple controllers.

SMPC approaches, based on secret sharing schemes that can be combined with homomorphic encryption and/or garbled circuits, are considered in the following works: an encrypted Extended Kalman Filter was explored in [108], where the encrypted gains are computed by repeated exchanges between a client and a server that achieve encrypted complex operations; an encrypted linear Finite Impulse Response filter with plaintext coefficients is computed in [209]; and in [20], an encrypted multi-sensor information filtering is proposed, where a grid operator aggregates the encrypted estimates from sensors and sends it to the mobile agent requesting its location.

A different privacy goal is Differential Privacy (DP), which adds randomness to the inputs and/or computation, so that the inputs cannot be reconstructed from the resulting output. Techniques from works such as Kalman Filter with DP [144] and LQG with DP [114] can be used to augment the output privacy of a secure filter and controller. However, such techniques reduce the accuracy of the result and stability is difficult to guarantee.

5.1.1.2 Contributions

We develop a protocol that privately performs the estimation and control as described by an LQG controller, *without revealing anything about the private states, gain matrices, control inputs and intermediary steps, while achieving fast running times.* Our contributions are the following. First, we describe the cryptographic tool Labeled Homomorphic Encryption and show how the labels can be naturally exploited in estimation and control applications. Since LQG requires evaluating a polynomial on encrypted data, we propose an extension to the Labeled Homomorphic Encryption scheme that can evaluate an encrypted polynomial of degree $d \geq 2$ by using offline communication and computation. Second, we propose a protocol that achieves the *fully* encrypted execution of an LQG controller on encrypted model and encrypted data and allows different parties to have different keys. We provide two solutions, depending on how much precomputed information is available and on the architecture of the problem. Finally, we illustrate the performance of the encrypted LQG on data from a temperature control application.

5.1.2 Problem setup

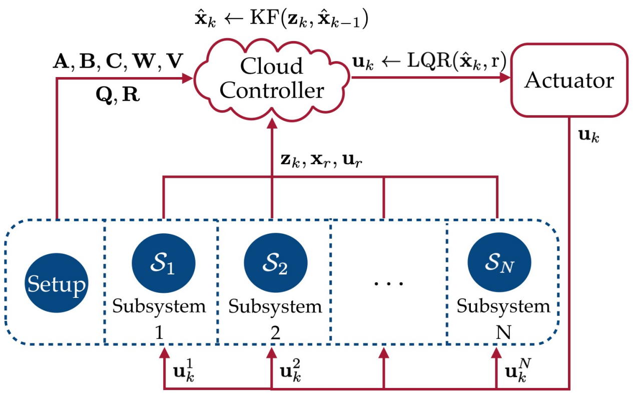
We consider agents or subsystems in the architecture in Figure 5.1, with local sensors, and an actuator that needs to apply the control inputs based on the measurements and references from the agents. The agents and the actuator employ a cloud server to privately compute an LQG controller, that does not have access to the model of the system, control gains, the measurements or the desired references. The system has the following model:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \\ \mathbf{z}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{v}_k, \quad k = 0, \dots, N-1, \end{aligned} \tag{5.1.1}$$

where $\mathbf{x}_k, \mathbf{w}_k \in \mathbb{R}^n, \mathbf{u}_k \in \mathbb{R}^m, \mathbf{z}_k, \mathbf{v}_k \in \mathbb{R}^p$. Each subsystem $i \in [N]$ has a partition of the states $\mathbf{x}_k^i \in \mathbb{R}^{n_i}$, a partition of the control inputs $\mathbf{u}_k^i \in \mathbb{R}^{m_i}$ and a partition of the measurements $\mathbf{z}_k^i \in \mathbb{R}^{p_i}$, such that their union forms system (5.1.1). The distributed system has one proxy entity that facilitates the cloud-computation: a setup entity, which holds the model of the system, that is not fully known by each individual subsystem. The subsystems want to conceal their data from the other participants in the computation, hence having partitions of the data is justified in this context.

We assume that the process and measurement noise vectors are uncorrelated i.i.d. random variables with zero mean and known positive semi-definite and respectively, positive definite covariance matrices: $E[\mathbf{w}_k\mathbf{w}_k^\intercal] = \mathbf{W}$ and $E[\mathbf{v}_k\mathbf{v}_k^\intercal] = \mathbf{V}$. The initial state is a random Gaussian variable with a finite mean and covariance matrix. Furthermore, we assume that $\{\mathbf{x}_0, \mathbf{w}_1, \dots, \mathbf{w}_k, \mathbf{v}_1, \dots, \mathbf{v}_k\}$ are mutually independent.

Our first control objective is to achieve stability for the system (5.1.1). The separation principle [38] allows the optimal control problem to be divided into two successive steps: the design of an optimal estimator for the system's state, which is the Kalman Filter under the



$\hat{\mathbf{x}}_k \leftarrow \mathrm{KF}(\mathbf{z}_k, \hat{\mathbf{x}}_{k-1})$
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{W}, \mathbf{V}$
$\mathbf{Q}, \mathbf{R}$
Cloud Controller
$\mathbf{u}_k \leftarrow \mathrm{LQR}(\hat{\mathbf{x}}_k, \mathrm{r})$
Actuator
$\mathbf{u}_k$
$\mathbf{z}_k, \mathbf{x}_r, \mathbf{u}_r$
Setup
$\mathcal{S}_1$
Subsystem 1
$\mathcal{S}_2$
Subsystem 2
...
$\mathcal{S}_N$
Subsystem N
$\mathbf{u}_k^1$
$\mathbf{u}_k^2$
$\mathbf{u}_k^N$

Figure 5.1: Architecture of the cloud-outsourced LQG problem: the subsystems send their measurements and desired references to the cloud. The cloud has to run the LQG algorithm on the measurements and the system's matrices and send the result to the actuator. The variables in the figure are described in equations (5.1.1)–(5.1.4).

assumption that the process and measurement noise vectors have a Gaussian distribution; and the design of an optimal controller for the system with the perfect information given by the estimator, which we achieve by minimizing a quadratic cost.

Our second control objective is to steer the system to a reference $\mathbf{r}$ for the measurements, $\mathbf{x}_r$ for the states and $\mathbf{u}_r$ for the control inputs, composed by the desired references of each subsystem. We want to determine the control input $\mathbf{u}_k$ such that the output deviation $\Delta\mathbf{z}_k := \mathbf{z}_k - \mathbf{r}$, the state deviation $\Delta\mathbf{x}_k := \mathbf{x}_k - \mathbf{x}_r$ and the control deviation $\Delta\mathbf{u}_k := \mathbf{u}_k - \mathbf{u}_r$ are small for all values of k. We can write the system:

$$\begin{aligned} \Delta\mathbf{x}_{k+1} &= \mathbf{A}\Delta\mathbf{x}_k + \mathbf{B}\Delta\mathbf{u}_k + \mathbf{w}_k \\ \Delta\mathbf{z}_k &= \mathbf{C}\Delta\mathbf{x}_k + \mathbf{v}_k, \quad k = 0, \dots, N-1, \end{aligned} \tag{5.1.2}$$

For simplicity, we consider the stationary LQG problem in this section, which is often used in real-time implementations due to the low memory requirements [38]. Assuming that the process and measurement noise processes are white, Gaussian and stationary, the controllability (stabilizability) of the pairs $(\mathbf{A}, \mathbf{B})$ and $(\mathbf{A}, \mathbf{W}^{\frac{1}{2}})$ and the observability (detectability) of the pairs $(\mathbf{A},\mathbf{C})$ and $(\mathbf{A},\mathbf{Q}^{\frac{1}{2}})$, the LQG problem with the cost over a horizon T:

$$J = \mathbb{E}\Big[\Delta\mathbf{z}_T^\intercal\mathbf{Q}\Delta\mathbf{z}_T + \sum_{k=0}^{T-1}\left(\Delta\mathbf{z}_k^\intercal\mathbf{Q}\Delta\mathbf{z}_k + \Delta\mathbf{u}_k^\intercal\mathbf{R}\Delta\mathbf{u}_k\right)\Big], \tag{5.1.3}$$

allows an infinite horizon solution [24, 38]:

$$\begin{aligned}\mathbf{u}_k &= -\mathbf{K}(\hat{\mathbf{x}}_k - \mathbf{x}_r) + \mathbf{u}_r,\\ \hat{\mathbf{x}}_{k+1} &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{L}\left(\mathbf{z}_{k+1} - \mathbf{C}(\mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k)\right),\end{aligned} \tag{5.1.4}$$

where $\hat{\mathbf{x}}$ is the estimated state. The steady-state Kalman and control gains are obtained by solving the following discrete algebraic Riccati equations, which converge under the assumptions described above:

$$\begin{aligned}&\mathbf{L} = \mathbf{P}\mathbf{C}^\intercal\left(\mathbf{C}\mathbf{P}\mathbf{C}^\intercal + \mathbf{V}\right)^{-1}, \quad \mathbf{K} = \left(\mathbf{B}^\intercal\mathbf{S}\mathbf{B} + \mathbf{R}\right)^{-1}\mathbf{B}^\intercal\mathbf{S}\mathbf{A},\\ &\mathbf{P} = \mathbf{A}^\intercal\mathbf{P}\mathbf{A} - \mathbf{A}^\intercal\mathbf{P}\mathbf{C}^\intercal\left(\mathbf{C}\mathbf{P}\mathbf{C}^\intercal + \mathbf{V}\right)^{-1}\mathbf{C}\mathbf{P}\mathbf{A} + \mathbf{W},\\ &\mathbf{S} = \mathbf{A}^\intercal\mathbf{S}\mathbf{A} - \mathbf{A}^\intercal\mathbf{S}\mathbf{B}\left(\mathbf{B}^\intercal\mathbf{S}\mathbf{B} + \mathbf{R}\right)^{-1}\mathbf{B}^\intercal\mathbf{S}\mathbf{A} + \mathbf{Q}.\end{aligned} \tag{5.1.5}$$

The matrices $\mathbf{A} - \mathbf{B}\mathbf{K}$ and $\mathbf{A} - \mathbf{L}\mathbf{C}\mathbf{A}$ determine the stability of the closed-loop system and the quality of the estimation. Notice that an iteration of the LQG (5.1.4) can be written as a multivariate polynomial in $\hat{\mathbf{x}}_k$, $\mathbf{u}_k$, $\mathbf{x}_r$, $\mathbf{u}_r$ and $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{K},\mathbf{L}$.

The LQG control presented can be abstracted in a more general framework. Consider a cloud server that collects encrypted data from several clients, which in Figure 5.1 were the subsystems. The data represents time series and is labeled with the corresponding time. A requester, which in Figure 5.1 was the actuator, makes queries, e.g., an LQG iteration, that can be written as multivariate polynomials over the data stored at the cloud server and solicits the result. We allow *semi-honest* parties, which are parties that follow the preset protocols, but can locally process the data in order to try to infer private information. This is a reasonable assumption, as cloud services are reputation based and cannot afford to tamper with the clients' data. Furthermore, we consider the classical computational privacy definition of an interactive protocol, given in Definition 2.2.6. We assume that only computations that have been previously agreed upon can be requested, such that the privacy of the data stored at the cloud server is not broken (e.g. the requester cannot simply ask for the data).

Our privacy goal is to develop a solution that allows the cloud to efficiently perform the LQG computation and send the results to the requester, without finding out anything about the private data and results, as well as preserving the privacy of the input data with respect to the requester.

5.1.3 Labeled homomorphic encryption and extension

Labeled Homomorphic Encryption, abbreviated as LabHE, was recently introduced in [27] and is a scheme that allows the computation of multivariate degree-two polynomials on encrypted data. The appeal of LabHE is that it uses a simple addendum to an additively homomorphic scheme in order to obtain, apart from unlimited additions between encrypted values, also a multiplication between two encrypted values. The underlying homomorphic encryption scheme can be instantiated with most of the existing schemes and inherits their properties. LabHE exploits the common trait that the party that requests the result of the encrypted computation knows what the computation is.

LabHE can process data from multiple users with different private keys, as long as the requesting party has a master key. This scheme makes use of the fact that the decryptor knows the query to be executed on the encrypted data, which we will refer to as a program. Furthermore, we want a cloud server that only has access to the encrypted data to be able to perform the program on the encrypted data and the decryptor to be able to decrypt the result. To this end, the inputs to the program need to be uniquely identified. Therefore, an encryptor assigns a unique label to each message and sends the encrypted data along with the corresponding encrypted labels to the server. Labels can be time instances, locations, id numbers etc. Denote by $\mathcal{M}$ the message space. A program that has labeled inputs is called a labeled program [27]. The details of this scheme are described in Preamble 2.5.2.

We propose a two-party extension of LabHE that achieves more encrypted multiplications at the expense of offline computation and communication. This extension is critical for the execution of LQG described in Section 5.1.2 in an encrypted manner.

Definition 5.1.1. An admissible function $f : \mathcal{M}^n \to \mathcal{M}$ is a multivariate polynomial of degree d on n variables, where $d = 2$ for the original version of LabHE and $d \geq 2$ for the extended version described below.

5.1.3.1 Extension of LabHE to degree d-polynomials

In [55], the authors show how to obtain the evaluation of degree-3 and 4 polynomials over encrypted data by using, instead of AHE, schemes that are level-2 homomorphic, i.e., already support encrypted additions and one encrypted multiplication, such as BGN [42]. In general, higher level homomorphic schemes involve more complex computations than AHE schemes. The solution proposed in [55] modifies the BGN scheme, and the resulting scheme allows only a small message space, which is a problem since the secrets generated by the pseudorandom generator are large and require expensive decryption (solving a discrete logarithm problem).

We propose an alternative extension for LabHE that achieves the evaluation of degree-d polynomials over encrypted data. The advantage of our method is that the online computations and communication are replaced by offline computations and communication. However, for general multivariate polynomials, the offline communication will be exponential in d.

The multiplication of two encrypted values in LabHE is possible because the party that performs the multiplication has access to $[[b_1]]$ and $[[b_2]]$ for two ciphertexts C_1, C_2. We notice that if a party that wants to perform a multiplication between three ciphertexts $C_1 = (m_1 - b_1, [[b_1]]), C_2 = (m_2 - b_2, [[b_2]]), C_3 = (m_3 - b_3, [[b_3]])$ has access to $[[b_1 \cdot b_2]], [[b_1 \cdot b_3]], [[b_2 \cdot b_3]]$, then it can compute $[[m_1 \cdot m_2 \cdot m_3 - b_1 \cdot b_2 \cdot b_3]]$:

$$m_1 m_2 m_3 - b_1 b_2 b_3 = (m_1 - b_1)(m_2 - b_2)(m_3 - b_3) + (m_1 - b_1) b_2 b_3 + (m_2 - b_2) b_1 b_3 +$$
$$+ (m_3 - b_3) b_1 b_2 + (m_1 - b_1)(m_2 - b_2) b_3 + (m_1 - b_1)(m_3 - b_3) b_2 + (m_2 - b_2)(m_3 - b_3) b_1.$$

We first extend the label program definition from Definition 2.5.5. Given t labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and an admissible function $g : \mathcal{M}^t \to \mathcal{M}$, the composed program $\mathcal{P}^g$

is obtained by evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, and can be denoted compactly as $\mathcal{P}^g = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The labeled inputs of $\mathcal{P}^g$ are all the distinct labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$.

We define the extension of LabHE as the tuple eLabHE = $(\widehat{\mathrm{Init}}, \widehat{\mathrm{KeyGen}}, \hat{\mathrm{E}}, \widehat{\mathrm{Eval}}_1, \widehat{\mathrm{Eval}}_2, \hat{\mathrm{D}})$, where the primitives $\widehat{\mathrm{Init}}$, $\widehat{\mathrm{KeyGen}}$, $\hat{\mathrm{E}}$ and $\hat{\mathrm{D}}$ are inherited from the LabHE scheme. We define $\widehat{\mathrm{Eval}}_1$, that is like an offline part of the $\widehat{\mathrm{Eval}}$ primitive, and has to be performed by the decryptor:

4.1) $\widehat{\mathrm{Eval}}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \mathcal{P})$: Takes the master public key, the master secret key, the users' public keys and the program $\mathcal{P} = (f, \tau_1, \ldots, \tau_t)$. Let $\mathrm{upk} = (\mathrm{upk}_1, \ldots, \mathrm{upk}_l)$. For all $j \in [l]$, it uses the master secret key to get the users' secret keys $\mathrm{usk}_j \leftarrow \mathrm{D}(\mathrm{msk}, \mathrm{upk}_j)$. Then, it computes $b_i \leftarrow F(\mathrm{usk}_{j_i}, \tau_i)$, for $i \in [t], j \in [l]$. For each monomial of order k in f, denoted as $g_k(\tau_T)$, for $2 < k < d$ and $T \subseteq [t], |T| = k$, it computes $b_i \leftarrow F(K, \tau_i)$, $i \in T$. Then, it outputs $\tilde{g}_k(b) = \big\{[[\prod_{i \in S} b_i]] \big| S \subseteq T, |S| > 2\big\}$.

We denote by $\tilde{g}(b)$ the vector of all $\tilde{g}_k(b)$ corresponding to all monomials of order k in f for $2 < k < d$. Then, we can overload the primitive $\widehat{\mathrm{Eval}}$ to compute admissible functions f that consist of multivariate polynomials of degree d on encrypted data and denote it $\widehat{\mathrm{Eval}}_2$:

4.2) $\widehat{\mathrm{Eval}}_2(\mathrm{mpk}, \tilde{f}, C_1, \ldots, C_t)$: Takes the master public key, a specification $\tilde{f} = (f, \tilde{g}(b))$, composed of an admissible function $f : \mathcal{M}^t \to \mathcal{M}$ and the tuple of monomials $\tilde{g}(b)$. It also takes t ciphertexts $C_1, \ldots, C_t$ and returns a ciphertext C. $\widehat{\mathrm{Eval}}_2$ is composed of the following computation blocks:

- $\widehat{\mathrm{Mlt}}(C_1, \ldots, C_d, \tilde{g}_d(b))$: Takes $C_i = (a_i, \beta_i) \in \mathcal{M} \times \mathcal{C}$ for $i \in [d]$ and the corresponding tuple of monomials $\tilde{g}_d(b)$ and outputs:

$$C = \sum_{\emptyset \neq S \subseteq [d]} \Big((\prod_{j \in S} a_j) \otimes [[\prod_{l \in [d] \setminus S} b_l]] \Big) = [[\prod_{i=1}^{d} m_i - \prod_{i=1}^{d} b_i]] =: \alpha \in \mathcal{C}.$$

- $\widehat{\mathrm{Add}}(C_1, C_2)$: If $C_i = (a_i, \beta_i) \in \mathcal{M} \times \mathcal{C}$ for $i = 1, 2$, then outputs $C = (a_1 + a_2, \beta_1 \oplus \beta_2) =: (a, \beta) \in \mathcal{M} \times \mathcal{C}$. If both $C_i = \alpha_i \in \mathcal{C}$, for $i = 1, 2$, then outputs

$C = \alpha_1 \oplus \alpha_2 =: \alpha \in \mathcal{C}$. If $C_1 = (a_1, \beta_1) \in \mathcal{M} \times \mathcal{C}$ and $C_2 = \alpha_2 \in \mathcal{C}$, then outputs $C = (a_1, \beta_1 \oplus \alpha_2) =: (a, \beta) \in \mathcal{M} \times \mathcal{C}$.

- $\mathrm{c\hat{M}lt}(c, C')$: Takes a plaintext $c \in \mathcal{M}$ and a ciphertext C'. If $C' = (a', \beta') \in \mathcal{M} \times \mathcal{C}$, outputs $C = (c \cdot a', c \otimes \beta') =: (a, \beta) \in \mathcal{M} \times \mathcal{C}$. If $C' = \alpha' \in \mathcal{C}$, outputs $C = c \otimes \alpha' =: \alpha \in \mathcal{C}$.

Like LabHE, the extension eLabHE also satisfies correctness, semantic security and context-hiding. Their definitions and proofs are provided in Appendix D.1.

Theorem 5.1.2. *The* eLabHE *scheme is correct.*

Theorem 5.1.3. *The* eLabHE *scheme is semantically secure.*

Theorem 5.1.4. *The* eLabHE *scheme is context-hiding.*

The LabHE extension we proposed can be used to evaluate degree-d multivariate polynomials over encrypted data from every level-d' homomorphic scheme, with $d' < d$. The idea is that, as long as the requester knows in advance the polynomial that has to be evaluated and the labels of the inputs, it can send offline to the cloud the encryptions of the secrets that the cloud cannot compute, i.e., $\left[\left[\prod_{i \in T} b_i\right]\right]$, where $d' \leq |T| < d$.

In what follows, we will compare the proposed extension of LabHE to other secure methods of achieving the evaluation of a degree-d polynomial on encrypted data.

Consider a party A that has to compute a product of $d > 2$ encrypted messages $m_1, \ldots, m_d$ and was given the corresponding ciphertexts $C_1, \ldots, C_d$. Party B has the secret key and should only obtain the result $m_1 \cdot \ldots \cdot m_d$. Without the extension we proposed above, and using a level-1 encryption scheme, party A can only compute encrypted products of two factors and then has to require B to refresh the encryption, in the following way: A splits $[[m_i \cdot m_j - b_i \cdot b_j]] = \alpha \in \mathcal{C}$ in a secret r and $\alpha \oplus [[-r]]$ and sends the latter to B; B decrypts and obtains $m_i \cdot m_j - r$, assigns it a different label τ and computes $b \leftarrow F(K, \tau)$, encrypts it and sends back $C = (a' = m_i \cdot m_j - r - b, \beta' = [[b]]) \in \mathcal{M} \times \mathcal{C}$; A reconstructs $(a' + r, \beta')$ and continues the computation. For a product of d factors, B has to perform $d - 1$ times the additive sharing and merging and send to A d ciphertexts in $\mathcal{C}$ online, while B has to perform d decryptions, $d-1$ encryptions and send to A $d-1$ ciphertexts in $\mathcal{M}\times\mathcal{C}$ online, and $d-1$ plaintext multiplications offline.

If we use the extension of LabHE proposed in [55], which uses a level-$(d-1)$ homomorphic scheme, A has to perform 2^d-1 encrypted additions, 2^d-2 plaintext-ciphertext multiplications and 2^d-d-2 encrypted multiplications and send to B the final result in $\mathcal{C}$ online. However, the decryption that B is required to perform is in the level-$(d-1)$ homomorphic scheme and has substantially high complexity.

If we use the extended version of LabHE we proposed in this section, A has to perform 2^d-1 encrypted additions and 2^d-2 plaintext-ciphertext multiplications and then sends to B only the final result in $\mathcal{C}$ online, while B offline computes and sends 2^d-d-2 encryptions in $\mathcal{C}$ and performs one online decryption. In conclusion, our extended version of LabHE replaces the online computations from B and online communication by online computation at A, offline computation at B and offline communication. However, one can see that for large degrees d, this scheme loses its practicality, because the required offline communication increases exponentially with the degree d and the number of monomials.

In particular, computing degree-d univariate polynomials over encrypted variables with our proposed extension can be done more efficiently than degree-d multivariate polynomials. Evaluating such polynomials can be very useful in approximating nonpolynomial functions via Taylor series or Chebyshev series.

We use Newton's binomial to observe:

$$m^d=(m-b+b)^d=\sum_{k=0}^{d}\binom{d}{k}(m-b)^{d-k}b^k.$$

Namely, for computing the encryption $C'=[[m^d-b^d]]\in\mathcal{C}$, A has to perform $d-1$ encrypted additions and $d-2$ plaintext-ciphertext multiplications and then sends to B only the final result in $\mathcal{C}$ online, while B offline computes and sends $d-2$ encryptions in $\mathcal{C}$ $([[b^2]],\dots,[[b^{d-1}]])$ and performs one online decryption.

Comparing this extension for univariate polynomials with a d-leveled homomorphic encryption scheme, the advantages of the LabHE extension are: offline generation and transmission of randomness, and smaller online size of ciphertexts sent from the agents. However, its disadvantages are that we cannot perform tree multiplication (since once a multiplication is done, we cannot perform another one) and we still need memory linear in the degree of the polynomial.

5.1.4 Encrypted execution of LQG

5.1.4.1 Encrypted LQG with public system model

If the state matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, the noise covariances $\mathbf{W}$, $\mathbf{V}$, the costs $\mathbf{Q}$, $\mathbf{R}$ can be public, the setup sends them in plaintext to the cloud controller. Then, the cloud can compute the Kalman gain $\mathbf{L}$ and feedback gain $\mathbf{K}$ as in (5.1.5). The subsystems encrypt with the public key of the actuator their measurements and desired reference, along with the initial state, and send them to the cloud controller. This case is linear in the encrypted data and AHE is sufficient to ensure privacy of the measurements, states, reference and control inputs: the cloud can compute locally, at each time step, the encrypted control input $\mathbf{u}_k$ as in (5.1.6).

$$\begin{aligned}
&\mathrm{E}(\mathbf{u}_k) = \mathrm{Add}(\mathrm{cMlt}(-\mathbf{K}, \mathrm{Add}(\mathrm{E}(\hat{\mathbf{x}}_k), \mathrm{E}(-\mathbf{x}_r))), \mathrm{E}(\mathbf{u}_r)) \qquad (5.1.6)\\
&\mathrm{E}(\hat{\mathbf{x}}_{k+1}) = \mathrm{Add}(\mathrm{cMlt}(\mathbf{L}, \mathrm{E}(\mathbf{z}_{k+1})), \mathrm{cMlt}(\mathbf{I} - \mathbf{LC}, \mathrm{Add}(\mathrm{cMlt}(\mathbf{A}, \mathrm{E}(\hat{\mathbf{x}}_k)), \mathrm{cMlt}(\mathbf{B}, \mathrm{E}(\mathbf{u}_k))))).
\end{aligned}$$

After this computation, the cloud sends the encrypted control input to the actuator, which decrypts it. This setup where the state matrices are public is considered, for example, in [92].

5.1.4.2 Encrypted LQG with private system model

As justified in the Introduction, in many situations it is important to protect not only the signals (e.g., the states, measurements), but also the system model. To this end, we propose a solution that uses Labeled Homomorphic Encryption to achieve encrypted multiplications and the private execution of LQG on encrypted data. LabHE has a useful property called *unbalanced efficiency* that can be observed from Section 5.1.3 and was described in [56], which states that only the server is required to perform operations on ciphertexts, while the decryptor performs computations only on the plaintext messages. We will employ this property by having the cloud perform the more complex operations and the actuator the more efficient ones.

In Figure 5.2, the actuator holds the LQG query, denoted by f_{LQG}, which describes the functionality of LQG. Offline, the actuator generates a pair of master keys and distributes the master public key to the rest of the parties. The setup and subsystems generate their secret keys and send the corresponding public keys to the actuator. Still offline, these parties generate the labels corresponding to their data with respect to the time stamp and the size of the data. As explained in Section 5.1.3, the labels are crucial to achieving the encrypted multiplications. Moreover, when generating them, it is important to make sure that no two labels that will be encrypted with the same key are the same. When the private data are times series, as in our problem, the labels can be easily generated using the time steps and sizes corresponding to each signal, with no other complex synchronization process necessary between the actors. This is shown in Protocol 5.1.1.

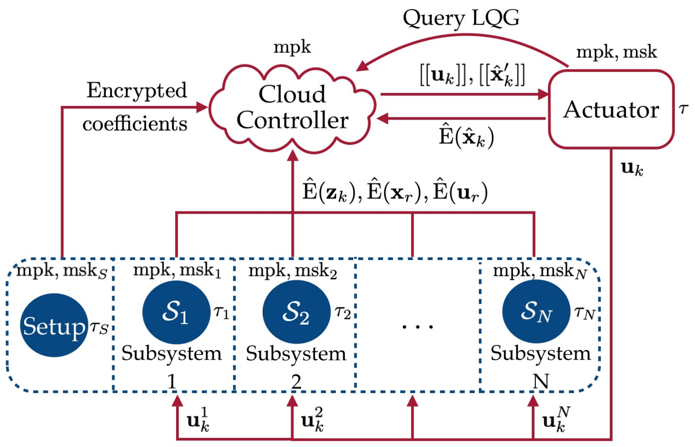


Figure 5.2: The setup and subsystems send their encrypted data to the cloud. The cloud runs the LQG algorithm on the private measurements and the private coefficients and sends the encrypted result to the actuator. The latter then actuates the system with the decrypted inputs received.

Our protocols consist of three phases: the offline phase, in which the computations that are unrelated to the specific data of the users are performed, the initialization phase, in which the computations related to the constant parameters in the problem are performed, and the online phase, in which computations on the variables of the problem are performed. The initialization phase can be offline, if the constant parameters are a priori known, and online otherwise.

The setup sends the LabHE encryptions of the state matrices and control gains to the cloud controller, once, before the execution begins. The subsystems send the encryptions of their initial states and desired reference to the cloud controller, also once, at the onset of the execution. Then, at every time step, the subsystems encrypt their measurements and send them to the cloud. After the cloud performs the encrypted LQG query for one time step, it sends the encrypted control input at the current time step to the actuator, which decrypts it and inputs it to the system. In Protocol 5.1.2, we describe how the encrypted LQG query is performed by the parties, which involves the actuator sending an encryption of the processed result back to the cloud such that the computation can continue in the future time steps.

When the state matrices, feedback gains and initial condition are private to the cloud and are stored in an encrypted form $\hat{\mathrm{E}}(\mathbf{A})$, $\hat{\mathrm{E}}(\mathbf{B}), \hat{\mathrm{E}}(\mathbf{C}), \hat{\mathrm{E}}(\mathbf{K}), \hat{\mathrm{E}}(\mathbf{L}), \hat{\mathrm{E}}(\hat{\mathbf{x}}_0)$, then the depth of the LQG program in terms of multiplications is higher than two. However, if the cloud computes the encryption of the coefficients:

$$\mathbf{\Gamma}_1 := (\mathbf{I} - \mathbf{LC})(\mathbf{A} - \mathbf{BK}), \quad \mathbf{\Gamma}_2 := (\mathbf{I} - \mathbf{LC})\mathbf{BK}, \quad \mathbf{\Gamma}_3 := (\mathbf{I} - \mathbf{LC})\mathbf{B}, \tag{5.1.7}$$

then, the multiplication depth in terms of full LabHE ciphertexts for the state estimate at iteration 1 is one and for the control input is two. We assume for the moment that the cloud has access to $\hat{\mathrm{E}}(\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\mathbf{\Gamma}_2), \hat{\mathrm{E}}(\mathbf{\Gamma}_3)$ and discuss how to achieve these products in Section 5.1.5.

For the subsequent time steps $k \geq 1$, Protocol 5.1.2 ensures that the actuator receives the control input at step k, corresponding to the program $\mathcal{P}_k$. For the computation of the state estimation at time $k+1$, the cloud needs to have the full LabHE ciphertext of $\hat{\mathrm{E}}(\hat{\mathbf{x}}_k) \in \mathcal{M} \times \mathcal{C}$, but as a result of the polynomial evaluations at time step k, it has the AHE ciphertext $[[\hat{\mathbf{x}}_k]]$. To privately refresh the encryption, which happens in lines 2–4 of Protocol 5.1.2, the cloud uses a one-time pad $\mathbf{r}_k$, as described in Preamble 2.4, and sends $[[\hat{\mathbf{x}}'_k - \mathbf{r}_k]]$ to the actuator, which is possible since the scheme is additively homomorphic. The actuator calls the decryption primitive and sends back $\hat{\mathrm{E}}(\hat{\mathbf{x}}_k - \mathbf{r}_k)$, from which the cloud retrieves $\hat{\mathrm{E}}(\hat{\mathbf{x}}_k)$.

<span style="font-variant:small-caps">Protocol</span> 5.1.1: Initialization of LQG

Input: Actuator: f_{LQG}; Subsystems: $\mathbf{x}_0, \mathbf{x}_r, \mathbf{u}_r$; Setup: $\mathbf{K}, \mathbf{L}, \mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3$.

Output: Actuator: $\mathbf{u}_0$; Cloud: $\hat{\mathrm{E}}(\mathbf{x}_0), \hat{\mathrm{E}}(\mathbf{K}), \hat{\mathrm{E}}(\mathbf{L}), \hat{\mathrm{E}}(\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\mathbf{\Gamma}_2), \hat{\mathrm{E}}(\mathbf{\Gamma}_3)$, $[[\tilde{\mathbf{u}}_r]], [[\hat{\mathbf{x}}_\Gamma]]$.

Offline:

1: Actuator: Generate $(\mathrm{mpk}, \mathrm{msk}) \leftarrow \hat{\mathrm{Init}}(1^\lambda)$ and distribute mpk to the others.

2: Subsystems, Setup: Get $(\mathrm{usk}, \mathrm{upk}) \leftarrow \hat{\mathrm{KeyGen}}(\mathrm{mpk})$ and send upk to the actuator.

3: Subsystems, Setup, Actuator: Allocate labels $\tau_1, \dots, \tau_t$ to the inputs of function f_{LQG}:
Subsystem i: for each measurement at time $k \in \{0, \dots, T-1\}$, $\mathbf{z}^i_k$ of size p^i, generate the corresponding labels $\tau_{z^i_k} = [kp^i \quad kp^i+1 \quad \dots \quad (k+1)p^i - 1]^\intercal$; then similarly for $\mathbf{x}^i_0, \mathbf{x}^i_r, \mathbf{u}^i_r$ with the labels starting from where the previous signals ended.
Setup: for matrix $\mathbf{K} \in \mathbb{R}^{m\times n}$, set $l = 0$, generate $\tau_k = \begin{bmatrix} l & l{+}1 & \dots & l{+}n{-}1 \\ \vdots & & & \vdots \\ l{+}(m{-}1)n & l{+}(m{-}1)n{+}1 & \dots & l{+}mn{-}1 \end{bmatrix}$ and update $l = mn$, then follow the same steps for the rest of the matrices, starting from l and updating it.
Actuator: follow the same steps as the subsystems and setup, and then generate similar labels for the state estimates $\hat{\mathbf{x}}_k$ starting from the last l.

4: Subsystems, Setup, Actuator: Perform the offline part of the LabHE encryption primitive and decryption for the actuator.

5: Cloud: Generate randomness for Protocol 5.1.2.

6: Actuator: Form the program $\mathcal{P} = (f_{LQG}, \tau_1, \dots, \tau_t)$.

Initialization:

7: Setup: Perform the online part of LabHE encryption and send to the cloud: $\hat{\mathrm{E}}(\mathbf{\Gamma}_1)$, $\hat{\mathrm{E}}(\mathbf{\Gamma}_2)$, $\hat{\mathrm{E}}(\mathbf{\Gamma}_3)$, $\hat{\mathrm{E}}(\mathbf{K})$, $\hat{\mathrm{E}}(\mathbf{L})$.

8: Subsystems: Perform the online part of LabHE encryption and send to the cloud: $\hat{\mathrm{E}}(\mathbf{x}_0), \hat{\mathrm{E}}(\mathbf{x}_r), \hat{\mathrm{E}}(\mathbf{u}_r)$.

9: Cloud: Compute $[[\tilde{\mathbf{u}}_r]] \leftarrow \mathrm{Add}(\hat{\mathrm{Mlt}}(\hat{\mathrm{E}}(\mathbf{K}), \hat{\mathrm{E}}(\mathbf{x}_r)), [[\mathbf{u}_r]])$, $[[\hat{\mathbf{x}}_\Gamma]] \leftarrow \mathrm{Add}(\hat{\mathrm{Mlt}}(\hat{\mathrm{E}}(\mathbf{\Gamma}_2), \hat{\mathrm{E}}(\mathbf{x}_r)), \hat{\mathrm{Mlt}}(\hat{\mathrm{E}}(\mathbf{\Gamma}_3), \hat{\mathrm{E}}(\mathbf{u}_r)))$.

```
Online:
10: Cloud: $[[\mathbf{u}_0]] \leftarrow \mathrm{Add}(\hat{\mathrm{Mlt}}(\hat{\mathrm{E}}(-\mathbf{K}), \hat{\mathrm{E}}(\hat{\mathbf{x}}_0)), [[\tilde{\mathbf{u}}_r]])$.
11: Cloud: Send to the actuator $[[\mathbf{u}_0]]$.
12: Actuator: Decrypt $\mathbf{u}_0$.
```

Notice that, technically, the encryptions $[[\hat{\mathbf{x}}_k]]$ and $[[\mathbf{u}_k]]$ obtained by the cloud are not just AHE encryptions of $\hat{\mathbf{x}}_k$ and $\mathbf{u}_k$, but they also contain some products of secrets that will disappear in the decryption process. In order to not burden the notation, we omit this distinction in the protocols.

PROTOCOL 5.1.2: Encrypted LQG at time step k

Input: Actuator: $\mathrm{msk}, \mathrm{mpk}, \mathcal{P}_k$; Cloud: $\hat{\mathrm{E}}(\mathbf{z}_k), \hat{\mathrm{E}}(\hat{\mathbf{x}}_{k-1}), \hat{\mathrm{E}}(\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\mathbf{L}), \hat{\mathrm{E}}(\mathbf{K}), [[\tilde{\mathbf{u}}_r]], [[\hat{\mathbf{x}}_\Gamma]]$.

Output: Actuator: $\mathbf{u}_k$; Cloud: $\hat{\mathrm{E}}(\hat{\mathbf{x}}_k)$.

```
Online:
1: Cloud: Compute $[[\hat{\mathbf{x}}_k]] \leftarrow \mathrm{Add}(\hat{\mathrm{Mlt}}(\hat{\mathrm{E}}(\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\hat{\mathbf{x}}_{k-1})), \hat{\mathrm{Mlt}}(\hat{\mathrm{E}}(\mathbf{L}), \hat{\mathrm{E}}(\mathbf{z}_k)), [[\hat{\mathbf{x}}_\Gamma]])$.
2: Cloud: Send to the actuator $[[\hat{\mathbf{x}}'_k]] \leftarrow \mathrm{Add}([[\hat{\mathbf{x}}_k]], [[-\mathbf{r}_k]])$, where $\mathbf{r}_k \overset{\$}{\leftarrow} \mathcal{M}^n$.
3: Actuator: Decrypt $[[\hat{\mathbf{x}}_k - \mathbf{r}_k]]$ and send back a LabHE encryption of $\hat{\mathrm{E}}(\hat{\mathbf{x}}_k - \mathbf{r}_k)$.
4: Cloud: $\hat{\mathrm{E}}(\hat{\mathbf{x}}_k) \leftarrow \hat{\mathrm{Add}}(\hat{\mathrm{E}}(\hat{\mathbf{x}}_k - \mathbf{r}_k), \mathbf{r}_k)$.
5: Cloud: $[[\mathbf{u}_k]] \leftarrow \mathrm{Add}(\hat{\mathrm{Mlt}}(\hat{\mathrm{E}}(-\mathbf{K}), \hat{\mathrm{E}}(\hat{\mathbf{x}}_k)), [[\tilde{\mathbf{u}}_r]])$.
6: Cloud: Send to the actuator $[[\mathbf{u}_k]]$ and output $\hat{\mathrm{E}}(\hat{\mathbf{x}}_k)$.
7: Actuator: Decrypt $\mathbf{u}_k$.
```

PROTOCOL 5.1.3: Encrypted LQG

Input: Actuator: f_{LQG}; Subsystems: $\mathbf{x}_0$; Setup: $\mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3, \mathbf{K}, \mathbf{L}$.

Output: Actuator: $\{\mathbf{u}_k\}_{k=0,\dots,T-1}$.

```
Offline + Initialization:
1: Subsystems, Setup, Cloud and Actuator: Run Protocol 5.1.1.
```

```
    Online:
2:  for k=1,...,T-1 do
3:      Cloud and Actuator: Run Protocol 5.1.2.
4:      Actuator: Input $\mathbf{u}_k$ to the system.
5:      Subsystems: Measure $\mathbf{z}_{k+1}$, encrypt and send it to the cloud.
6:  end for
```

In Protocols 5.1.1 and 5.1.2, the actuator does not need to know the system matrices in order to compute the program and decrypt the results, but only their labels.

Theorem 5.1.5. *Protocol 5.1.3 achieves privacy of the encrypted LQG with respect to semi-honest parties, cf. Definition 2.2.5.*

Let us now consider possible coalitions between the parties shown in Figure 5.2. The cloud and the actuator are not allowed to collude, since the cloud has all the data in the system encrypted with the actuator's master key. Furthermore, all the subsystems and setup cannot be corrupted by an adversary at the same time. Under these assumptions, the following theorem holds:

Theorem 5.1.6. *Protocol 5.1.3 achieves privacy of the encrypted LQG with respect to collusions between semi-honest parties, cf. Definition 2.2.6.*

The privacy of Protocol 5.1.3, composed of Protocol 5.1.1 and $T-1$ runs of Protocol 5.1.2 follows from the semantic security and context-hiding property of LabHE and the perfect secrecy of the one-time pad. The proofs are given in Section D.2.

5.1.5 Encrypted computation of LQG coefficients

The setup party does not need to be online for the computation of LQG. For a given system, the encryptions of the system model and gains $\hat{\mathrm{E}}(\mathbf{A}), \hat{\mathrm{E}}(\mathbf{B}), \hat{\mathrm{E}}(\mathbf{C}), \hat{\mathrm{E}}(\mathbf{K}), \hat{\mathrm{E}}(\mathbf{L})$ might have been given offline to the cloud or distributed among the subsystems. In such cases, the encryptions of $\mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3$ or other polynomial functions of these matrices are not available, but they can be computed at the beginning of the protocol by the cloud and actuator as the output of a program evaluating a degree-3 polynomial, using eLabHE, and stored as $\hat{\mathrm{E}}(\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\mathbf{\Gamma}_2), \hat{\mathrm{E}}(\mathbf{\Gamma}_3)$. Notice from (5.1.7) that:

$$\mathbf{\Gamma}_3 = \mathbf{B} - \mathbf{LCB}, \quad \mathbf{\Gamma}_2 = \mathbf{\Gamma}_3\mathbf{K}, \quad \mathbf{\Gamma}_1 = \mathbf{A} - \mathbf{LCA} + \mathbf{\Gamma}_2. \tag{5.1.8}$$

Protocol 5.1.4: Initialization of LQG, extended version

```
Input: Actuator: $f_{LQG}, f_{\Gamma_1}, f_{\Gamma_2}, f_{\Gamma_3}$; Subsystems: $\mathbf{x}_0, \mathbf{x}_r, \mathbf{u}_r$; Setup: $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{K}, \mathbf{L}$.
Output: Actuator: $\mathbf{u}_0$; Cloud: $\hat{\mathrm{E}}(\mathbf{x}_0), \hat{\mathrm{E}}(\mathbf{K}), \hat{\mathrm{E}}(\mathbf{L}), \hat{\mathrm{E}}(\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\mathbf{\Gamma}_2)$, $\hat{\mathrm{E}}(\mathbf{\Gamma}_3)$, $[[\tilde{\mathbf{u}}_r]]$, $[[\hat{\mathbf{x}}_\Gamma]]$.
    Offline:
1: Perform lines 1–4 from Protocol 5.1.1.
2: Actuator: Form the programs $\mathcal{P} = (f_{LQG}, \tau_1, \dots, \tau_t)$, $\mathcal{P}_{\Gamma_3} = (f_{\Gamma_3}, \tau_B, \tau_C, \tau_L, \tau_k)$, $\mathcal{P}_{\Gamma_2} = (f_{\Gamma_2}, \tau_{\Gamma_3}, \tau_k)$, $\mathcal{P}_{\Gamma_1} = (f_{\Gamma_1}, \tau_A, \tau_C, \tau_L, \tau_k, \tau_{\Gamma_3})$. Compute $\tilde{f}_i \leftarrow \hat{\mathrm{Eval}}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \mathcal{P}_{\Gamma_i})$, for $i = 1, 2, 3$, corresponding to (5.1.8) and send it to the cloud. Create labels $\tau_{\Gamma_i}$ for refreshing $\mathbf{\Gamma}_i$.
3: Cloud: Generate randomness for Protocol 5.1.2 and Initialization.
    Initialization:
4: Setup: Encrypt matrices and send them to the cloud: $\hat{\mathrm{E}}(\mathbf{A})$, $\hat{\mathrm{E}}(\mathbf{B}), \hat{\mathrm{E}}(\mathbf{C}), \hat{\mathrm{E}}(\mathbf{K}), \hat{\mathrm{E}}(\mathbf{L})$.
5: Cloud: Compute $[[\mathbf{\Gamma}_3]] \leftarrow \hat{\mathrm{Eval}}_2(\mathrm{mpk}, \tilde{f}_3, \hat{\mathrm{E}}(\mathbf{B}), \hat{\mathrm{E}}(\mathbf{C}), \hat{\mathrm{E}}(\mathbf{L}))$, share it and send it to the actuator a share $[[\mathbf{\Gamma}'_3]]$.
6: Actuator: Decrypt $\mathbf{\Gamma}'_3$, allocate it the label $\tau_{\Gamma_3}$ and send the LabHE encryption to the cloud: $\hat{\mathrm{E}}(\mathbf{\Gamma}'_3)$.
7: Cloud: Reconstruct and obtain $\hat{\mathrm{E}}(\mathbf{\Gamma}_3)$.
8: Cloud: Compute $[[\mathbf{\Gamma}_2]] \leftarrow \hat{\mathrm{Eval}}_2(\mathrm{mpk}, \tilde{f}_2, \hat{\mathrm{E}}(\mathbf{\Gamma}_3), \hat{\mathrm{E}}(\mathbf{K}))$, $[[\mathbf{\Gamma}_1]] \leftarrow \hat{\mathrm{Eval}}_2(\mathrm{mpk}, \tilde{f}_1, \hat{\mathrm{E}}(\mathbf{A}), \hat{\mathrm{E}}(\mathbf{C}), \hat{\mathrm{E}}(\mathbf{L}), \hat{\mathrm{E}}(\mathbf{K}), \hat{\mathrm{E}}(\mathbf{\Gamma}_3))$, share them and send to the actuator the shares $[[\mathbf{\Gamma}'_2]], [[\mathbf{\Gamma}'_1]]$.
9: Actuator: Decrypt $\mathbf{\Gamma}_i$, allocate it the label $\tau_{\Gamma_i}$ and send the LabHE encryptions to the cloud: $\hat{\mathrm{E}}(\mathbf{\Gamma}'_i)$, for i = 1,2.
```

```
10: Cloud: Reconstruct and obtain $\hat{\mathrm{E}}(\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\mathbf{\Gamma}_2)$.
11: The rest follows as in lines 7–12 in Protocol 5.1.1.
```

The privacy of Protocol 5.1.4 follows from the context-hiding and semantic security of the eLabHE scheme described in Section 5.1.3.1 and the perfect privacy of one-time pads.

Remark 5.1.7. We can modify Protocol 5.1.2 such that the cloud sends the control input to the actuator before requesting a refreshed encryption of the state estimate. Specifically, the cloud can compute first, in an encrypted fashion:

$$\mathbf{u}_k = -\mathbf{K}\mathbf{\Gamma}_1\hat{\mathbf{x}}_{k-1} - \mathbf{K}\mathbf{L}\mathbf{z}_k - \mathbf{K}(\mathbf{\Gamma}_2\mathbf{x}_r + \mathbf{\Gamma}_3\mathbf{u}_r) + \mathbf{K}\mathbf{x}_r + \mathbf{u}_r,$$

and then follow with computing the state estimate $\hat{\mathbf{x}}_k$. Such a change is recommended when timing is crucial, because it allows the actuator to send the control input to the plant faster, at the initial expense of four extra encrypted multiplication $\hat{\mathrm{E}}(\mathbf{K}\mathbf{\Gamma}_1), \hat{\mathrm{E}}(\mathbf{K}\mathbf{\Gamma}_2), \hat{\mathrm{E}}(\mathbf{K}\mathbf{\Gamma}_3)$, $\hat{\mathrm{E}}(\mathbf{K}\mathbf{L})$ that can be performed in Protocol 5.1.4.

5.1.6 Numerical results

5.1.6.1 Implementation details

The message space for AHE is discrete and the messages need to be represented on bits. Hence, we need to quantize the signals and their coefficients. We adopt a fixed-point number representation, where a value is represented as a signed integer in the two's complement format, with one sign bit, l_i integer bits and l_f fractional bits.

Most public space AHE schemes have the ring of integers $\mathbb{Z}_N$ as message space, where N is an RSA modulus. To prevent brute force factorization, N is required to be at least 1024 bits. Hence, the message space is large enough to represent messages with precision of 128 bits, which is the standard quadruple precision. In this case, the quantization and round-off errors can be considered negligible. Under stability conditions of the quantized matrices, the stability of quantized Kalman Filter can be proved [205]. The quantization effects of encryption over control performance are analyzed in [11, 138, 198, 199].

We assume the channels are reliable and the packets cannot be tampered with. The memory of the parties is finite, so we cannot consider that Protocol 5.1.3 runs for an infinite time. In the offline phase, the labels are generated for a fixed number of time steps T. Once these T time steps elapse, the parties have to generate new labels for the signals (not for the state matrices and gains, if those are not desired to be changed). This can be done in parallel with the computations that take place in the first T time steps.

5.1.6.2 Case study

We illustrate the performance of the proposed encrypted LQG controller on the problem of temperature regulation in a smart building, where a central cloud controller computes the control inputs in a private manner, based on encrypted data from sensors, so that sensitive information like the occupancy of the rooms is not revealed.

We consider the data from the HAMLab ISE model in [217], available at [118], which models the building as one zone. We use a modified model that considers the building to be split into two zones, which we assume have two different owners, who want different set points for the temperature in the zones, as well as privacy with respect to their presence there and desired settings. The state consists of the temperatures for each of the two zones (indoors air, floor, separating wall, internal facade and external facade temperature), the control input represents the heating/cooling for the two zones and the disturbances consist of the outdoors air temperature, the occupancy generated heat and the heat caused by the sun through the windows for the two zones. The measurements are considered to be the same as the states. The system has state dimension $n = 10$ and control input dimension $m = 2$. We simulate the results for a sampling time of 420 seconds. Assume the states corresponding to the indoors air temperature for the two zones have to be steered to 15°C and 25°C during the day (considered between 6 am to 8 pm), and to 10°C and 20°C during the night.

Each zone has a local device that generates secret keys for encrypting the labels. The matrices corresponding to the state model and the corresponding Kalman and control gains have been encrypted with labels generated by a different secret key by another machine in the building, called the Setup. The actuator generates a pair of master key and secret key and sends the master key to the three parties described so far, which encrypt their secret keys with the master key and send them to the actuator. The local devices for the two zones send the corresponding encrypted measurements and encrypted references to the cloud controller. If the cloud readily receives all the coefficients encrypted from the Setup, it executes Protocol 5.1.1, and otherwise, it executes Protocol 5.1.4 to obtain full LabHE encryptions of the gains and rest of the coefficients. The cloud then computes the estimate of the current state, refreshes the encryption with the help of the actuator, then computes the control input and sends it to the actuator, as described in Protocol 5.1.2.

We instantiated the AHE scheme with the Paillier scheme [177], for which we chose a modulus N of 1024 bits. The pseudorandom generator is chosen to be the SHA-3 hash function with 224 output bits. The protocols were run on a standard MacBook Pro laptop with a 2.2 GHz Intel Core i7 and 16 GB of RAM.

Figure 5.3 shows the performance of estimation and tracking for a fixed-point representation with $l_i = 24$ integer bits and $l_f = 24$ fractional bits, where r represents the set point, T^i_{in} is the true temperature in zone i and $\hat{T}^i_{in}$ is the temperature estimate.

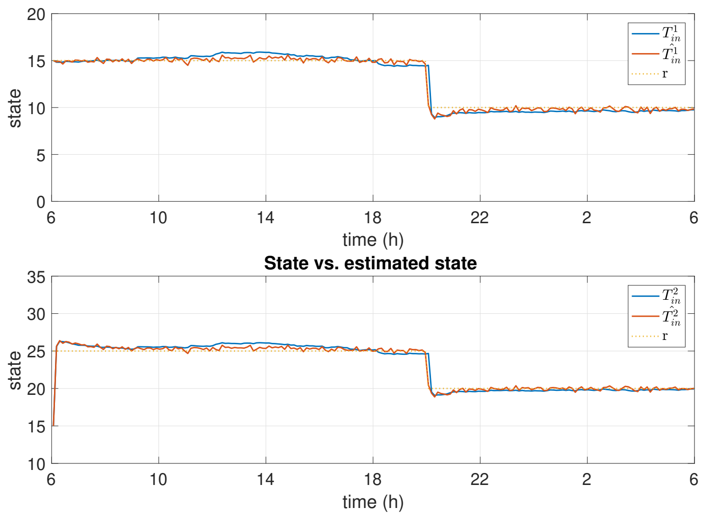


Figure 5.3: Performance of the estimation and tracking for encrypted LQG for $l_i = 24$, $l_f = 24$.

For a simulation of $T = 100$ time steps, corresponding roughly to 12 hours with the chosen sampling period, the offline execution times corresponding to the Protocol 5.1.1 and 5.1.4 and online execution times are shown in Table 5.1. The online time corresponding

Time (s)	Cloud	Actuator	Agent$_{1,2}$	Setup
Offline P. 5.1.1	0.228	5.721	1.138	1.239
Initialization P. 5.1.1	0.156	0	4e-5	0.0024
Offline P. 5.1.4	0.226	33.32	1.131	0.744
Initialization P. 5.1.4	21.417	0.603	4e-5	0.0012
Online for one step	0.219	0.0029	4e-5	0

Table 5.1: Average times for the encrypted LQG computation for $l_i = 24$, $l_f = 24$, $N_\sigma = 1024$, 224-bit secrets, 100 time steps.

to computing the estimate and control input in a time step are under 0.3 seconds for all actors. Since the sampling times for buildings are usually large, the encrypted computations finish much before a new measurement is acquired.

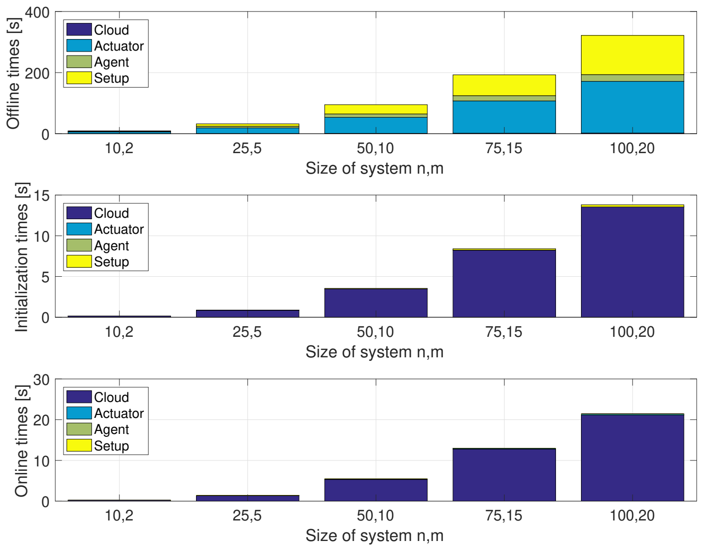


Figure 5.4: Execution times for Protocol 5.1.3 for different system dimensions, with $l_i = 24$, $l_f = 24$, $N_\sigma = 1024$, 224-bit secrets, and 100 time steps.

The execution times for varying system dimensions, considering one agent with all the states and control inputs for ease of comparison, are presented in Figure 5.4 and Figure 5.5. Figure 5.4 compares which actor performs the most computational intense task for each phase of Protocol 5.1.3, while Figure 5.5 shows how much time each actor spends on each phase using a logarithmic scale for visualization purposes. In order to evaluate the time performance of the proposed protocols, one should look at the online times for the subsystems and actuator, since these are the most resource-sensitive actors. We observe that even for larger systems of 100 states and 20 inputs, the online time for one iteration for the agent is 0.0005 seconds and 0.27580 seconds for the actuator, which are excellent times.

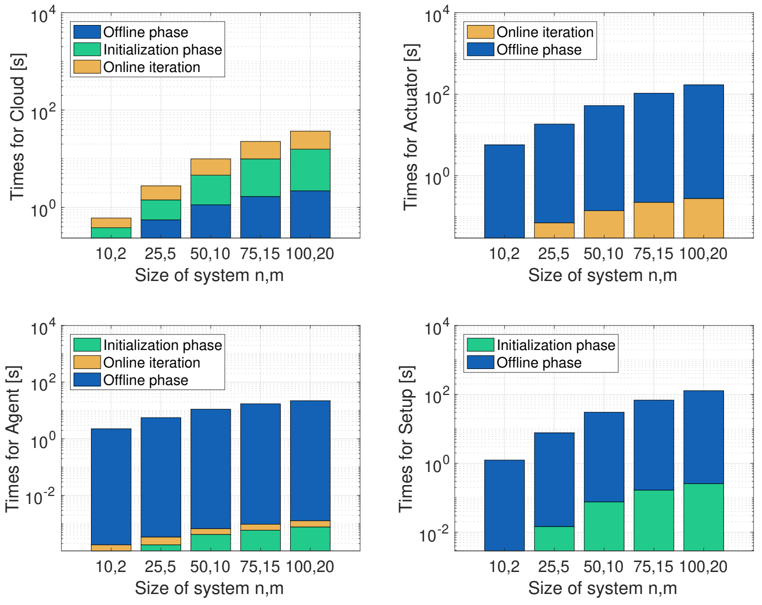


Figure 5.5: Execution times on a logarithmic scale for Protocol 5.1.3 for different system dimensions, with $l_i = 24$, $l_f = 24$, $N_\sigma = 1024$, 224-bit secrets, and 100 time steps.

Notice that in the initialization phase and in the online iterations, the cloud has the most computational requirements, which is as desired, since the cloud is a powerful server, and it has the necessary resources to improve the times. On the other hand, the rest of the parties are expected to perform more offline computations than the cloud, which correspond to the label generation and LabHE encryption process. The actuator is not required to do anything in the initialization phase corresponding to Protocol 5.1.1 and the setup does not play a role in the online iterations.

For larger system dimensions ($n \geq 50$), the offline and initialization phases become computationally and memory intensive for the execution of Protocol 5.1.4, due to computing $\mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3$ as products of encrypted matrices. As described in Section 5.1.3.1, the actuator is required to compute the encryptions of products of the monomials in the polynomial evaluations and transmit them to the cloud, which can take 4 hours on a 2.2 GHz Intel Core i7 processor. However, the online execution times remain the same as in Figures 5.4 and 5.5.

5.2 Secure evaluation of distributed linear control

5.2.1 Introduction

The recent drive towards increasing interconnectivity of systems has determined more and more systems to be operated using distributed control schemes. Examples such as smart grids, water-supply systems, robot swarms, or intelligent transportation systems benefit from this distributed computing framework. Applying distributed but cooperative controllers requires communication between the various subsystems or agents. In the resulting networked control system, sensible data is transmitted via possibly public networks and processed at neighboring agents, which can pose a privacy threat. Recent examples of data leakage and abuse in (critical) infrastructures, such as disturbing the normal functionality of power plants or inferring people's presence at home from smart meter measurements, have drawn attention to the risks of sharing data in the clear. The challenge thereby is to solve the conflict of ensuring individual privacy while simultaneously allowing for cooperation.

5.2.1.1 Related work

Encrypted distributed control calls for slightly different techniques than encrypted cloud-based control that we saw in the first part of the chapter. Recent works on encrypted consensus [112, 113, 192], on encrypted distributed optimization [95, 155, 231, 237] and in smart grid encrypted data aggregation [89, 142, 147] give insights about homomorphically encrypted distributed computation, but treat different aspects than the ones we are interested in for this section. It has recently been shown in [199] that encrypted cooperative control is capable of solving the privacy conflict using homomorphic encryption and allows secure interaction between the participating agents. However, while [199] demonstrates that private cooperative control is realizable, their proposed encrypted control scheme reveals more information about some participants' private local data than required to evaluate the local control laws. This results in a privacy leak that is difficult to address with existing approaches. We discuss more about this in Section 5.2.3.

5.2.1.2 Contributions

This chapter presents a scheme that we call *private control update aggregation* (pCUA), which ensures that an agent learns nothing apart from the unconcealable information, i.e., the sum of contributions from its neighbors. As a corollary, we close the identified privacy leak of the scheme in [199]. Specifically, the values of the states, control actions and control gains of each agent are hidden from the rest of the participants. To this end, we employ our *private weighted sum aggregation* (pWSAh) scheme with hidden weights from Chapter 4.3.2, where each agent acts as an aggregator for the contributions of its neighbors. The solution relies on secret sharing and additively homomorphic encryption. We implement the proposed solution for various connectivity degrees of a network and showcase its efficiency. Finally, we analyze the observability of the system from the perspective of malicious agents.

5.2.2 Problem setup

We consider a control scheme tailored for multi-agent systems with linear dynamics. Specifically, consider a system with M agents that obey the local dynamics:

$$\mathbf{x}_i(t+1) = \mathbf{A}_i\mathbf{x}_i(t) + \mathbf{B}_i\mathbf{u}_i(t), \quad \mathbf{x}_i(0) = \mathbf{x}_{i,0}, \tag{5.2.1}$$

with $\mathbf{x}_i \in \mathbb{R}^{n_i}$ and $\mathbf{u}_i \in \mathbb{R}^{m_i}$, for every $i \in [M]$. Assume that the agents are part of a simple, connected, fixed and undirected communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertex set denoted as $\mathcal{V} = [M]$ and the edge set denoted as $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. An edge $(i, j) \in \mathcal{E}$ specifies that agent i can communicate with agent j, in which case we say agent i and agent j are neighbors.

In a cooperative structured control, we can use the following local control laws to stabilize the systems, where $\mathcal{N}_i := \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\}$ represents the set of neighbors of agent i:

$$\mathbf{u}_i(t) = \mathbf{K}_{ii}\mathbf{x}_i(t) + \sum_{j\in\mathcal{N}_i} \mathbf{K}_{ij}\mathbf{x}_j(t), \tag{5.2.2}$$

These local control laws result from the design of a centralized linear controller of the form:

$$\begin{bmatrix} \mathbf{u}_1(t) \\ \vdots \\ \mathbf{u}_M(t) \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11} & \dots & \mathbf{K}_{1M} \\ \vdots & \ddots & \vdots \\ \mathbf{K}_{M1} & \dots & \mathbf{K}_{MM} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1(t) \\ \vdots \\ \mathbf{x}_M(t) \end{bmatrix} \tag{5.2.3}$$

that takes into account the structural constraints of the communication graph by requiring $\mathbf{K}_{ij} = \mathbf{0}$ whenever $j \notin \mathcal{N}_i \cup \{i\}$. The stabilizing local control feedback gains $\mathbf{K}_{ij}$ can be designed to take into account the structural constraints of the communication graph, see, e.g. [151].

5.2.3 Previous solutions for cooperative control

Local control laws of the form (5.2.2) were considered in [7, 12, 199]. Specifically, [199] introduces a private computation and exchange of the "input portions":

$$\mathbf{v}_{ij}(t) := \mathbf{K}_{ij}\mathbf{x}_j(t) \in \mathbb{R}^{m_i}. \tag{5.2.4}$$

to compute the local control input as in (5.2.2).

However, in order to compute the aggregated value $\mathbf{x}_a$, the aggregator does not require access to the individual $\mathbf{v}_{ij}$'s. In fact, if we want to ensure privacy of the data that is unknown to the aggregator, the best we can do is for the aggregator to only be able to compute the "aggregated portions":

$$\mathbf{u}_i(t) - \mathbf{v}_{ii}(t) = \mathbf{v}_{i,\mathcal{N}_i}(t) := \sum_{j\in\mathcal{N}_i} \mathbf{v}_{ij}(t). \tag{5.2.5}$$

The "input portions" $\mathbf{v}_i(t)$ reveal neither the exact local state $\mathbf{x}_i$ nor the local controller matrix $\mathbf{W}_i$ to the aggregator, but can at least reveal the relative rate of decrease/increase of some signals of the agents over multiple time steps. More details about the information leak can be found in Chapter 5.2.5 and in [12], where a solution to the pWSAh problem that achieves aggregator obliviousness is proposed. The solution in [12] required generating fresh secrets at every time step and proposed an online decentralized method that reduced the collusion threshold between participants.

Finally, [7] proposed a theoretical solution that addressed the two issues of [12] (this solution is described in Chapter 4.5): it reduced the number of generated secrets, kept the collusion threshold at $M-1$ corrupted participants and reduced the number of messages exchanged between the agents and aggregator. This comes at the cost of using a lattice-based homomorphic encryption scheme and augmented learning with error ciphertexts, which can be larger than Paillier ciphertexts and might require more expensive operations.

In the next section, we use our proposed solution for private weighted aggregation with hidden weights from Chapter 4.3.2, that provides a compact and efficient private weighted sum aggregation scheme by packing multiple values in one homomorphic Paillier ciphertext and an online decentralized method of distributing the secret shares of zero among the agents without reducing the collusion threshold.

5.2.4 Private control update aggregation and implementation

Each agent i is the aggregator of the contributions of its neighbors $\mathbf{K}_{ij}\mathbf{x}_j(t)$. There is a system operator that acts as the dealer, who designs and encrypts the control feedback weights $\mathbf{K}$. Notice that this formulation fits the pattern of weighted aggregation, described in Chapter 4. Specifically, the private control update aggregation scheme pCUA is composed of the following algorithms ($\text{Setup}_{\text{pCUA}}, \text{InitW}_{\text{pCUA}}\text{Enc}_{\text{pCUA}}, \text{AggrDec}_{\text{pCUA}}$):

- $\text{Setup}_{\text{pCUA}}(1^\kappa, \mathcal{V}, \mathcal{E}, \mathbf{A}, \mathbf{B}, T)$: The dealer generates the control gain $\mathbf{K}$ like in (5.2.3) using the system parameters $\mathbf{A}, \mathbf{B}$. For every agent $i \in \mathcal{V}$, run pWSAh*.Setup and generate a pair of keys $(\text{pk}_i, \text{sk}_i)$. For every $t \in [T]$, generate and denote by $\mathbf{S}_{ij}$ the matrix of secret shares of zero of agent j has for its contribution to agent i. Set $\text{prm} = (\kappa, \{\text{pk}_i\}_{i\in\mathcal{V}})$, $\mathfrak{sk}_i = (\text{sk}_i, \mathbf{K}_{ii}, \mathbf{S}_{ii}, \mathbf{S}_{j\in\mathcal{N}_i,i})$ for $i \in \mathcal{V}$.

- $\text{InitW}_{\text{pCUA}}(\text{prm}, \mathcal{V}, \mathcal{E}, \{\mathbf{K}_{ij}\}_{i\in\mathcal{V}, j\in\mathcal{N}_i})$: given the public key of the Paillier scheme for agent $i \in \mathcal{V}$, encrypt $\mathbf{K}_{ji}$ column-wise as in pWSAh*.InitW and return to agent $j \in \mathcal{V}$: $\mathbf{sw}_j = \{\text{E}(\text{pk}_i, \mathbf{K}_{ij})\}_{i\in[\mathcal{N}_j]}$.

- $\text{Enc}_{\text{pCUA}}(\text{prm}, \mathbf{sw}_i, \mathfrak{sk}_i, t, \mathbf{x}_i(t))$: For each $i \in \mathcal{V}$ and $j \in \mathcal{N}_i$, computes $\mathbf{c}_{j,i}(t)$ as in pWSAh*.Enc.

- $\text{AggrDec}_{\text{pCUA}}(\text{prm}, \mathfrak{sk}_i, t, \{\mathbf{c}_{ij}(t)\}_{i,j\in\mathcal{N}_i})$: For each $i \in \mathcal{V}$, aggregate and decrypt the neighbors ciphertexts as in pWSAh*.AggrDec then add the local value $\mathbf{K}_{ii}\mathbf{x}_i(t)$ to obtain $\mathbf{u}_i(t)$.

In the rest of this section, we will directly apply the methods described in Chapter 4 as illustrated above and showcase the numerical performance. Specifically, consider a network of 50 agents, with each agent having local states and local control inputs both of dimension 6. We simulate pWSAh* for various values of the average node degree in the network, obtained by varying the probability of drawing edges between agents. Simulations were run in Python 3 on a 2.2 GHz Intel Core i7 processor. As usual when operating with encryption schemes defined on groups of modular residues, we need to use a fixed-point representation scheme in order to quantize the real values and encode them into integers. The effect of this quantization on the cooperative control scheme is described in [199]. In the simulations, we choose the message representation to be on $l = 32$ bits: 16 integer bits and 16 fractional bits, the statistical security size to be 80 bits and the Paillier moduli for each agent to have 2048 bits. With these chosen values, all 6 elements of the local contributions can be encoded into a single Paillier ciphertext in pWSAh*.

We present the simulation results for the algorithmic solutions described in Sections 4.3.2, 4.3.3.1 and 4.3.3.2, in both naive implementation (Section 4.4.2.4) and using packing (Section 4.4.2.5). The running times are averaged over 50 instances and represent the total time it takes for an agent at a time step to generate and distribute the secret shares for the computation for itself and its neighbors *and* to aggregate the contributions of its neighbors *and* to compute and send out its own contribution to its neighbors. The shares are locally encrypted with an AES cipher with 128-bit key–for the offline centralized phase, each agent has an AES key with the dealer, and each pair of neighbors has their own AES key for the online decentralized phase. Figures 5.6, 5.7, 5.8 and 5.10 show the times for an agent with the average connectivity degree, respectively the minimum and maximum connectivity degree (represented by the arrows).

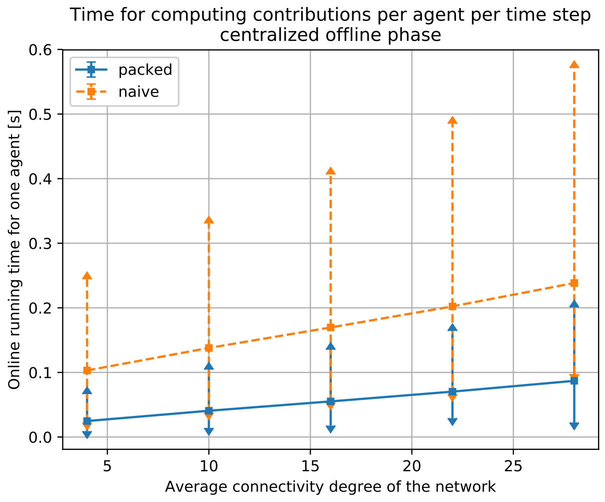


Figure 5.6: Average running times for the pWSAh* scheme with the steps described in Section 4.3.2 in a network of 50 agents.

Figure 5.6 compares the running times for the local computation at each agent in a time step using the scheme with centralized offline generation of the secret shares, between naive and packed encryption. This method has the smallest online running time for agents, but the largest offline time for the dealer that generates the shares for all agents, for many time steps (see Figure 5.9). Figure 5.7 compares the online running times between naive and packed encryption in the case of the one-step decentralized online generation of shares and Figure 5.8 for the two-step decentralized online generation of shares. These methods have roughly an eight-fold increase in the online time compared to the method that makes heavy use of a trusted dealer, but the dealer has less work to do in the offline phase. As expected, less security, in terms of reducing the collusion threshold, yields better online time (Figure 5.7 vs. Figure 5.8).

In the centralized offline share generation scheme, packing decreases the maximum online time between 64% and 71%. In the online decentralized cases, where the agents are also responsible for generating, encrypting, sending and decrypting the shares, packing reduces the maximum online running time between 76% and 80%. Overall, we see that in the packed version, the sampling time needs to be at most 1.1 seconds.

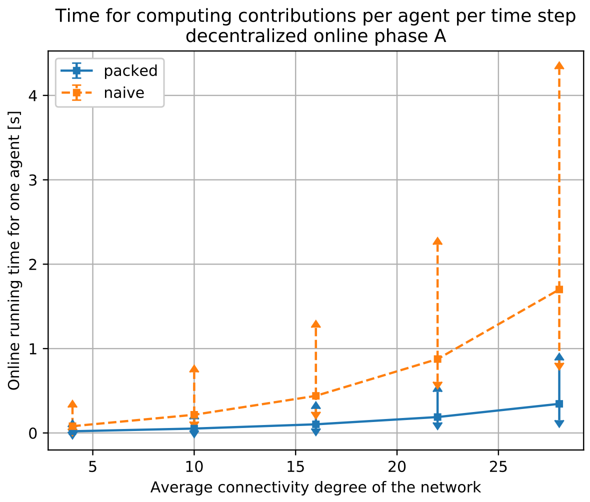


Figure 5.7: Average running times for the pWSAh* scheme with the steps described in Section 4.3.3 in a network of 50 agents.

Second, the communication load is reduced when using packing: one Paillier ciphertext of 0.256 KB is sent from the neighboring agents to the aggregating agent, instead of six ciphertexts amounting to 1.536 KB, and each agent sends four batches of 16 bytes AES-encrypted shares to each neighbor, instead of seven batches of 16 bytes.

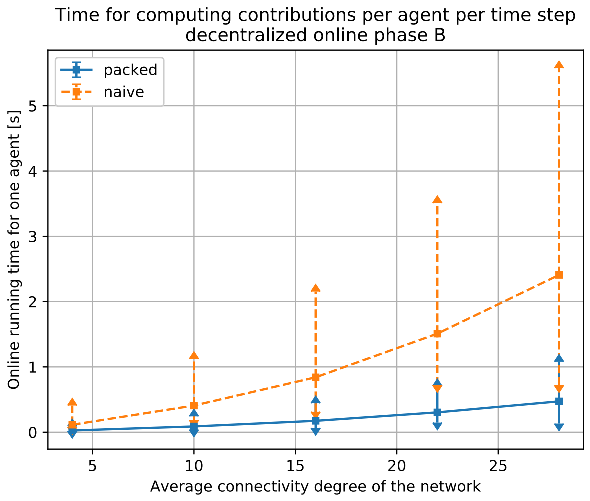


Figure 5.8: Average running times for the pWSAh* scheme with the steps described in Section 4.3.3.2 in a network of 50 agents.

A third substantial advantage of packing is decreasing the offline time, consisting of the weights encryption and the share generation and encryption at the system operator, depicted

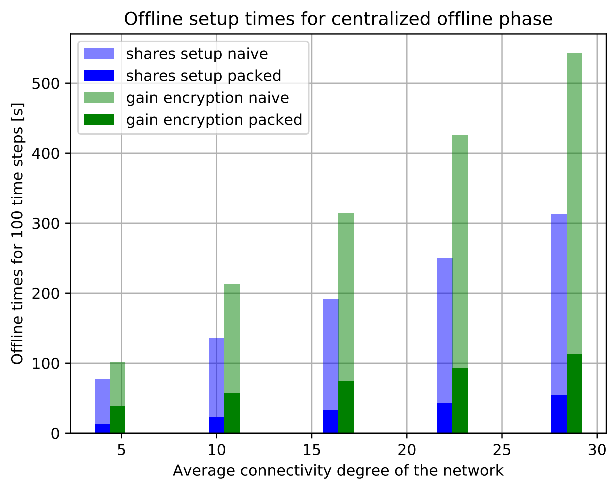


Figure 5.9: Offline setup phase running times for the pWSAh* scheme as in Section 4.3.2 in a network of 50 agents with shares generated for 100 time steps. The offline setup times for the decentralized share generation schemes in Section 4.3.3 consist only of the gain encryption times.

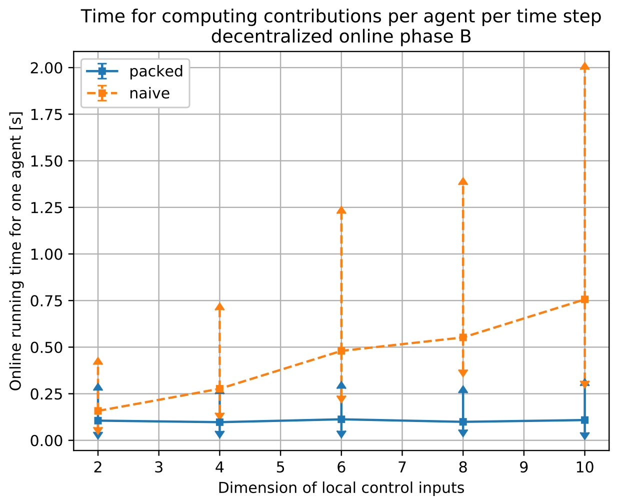


Figure 5.10: Average running times for the pWSAh* scheme with the steps described in Section 4.3.3.2 in a network of 25 agents.

in Figure 5.9. Specifically, we see up to 80% improvement in the offline running time when using packing.

To further illustrate the efficiency of packing, we show in Figure 5.10 how the performance improves with the number of control inputs, i.e., the number of values that are packed into one value, in the case of decentralized online share generation. We simulate a network of

25 agents with an average network connectivity degree of 10. Each agent has a local state of dimension 10 and local control input of dimension varying between 2 and 10, which are packed in one Paillier ciphertext. The online running time remains almost the same in the packed version, despite having more control inputs, compared to the increasing online time in the naive case.

5.2.5 Observability analysis

This section motivates the benefits of aggregation for the privacy of the agents from a system-theoretic point of view.

5.2.5.1 Available information without aggregation

In the encrypted control scheme [199], agent i has access to the input portions $\mathbf{v}_{ij}(t)$ of all neighboring agents $j \in \mathcal{N}_i$. Thus, it can use (5.2.4) and stored data to estimate the matrices $\mathbf{K}_{ij}$ or the neighboring states $\mathbf{x}_j(t)$. Under the assumption that $m_i \leq n_i$, the corresponding system of equations:

$$\begin{bmatrix} \mathbf{v}_{ij}(0) & \dots & \mathbf{v}_{ij}(t) \end{bmatrix} = \mathbf{K}_{ij} \begin{bmatrix} \mathbf{x}_j(0) & \dots & \mathbf{x}_j(t) \end{bmatrix} \tag{5.2.6}$$

is underdetermined for every t. However, taking application-related restrictions on $\mathbf{K}_{ij}$ and $\mathbf{x}_j$ into account, it might be possible to reconstruct the constant matrix $\mathbf{K}_{ij}$. Moreover, especially for systems with similar agents, such as robot swarms, agent i might obtain information about the dynamics of agent j in terms of $\mathbf{A}_j$ and $\mathbf{B}_j$. The ability to reconstruct $\mathbf{x}_j$ from observations $\mathbf{v}_{ij}$ given $\mathbf{K}_{ij}$, $\mathbf{A}_j$, and $\mathbf{B}_j$ then becomes an observability problem. In fact, observing $\mathbf{x}_j$ requires full rank of the observability matrix:

$$\mathbf{V}_{ij} := \begin{bmatrix} \mathbf{K}_{ij}\mathbf{A}_j^0 \\ \vdots \\ \mathbf{K}_{ij}\mathbf{A}_j^{n_j-1} \end{bmatrix}.$$

5.2.5.2 Available information with aggregation

In the aggregated case described in Section 5.2.4, less data is available to agent i. The system of equations (5.2.6) becomes:

$$\begin{bmatrix}\mathbf{v}_{i,\mathcal{N}_i}(0) & \dots & \mathbf{v}_{i,\mathcal{N}_i}(t)\end{bmatrix} = \mathbf{K}_{i,\mathcal{N}_i}\begin{bmatrix}\mathbf{x}_{\mathcal{N}_i}(0) & \dots & \mathbf{x}_{\mathcal{N}_i}(t)\end{bmatrix},$$

with $\mathbf{K}_{i,\mathcal{N}_i} := \begin{bmatrix}\mathbf{K}_{ij_1} \dots \mathbf{K}_{ij_{|\mathcal{N}_i|}}\end{bmatrix}$, $\mathbf{x}_{\mathcal{N}_i} := \begin{bmatrix}\mathbf{x}_{j_1}^\intercal \dots \mathbf{x}_{j_{|\mathcal{N}_i|}^\intercal}\end{bmatrix}^\intercal$, and $\mathcal{N}_i = \{j_1, \dots, j_{|\mathcal{N}_i|}\}$. For $|\mathcal{N}_i| > 1$, the degree of freedom here is significantly higher than in (5.2.6). Still, it might again be possible to infer the constant matrix $\mathbf{K}_{i,\mathcal{N}_i}$. In addition, agent i might know the neighboring dynamics: $\mathbf{A}_{\mathcal{N}_i} := \mathbf{blkdiag}(\mathbf{A}_{j_1}, \dots, \mathbf{A}_{j_{|\mathcal{N}_i|}})$. However, the observability of $\mathbf{x}_{\mathcal{N}_i}$ now requires full rank of:

$$\mathbf{V}_{i,\mathcal{N}_i} := \begin{bmatrix}\mathbf{K}_{i,\mathcal{N}_i}\mathbf{A}_{\mathcal{N}_i}^0 \\ \vdots \\ \mathbf{K}_{i,\mathcal{N}_i}\mathbf{A}_{\mathcal{N}_i}^{n_j-1}\end{bmatrix}.$$

The relation $\mathbf{K}_{i,\mathcal{N}_i}\mathbf{A}_{\mathcal{N}_i}^k = \begin{bmatrix}\mathbf{K}_{ij_1}\mathbf{A}_{j_1}^k \dots \mathbf{K}_{ij_{|\mathcal{N}_i|}}\mathbf{A}_{j_{|\mathcal{N}_i|}}^k\end{bmatrix}$, combined with an analysis of the $|\mathcal{N}_i|$ column-blocks of $\mathbf{V}_{i,\mathcal{N}_i}$ and the Cayley-Hamilton theorem lead to the following result.

Lemma 5.2.1. *The full rank of $\mathbf{V}_{i,\mathcal{N}_i}$ implies the full rank of $\mathbf{V}_{ij}$ for every $j \in \mathcal{N}_i$.*

Lemma 5.2.1 states that observability in the aggregated case implies observability in the unaggregated case. In other words, observability in the aggregated case is less likely, which increases the privacy of the agents. Note that the converse statement of Lemma 5.2.1 is, in general, not true.

Chapter 6

Model Predictive Control

This chapter explores the privacy of cloud outsourced Model Predictive Control (MPC) for a linear system with input constraints. In our cloud-based architecture, a client sends the private states to the cloud who performs the MPC computation and returns the control inputs. In order to guarantee that the cloud can perform this computation without obtaining *anything* about the client's private data, we employ homomorphic cryptosystems and oblivious communication tools. In the first part of the chapter, we consider an MPC application where only signals such as the states and control inputs are required to be private, but the constant parameters and costs can be public. We propose protocols using an additively homomorphic encryption scheme for two cloud-MPC architectures motivated by the current developments in the Internet of Things: a client-server architecture and a two-server architecture. In the first case, a control input for the system is privately computed by the cloud server, with the assistance of the client. In the second case, the control input is privately computed by two independent, non-colluding servers, with no additional requirements from the client. We prove that the proposed protocols preserve the privacy of the client's data and of the resulting control input. Furthermore, we compute bounds on the errors introduced by encryption. We present numerical simulations for the two architectures and discuss the trade-off between communication, MPC performance and privacy.

In the second part of the chapter, we consider a more restrictive MPC application where both signals and the constant parameters are required to be private. Moreover, we consider the system to be split into multiple untrusted subsystems, and neither the cloud controller, nor the actuator can be trusted. We propose a protocol that extends the above private two-server protocol to work with somewhat homomorphic encryption and we prove it achieves privacy even under collusions between the participants.

This chapter covers the work presented in [9, 11].

6.1 Introduction

Model Predictive Control (MPC) is a powerful scheme that is successfully deployed in practice [162] for systems of varying dimension and architecture, including on cloud platforms. In competitive scenarios, such as energy generation in the power grid, domestic scenarios, such as heating control in smart houses, or time-sensitive scenarios, such as traffic control, the control scheme should come with privacy guarantees to protect from eavesdroppers or from an untrustworthy cloud. For instance, in smart houses, client-server setups can be envisioned, where a local trusted computer aggregates the measurements from the sensors, but does not store their model and specifications, and depends on a server to compute the control input or reference. The server can also posses other information, such as the weather. In a heating application, the parameters of the system can be known by the server, i.e., the energy consumption model of the house, but the data measurements and how much the owner wants to consume should be private. In traffic control, the drivers are expected to share their locations, which should remain private, but are not expected to contribute to the computation. Hence, the locations are collected and processed at the server's level, which then sends the result back to the cars or to traffic lights.

Contributions

We investigate a privacy-preserving cloud-based Model Predictive Control application, which at a high level, requires privately computing additions, multiplications, comparisons and oblivious updates. Our solution encompasses several SMPC techniques: homomorphic encryption, secret sharing, oblivious transfer.

In Section 6.2, we discuss the implicit MPC computation for a linear system with input constraints, where we privately compute a control input, while maintaining the privacy of the state, using an additively homomorphic cryptosystem. In the first case we consider, the control input is privately computed by a server, with the help of the client. In the second case, the computation is performed by two non-colluding servers. The convergence of the state trajectory to the reference is public knowledge, so it is crucial to not reveal anything else about the state and other sensitive quantitities. Therefore, we use a privacy model that stipulates that *no computationally efficient algorithm run by the cloud can infer anything about the private data*, or, in other words, an adversary doesn't know more about the private data than a random guess. Although this model is very strict, it thoroughly characterizes the loss of information.

This work explores a fundamental issue of privacy in control: the trade-off between computation, communication, performance and privacy. Our main contributions are two privacy-preserving protocols for MPC and the analysis of the errors induced by the encryption.

In Section 6.3, we extend the results from Section 6.2 to account for more privacy requirements of the data in the system: we require the private computation of the control input, while maintaining both the privacy of the state and the privacy of the system model and MPC parameters. Furthermore, we consider that the system to be controlled is comprised of multiple subsystems and there is no client that holds all the information in the clear. By using a somewhat homomorphic encryption scheme that allows multiple keys, we can securely evaluate the MPC functionality on the data coming from separate subsystems, without revealing private information.

Related work

In control systems, ensuring the privacy of the measurements and control inputs from eavesdroppers and from the controller has been so far tackled with differential privacy, homomorphic encryption and transformation methods. Kalman filtering with DP was addressed in [144], current trajectory hiding in [141], linear distributed control [224], and distributed MPC in [233]. The idea of encrypted controllers was introduced in [139] and [92], using

AHE, and in [134] using FHE. Kalman filtering with AHE was further explored in [227].

Recent work in [198] has tackled the problem of privately computing the input for a constrained linear system using explicit MPC, in a client-server setup. There, the client performs the computationally intensive trajectory localization and sends the result to the server, which then evaluates the corresponding affine control law on the encrypted state using AHE. Although explicit MPC has the advantage of computing the parametric control laws offline, the evaluation of the search tree at the cloud's level is intractable when the number of nodes is large, since all nodes have to be evaluated in order to not reveal the polyhedra the in which the state lies, and comparison cannot be performed locally on encrypted data. Furthermore, the binary search in explicit MPC is intensive and requires the client to store all the characterization of the polyhedra, which we would like to avoid. Taking this into consideration, we focus on implicit MPC.

The performance degradation of a linear controller due to encryption is analyzed in [138]. In our work, we investigate performance degradation for the nonlinear MPC control.

Special notation for this chapter

Given a real quantity $x \in \mathbb{R}$, we use the notation $\bar{x}$ for the corresponding quantity in fixed-point representation on one sign bit, l_i integer and l_f fractional bits.

6.2 Secure evaluation of MPC with public parameters

6.2.1 Problem setup

We consider a discrete-time linear time-invariant system:

$$\mathbf{x}(t+1) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \tag{6.2.1}$$

with the state $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ and the control input $\mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^m$. The Model Predictive Control (MPC) is the optimal control receding horizon problem with constraints written as:

$$\begin{aligned} J_N^*(\mathbf{x}(t)) = & \min_{\mathbf{u}_0,\dots,\mathbf{u}_{N-1}} \frac{1}{2}\left(\mathbf{x}_N^\intercal \mathbf{P}\mathbf{x}_N + \sum_{k=0}^{N-1} \mathbf{x}_k^\intercal \mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\intercal \mathbf{R}\mathbf{u}_k\right) \\ & \text{s.t. } \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k,\ k = 0,\dots,N-1;\quad \mathbf{x}_0 = \mathbf{x}(t); \\ & \mathbf{u}_k \in \mathcal{U},\ k = 0,\dots,N-1, \end{aligned} \tag{6.2.2}$$

where N is the horizon length and $\mathbf{P}, \mathbf{Q}, \mathbf{R} \succ 0$ are cost matrices. We consider input constrained systems: $\mathcal{X} = \mathbb{R}^n$, $\mathbf{0} \in \mathcal{U} = \{\mathbf{l}_u \preceq \mathbf{u} \preceq \mathbf{h}_u\}$, and impose stability without a terminal state constraint, $\mathcal{X}_f = \mathbb{R}^n$, but with appropriately chosen costs $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ and horizon N such that the closed-loop system has robust performance to bounded errors due to encryption. A survey on the conditions for stability of MPC is given in [163]. We chose the input constrained formulation for reasons related to error bounding, explained in Section 6.2.4.

Through straightforward manipulations, (6.2.2) can be written as a quadratic program (see details on obtaining the matrices $\mathbf{H}$ and $\mathbf{F}$ in [44, Ch. 8,11]) in the variable $\mathbf{U} := \left[\mathbf{u}_0^\intercal\ \mathbf{u}_1^\intercal\ \dots\ \mathbf{u}_{N-1}^\intercal\right]^\intercal$:

$$\mathbf{U}^*(\mathbf{x}(t)) \ = \arg\min_{\mathbf{U}\in\mathcal{U}} \frac{1}{2}\mathbf{U}^\intercal \mathbf{H}\mathbf{U} + \mathbf{U}^\intercal \mathbf{F}^\intercal \mathbf{x}(t). \tag{6.2.3}$$

For simplicity, we keep the same notation for the augmented constraint set $\mathcal{U}$. After obtaining the optimal solution, the first m components of $\mathbf{U}^*(\mathbf{x}(t))$ are applied as input to the system (6.2.1): $\mathbf{u}^*(\mathbf{x}(t)) = \{\mathbf{U}^*(\mathbf{x}(t))\}_{1:m}$. This problem easily extends to the case of following a reference.

6.2.1.1 Solution without privacy requirements

The constraint set $\mathcal{U}$ is a hyperbox, so the projection step required for solving (6.2.3) has a closed form solution, denoted by $\Pi_{\mathcal{U}}(\cdot)$ and the optimization problem can be efficiently solved with the projected Fast Gradient Method (FGM) [173], given in (6.2.4):

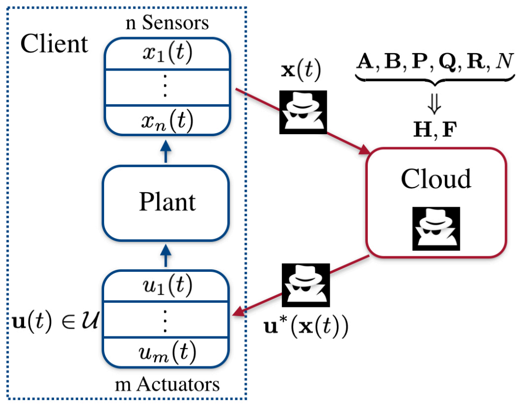
$$\begin{aligned}
&\text{For } k = 0\dots, K-1 \\
&\qquad \mathbf{t}_k \leftarrow \left(\mathbf{I}_{Mm} - \frac{1}{L}\mathbf{H}\right)\mathbf{z}_k - \frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t) && \text{(6.2.4a)} \\
&\qquad \mathbf{u}_{k+1} \leftarrow \Pi_{\mathcal{U}}(\mathbf{t}_k) && \text{(6.2.4b)} \\
&\qquad \mathbf{z}_{k+1} \leftarrow (1+\eta)\mathbf{u}_{k+1} - \eta \mathbf{u}_k && \text{(6.2.4c)}
\end{aligned}$$

where $\mathbf{z}_0 \leftarrow \mathbf{u}_0$. The objective function is strongly convex, since $\mathbf{H} \succ 0$, therefore we can use the constant step sizes $L = \lambda_{max}(\mathbf{H})$ and $\eta = (\sqrt{\kappa(\mathbf{H})} - 1)/(\sqrt{\kappa(\mathbf{H})} + 1)$, where $\kappa(\mathbf{H})$ is the condition number of $\mathbf{H}$. Warm starting can be used at subsequent time steps of the receding horizon problem by using part of the previous solution $\mathbf{u}_K$ to construct a feasible initial iterate of the new optimization problem.

6.2.1.2 Privacy objectives

The unsecure cloud-MPC problem is depicted in Figure 6.1. In this section, the system's constant parameters $\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}, N$ are considered public, motivated by the fact the parameters are intrinsic to the system and hardware, and could be guessed or identified; however, the measurements, control inputs and constraints are not known and should remain private. Nevertheless, in Section 6.3.2, we provide a secure solution when the system's constant parameters are desired to be private, motivated in the context of critical infrastructure. The goal of this work is to devise private cloud-outsourced versions of (6.2.4) such that the client obtains the control input $\mathbf{u}^*(t)$ for system (6.2.1) with only a minimum amount of computation. The cloud (consisting of either one or two servers) should not infer anything else than what was known prior to the computation about the measurements $\mathbf{x}(t)$, the control inputs $\mathbf{u}^*(t)$ and the constraints $\mathcal{U}$. We tolerate *semi-honest* servers, meaning that they correctly follow the steps of the protocol but may store the transcript of the messages exchanged and process the data received to try to learn more information than allowed.

The purpose of the section is to design protocols with the functionality of (6.2.4) that



Client
n Sensors
$x_1(t)$
$x_n(t)$
Plant
$u_1(t)$
$u_m(t)$
$\mathbf{u}(t) \in \mathcal{U}$
m Actuators
$\mathbf{x}(t)$
$\underbrace{\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}, N}$
$\Downarrow$
$\mathbf{H}, \mathbf{F}$
Cloud
$\mathbf{u}^*(\mathbf{x}(t))$

Figure 6.1: Unsecure MPC: the system model $\mathbf{A}, \mathbf{B}$, horizon N and costs $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ are public. The state $\mathbf{x}(t)$, control input $\mathbf{u}^*(t)$ and input constraints $\mathcal{U}$ are privacy-sensitive.

satisfy client data confidentiality for one or two servers, as described in Definition 2.2.8 or Definition 3.3.1. To this end, we use the additively encryption scheme defined in Preamble 2.5.1. We discuss in Section 6.2.4 how we connect the domain of the inputs in the Paillier encryption scheme with the domain of real numbers needed for the MPC problem. In Sections 6.2.2 and 6.2.3, we address two private cloud-MPC solutions that present a trade-off between the computational effort at the client and the total time required to compute the solution $\mathbf{u}^*(t)$, which is analyzed in Section 6.2.5.

6.2.2 Client-server architecture

To be able to use the Paillier encryption, we need to represent the messages on a finite set of integers, parametrized by N_σ, i.e., each message is an element in $\mathbb{Z}_{N_\sigma}$. Usually, the values less than $N_\sigma/3$ are interpreted to be positive, the numbers between $2N_\sigma/3$ and N_σ to be negative, and the rest of the range allows for overflow detection. In this section and Section 6.2.3, we consider a fixed-point representation of the values and perform implicit multiplication steps to obtain integers and division steps to retrieve the true values. We analyze the implications of the fixed-point representation over the MPC solution in Section 6.2.4.

We introduce a client-server model, depicted in Figure 6.2a. We present an interactive protocol that privately computes the control input for the client, while maintaining the privacy of the state in Protocol 6.2.1. The client generates the public key pk and secret key sk of a Paillier encrypted scheme and encrypts its data. The cloud will thus perform computations on the encrypted data of the client.

The Paillier encryption is not order preserving, so the projection operation cannot be performed locally by the server. Hence, the server sends the encrypted iterate $[[\mathbf{t}_k]]$ for the client to project it. Then, the client encrypts the feasible iterate and sends it back to the cloud.

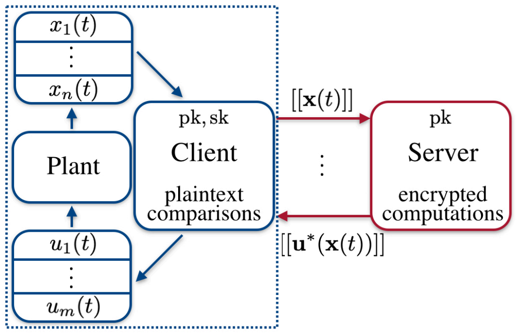
We drop the $\bar{(\cdot)}$ from the iterates in order to not burden the notation.

PROTOCOL 6.2.1: Encrypted projected Fast Gradient Descent in a client-server architecture

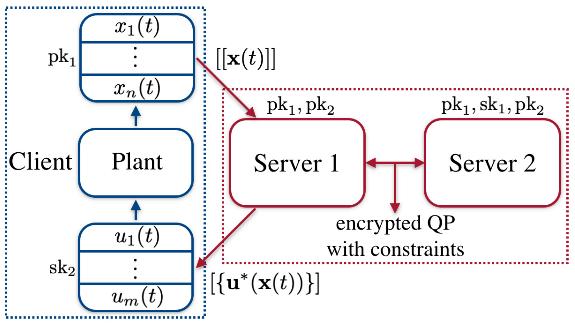
```
Input: $C$: $\mathbf{x}(t), K_c, K_w, l_i, l_f, \bar{\mathcal{U}}$, pk, sk; $S$: $\bar{\mathbf{H}}_f, \bar{\mathbf{F}}, \bar{\eta}, K_c, K_w, l_i, l_f$, pk, cold-start, $[[\mathbf{U}_w]]$
Output: $C$: $\mathbf{u} = (\mathbf{U}_K(\mathbf{x}(t)))_{1:m}$
 1: $C$: Encrypt and send $[[\mathbf{x}(t)]]$ to $S$
 2: if cold-start then
 3:     $S$: $[[\mathbf{U}_0]] = [[\mathbf{0}_{Nm}]]$; $C, S$: $K \leftarrow K_c$
 4: else
 5:     $S$: $[[\mathbf{U}_0]] = \big[[[(\mathbf{U}_w)_{m+1:Nm}]], [[\mathbf{0}_m]]\big]$; $C, S$: $K \leftarrow K_w$
 6: end if
 7: $S$: $[[\mathbf{z}_0]] = [[\mathbf{U}_0]]$
 8: for k=0...,K-1 do
 9:     $S$: $[[\mathbf{t}_k]] = (\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f) \otimes [[\mathbf{z}_k]] \oplus (-\bar{\mathbf{F}}_f^{\intercal}) \otimes [[\mathbf{x}(t)]]$ and send it to $C$
10:     $C$: Decrypt $\mathbf{t}_k$ and truncate to $l_f$ fractional bits
11:     $C$: $\mathbf{U}_{k+1} = \Pi_{\bar{\mathcal{U}}}(\mathbf{t}_k)$                ▷ Projection on $\bar{\mathcal{U}}$
12:     $C$: Encrypt and send $[[\mathbf{U}_{k+1}]]$ to $S$
13:     $S$: $[[\mathbf{z}_{k+1}]] = (1 + \bar{\eta}) \otimes [[\mathbf{U}_{k+1}]] \oplus (-\bar{\eta}) \otimes [[\mathbf{U}_k]]$
14: end for
15: $C$: Decrypt and output $\mathbf{u} = (\mathbf{U}_K)_{1:m}$
```

Theorem 6.2.1. *Protocol 6.2.1 achieves privacy as in Definition 2.2.8 with respect to a semi-honest server.*

Proof. The initial value of the iterate does not give any information to the server about the result, as the final result is encrypted and the number of iterations is a priori fixed. The view of the server, as in Definition 2.2.8, is composed of the server's inputs, the messages received $\{[[\mathbf{U}_k]]\}_{k=0,\dots,K}$, which are all encrypted, and no output. We construct a simulator which replaces the messages with random encryptions of corresponding length. Due to the semantic security of the Paillier cryptosystem, which was proved in [177], the view of the simulator is computationally indistinguishable from the view of the server. $\square$



(a) Private client-server MPC setup for a plant.



(b) Private two-server MPC setup for a plant.

Figure 6.2: Secure MPC solution.

6.2.3 Two-server architecture

Although in Protocol 6.2.1 the client needs to store and process substantially less data than the server, the computational requirements might be too stringent for large values of K and N_σ. In such a case, we outsource the problem to two servers, and only require the client to encrypt $\mathbf{x}(t)$ and send it to one server and decrypt the received the result $[[\mathbf{u}^*]]$. In this setup, depicted in Figure 6.2b, the existence of two non-colluding servers is assumed.

In Figure 6.2b and in Protocol 6.2.2, we will denote by $[[\cdot]]$ a message encrypted with pk$_1$ and by $[\{\cdot\}]$ a message encrypted by pk$_2$. The reason we use two pairs of keys is so the client and support server do not have the same private key and do not need to interact.

As before, we need an interactive protocol to achieve the projection. We use the ideas from the private solution for a quadratic optimization problem outlined in Section 3.2, but we work in the primal problem rather than in the dual. We use the DGK comparison protocol, proposed in [77, 78], such that, given two encrypted values of l bits $[[a]], [[b]]$ to

S_1, after the protocol, S_2, who has the private key, obtains a bit $(\beta = 1) \equiv (a \leq b)$, without finding anything about the inputs. Moreover, S_1 finds nothing about β. We augment this protocol by introducing a step (in line 11 of Protocol 6.2.2) before the comparison, in which S_1 randomizes the order of the two values to be compared, such that S_2 does not know the significance of β with respect to the inputs. Furthermore, by performing a blinded exchange, S_1 obtains the minimum (respectively, maximum) value of the two inputs, without any of the two servers knowing what the result is. This is achieved via the oblivious transfer described in Preamble 2.6. The above procedure is performed in line 13 in Protocol 6.2.2.

The comparison protocol works with inputs that are represented on l bits. The variables we compare are results of additions and multiplications, which can increase the number of bits, thus, we need to ensure that they are represented on l bits before inputting them to the comparison protocol. This introduces an extra step in line 10 in which S_1 communicates with S_2 in order to obtain the truncation of the comparison inputs: S_1 adds noise to t_k and sends it to S_2 which decrypts it, truncates the result to l bits and sends it back. S_1 then subtracts the truncated noise.

In order to guarantee that S_2 does not find out the private values after decryption, S_1 adds a sufficiently large random noise to the private data. The random numbers used for blinding are chosen from $(0, 2^{l+\lambda_\sigma}) \cap \mathbb{Z}_{N_\sigma}$, which ensures the statistical indistinguishability between the sum of the random number and the private value and a random number of equivalent length [45], where λ_σ is the statistical security parameter.

Lines 10–14 and lines 17–19 of Protocol 6.2.2 are performed element-wise on the encrypted vectors.

PROTOCOL 6.2.2: Encrypted projected Fast Gradient Descent in a two-server architecture

Input: C : $\mathbf{x}(t), \text{pk}_{1,2}, \text{sk}_2$; S_1 : $\bar{\mathbf{H}}_f, \bar{\mathbf{F}}, \bar{\eta}, K_c, K_w, l_i, l_f, \text{pk}_{1,2}$, cold-start, $[[\mathbf{U}_w]]$; S_2 : K_c, K_w, l_i, l_f, $\text{pk}_{1,2}, \text{sk}_1$, cold-start

Output: C: $\mathbf{u} = (\mathbf{U}_K(\mathbf{x}(t)))_{1:m}$

```
1: $C$: Encrypt and send $[[\mathbf{x}(t)]], [[\mathbf{h}_u]], [[\mathbf{l}_u]]$ to $S_1$
2: **if** cold-start **then**
```

```
3:     $S_1$: $[[\mathbf{U}_0]] \leftarrow [[\mathbf{0}_{Nm}]]$; $S_1, S_2$:$K \leftarrow K_c$
4: else
5:     $S_1$: $[[\mathbf{U}_0]] \leftarrow \big[[[(\mathbf{U}_w)_{m+1:Nm}]], [[\mathbf{0}_m]]\big]$; $S_1, S_2$:$K \leftarrow K_w$
6: end if
7: $S_1$: $[[\mathbf{z}_0]] \leftarrow [[\mathbf{U}_0]]$
8: for k=0...,K-1 do
9:     $S_1$: $[[\mathbf{t}_k]] \leftarrow (\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f) \otimes [[\mathbf{z}_k]] \oplus (-\bar{\mathbf{F}}_f^\intercal) \otimes [[\mathbf{x}(t)]]$
10:    $S_1, S_2$: $[[\mathbf{t}_k]] \leftarrow$ truncate $[[\mathbf{t}_k]]$
11:    $S_1$: $\mathbf{a}_k, \mathbf{b}_k \leftarrow$ randomize $[[\mathbf{t}_k]], [[\mathbf{h}_u]]$
12:    $S_1, S_2$: perform Protocol 2.7.2 s.t. $S_2$ obtains $(\boldsymbol{\beta}_k = 1) \equiv (\mathbf{a}_k \leq \mathbf{b}_k)$
13:    $S_1$, $S_2$: $[[\mathbf{U}_{k+1}]] \leftarrow \mathrm{OT}'\,([[\mathbf{a}_k]], [[\mathbf{b}_k]], \boldsymbol{\beta}_k \veebar \mathbf{1}, \mathrm{msk})$. $S_1$ receives $[[\mathbf{U}_{k+1}]]$
                                                        ▷ $\mathbf{U}_{k+1} \leftarrow \min(\mathbf{t}_k, \mathbf{h}_u)$
14:    $S_1, S_2$: Redo 11–13 to get $[[\mathbf{U}_{k+1}]] \leftarrow \max(\mathbf{U}_{k+1}, \mathbf{l}_u)$
15:    $S_1$: $[[\mathbf{z}_{k+1}]] \leftarrow (1 + \bar{\eta}) \otimes [[\mathbf{U}_{k+1}]] \oplus (-\bar{\eta}) \otimes [[\mathbf{U}_k]]$
16: end for
17: $S_1$: Pick $\boldsymbol{\rho}$ and send $[[(\mathbf{U}_K)_{1:m}]] \oplus [[\boldsymbol{\rho}]]$ to $S_2$
18: $S_2$: Decrypt, encrypt with $\mathrm{pk}_2$ and send to $S_1$: $[\{\mathbf{u} + \boldsymbol{\rho}\}]$
19: $S_1$: $[\{\mathbf{u}\}] \leftarrow [\{\mathbf{u} + \boldsymbol{\rho}\}] \oplus [\{-\boldsymbol{\rho}\}]$ and send it to $C$
20: $C$: Decrypt and output $\mathbf{u}$
```

Theorem 6.2.2. *Protocol 6.2.2 achieves privacy as in Definition 3.3.1, as long as the two semi-honest servers do not collude.*

The proof is given in Appendix E.1.

Remark 6.2.3. One can expand Protocols 6.2.1 and 6.2.2 over multiple time steps, such that $\mathbf{U}_0$ is obtained from the previous iteration and not given as input, and formally prove their privacy. The fact that the state will converge to a neighborhood of the origin is public knowledge, and is not revealed by the execution of the protocol.

Through communication, encryption and statistical blinding, the two servers can privately compute nonlinear operations. However, this causes an increase in the computation time due to the extra encryptions and decryptions and communication rounds, as will be pointed out in Section 6.2.5.

6.2.4 Fixed-point arithmetics MPC

The values that are encrypted or added to or multiplied with encrypted values have to be integers. We consider fixed-point representations with one sign bit, l_i integer bits and l_f fractional bits and multiply them by 2^{l_f} to obtain integers.

Working with fixed-point representations can lead to *overflow*, *quantization* and *arithmetic round-off* errors. Thus, we want to compute the deviation between the fixed-point solution and optimal solution of (6.2.4). The bounds on this deviation can be used in an offline step prior to the online computation to choose an appropriate fixed-point precision for the performance of the system.

Consider the following assumption:

Assumption 6.2.4. The number of fractional bits l_f and constant $c \geq 1$ are chosen large enough such that:

(i) $\bar{\mathcal{U}} \subseteq \mathcal{U}$: the fixed-point precision solution is still feasible.

(ii) The eigenvalues of the fixed-point representation $\bar{\mathbf{H}}_f$ are contained in the set $(0, 1]$, where $\mathbf{H}_f := \bar{\mathbf{H}}/(c\bar{L})$ and $\bar{L} := \lambda_{max}(\bar{\mathbf{H}})$. The constant c is required in order to overcome the possibility that $\overline{(1/\bar{L})\bar{\mathbf{H}}}$ has the maximum eigenvalue larger than 1 due to fixed-point arithmetic errors.

(iii) The fixed-point representation of the step size satisfies:

$$0 \leq \left(\sqrt{\kappa(\bar{\mathbf{H}})} - 1\right) \Big/ \left(\sqrt{\kappa(\bar{\mathbf{H}})} + 1\right) \leq \bar{\eta} < 1.$$

Item (i) ensures that the feasibility of the fixed-point precision solution is preserved, item (ii) ensures that the strong convexity of the fixed-point objective function still holds and item (iii) ensures that the fixed-point step size is such that the FGM converges.

Overflow errors: Bounds on the infinity-norm on the fixed-point dynamic quantities of interest in (6.2.4) were derived in [129] for each iteration k, and depend on a bounded set $\mathcal{X}_0$ such that $\mathbf{x}(t) \in \mathcal{X}_0$ and $\bar{\mathbf{x}}(t) \in \bar{\mathcal{X}}_0$:

$$
\begin{aligned}
\|\bar{\mathbf{U}}_{k+1}\|_\infty &\leq \max\{\|\bar{\mathbf{l}}_u\|_\infty, \|\bar{\mathbf{h}}_u\|_\infty\} = R_{\bar{\mathcal{U}}} \\
\|\bar{\mathbf{z}}_{k+1}\|_\infty &\leq (1+2\bar{\eta})\mathcal{R}_{\bar{\mathcal{U}}} =: \zeta, \\
\|\bar{\mathbf{t}}_k\|_\infty &\leq \|\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f\|_\infty \zeta + \|\bar{\mathbf{F}}_f\|_\infty \mathcal{R}_{\bar{\mathcal{X}}_0},
\end{aligned}
$$

where $\mathbf{F}_f = \bar{\mathbf{F}}/(c\bar{L})$ and $\mathcal{R}_{\mathcal{S}}$ represents the radius of a set $\mathcal{S}$ w.r.t. the infinity norm. We select from these bounds the number of integer bits l_i such that there is no overflow.

6.2.4.1 Difference between real and fixed-point solution

We want to determine a bound on the error between the fixed-point precision solution and the real solution of the MPC problem (6.2.3). The total error is composed of the error induced by having fixed-point coefficients and variables in the optimization problem, and by the round-off errors. Specifically, denote by $\mathbf{U}_K$ the solution in exact arithmetic of the MPC problem (6.2.3) obtained after K iterations of (6.2.4). Denote by $\tilde{\mathbf{U}}_K$ the solution obtained after K iterations but with $\mathbf{H}, \mathbf{F}, \mathbf{x}(t), \mathcal{U}, L, \eta$ replaced by their fixed-point representations. Finally, denote by $\bar{\mathbf{U}}_K$ the solution of Protocols 6.2.1 and 6.2.2 after K iterations, where the iterates $[[\mathbf{t}_k]], [[\mathbf{U}_k]]$ have fixed-point representation (truncations are performed). We obtain the following upper bound on the difference between the solution obtained on the encrypted data and the nominal solution of the implicit MPC problem (6.2.3) after K iterations:

$$
\|\bar{\mathbf{U}}_K - \mathbf{U}_K\|_2 \leq \|\tilde{\mathbf{U}}_K - \mathbf{U}_K\|_2 + \|\bar{\mathbf{U}}_K - \tilde{\mathbf{U}}_K\|_2.
$$

We present the derivation of the bounds of these accumulated errors in Appendix E.2.

Denote by ϵ_1 the bound on the quantization errors and by ϵ_2 the bound on the arithmetic round-off errors. As $l_f \to \infty$, $\epsilon_1 \to 0$ and $\epsilon_2 \to 0$. The persistent noise in (E.2.2) and (E.2.4), which is composed by quantization errors and round-off errors, becomes zero when the number of fractional bits mimics a real variable. This makes systems (E.2.2) and (E.2.4) input-to-state stable.

6.2.4.2 Choice of error level

The error bound $\epsilon := \epsilon_1 + \epsilon_2$ can be computed as a function of the number of fractional bits l_f. We can incorporate these errors as a bounded disturbance $\mathbf{d}(\cdot)$ in system (6.2.1): $\mathbf{d}(t) = \mathbf{B}\boldsymbol{\xi}(t)$, where $\boldsymbol{\xi}(t) = \mathbf{u}^*(t) - \bar{\mathbf{u}}(t)$. Then, we can design the terminal cost as described in [180], such that the controller achieves inherent robust stability to process perturbations $\|\mathbf{d}(t)\|_2 \leq \delta$, and choose: $\epsilon \leq \frac{\delta}{\|\mathbf{B}\|}$. Alternatively, we can incorporate the error in the suboptimality of the cost and assume that we obtain a cost $\bar{J}_N(\mathbf{x}(t), \bar{\mathbf{U}}_K) \leq J_N^*(\mathbf{x}(t)) + \epsilon'$, and compute ϵ such that asymptotic stability is achieved as in [164].

In the offline phase, the fixed-point precision of the variables is chosen such that there is no overflow and one of the conditions on ϵ is satisfied. Note that these conditions can be overly-conservative. This ensures that the MPC performance is guaranteed with a large margin, but the computation is slower because of the large number of bits. Once l_i and l_f have been chosen, we can pick N_σ such that the there is no overflow in Protocol 6.2.1 and Protocol 6.2.2, respectively. In Protocol 6.2.1, truncation cannot be done on encrypted data by the server, thus, the multiplications by 2^{l_f} are accumulated between $\mathbf{z}_k$ and $\mathbf{t}_k$, which means that $\mathbf{t}_k$ is multiplied by 2^{3l_f}. Hence, we choose N_σ such that $l_i + 3l_f + 2 < \log_2(N_\sigma/3)$ holds. For Protocol 6.2.2 we pick N_σ such that $l_i + 3l_f + 3 + \lambda_\sigma < \log_2(N_\sigma/3)$ holds, where λ_σ is the statistical security parameter. If the N_σ that satisfies these requirements is too large, one should use the simple gradient method, which will have multiplications up to 2^{2l_f}.

Remark 6.2.5. Instead of the FGM, we can use the simple gradient descent method, where fewer errors accumulate, and less memory is needed, but more iterations are necessary to reach to the optimal solution.

6.2.5 Numerical results and trade-off

The number of bits in the representation is crucial for the performance of the MPC scheme. At the same time, a more accurate representation slows down the private MPC computation.

In Figure 6.3, for a toy model of a spacecraft from [94] based on [124], we compare the predicted theoretical error bounds from Theorems E.2.1 and E.2.3 and the norm of the actual errors between the control input obtained with the client-server protocol and the control input of the unencrypted MPC. The simulation is run for a time step of MPC with 18 iterations, with three choices of the number of fractional bits: 16, 24 and 32 bits. The results are similar for the two-server protocol. We can observe that the predicted errors are around two orders of magnitude larger than the real errors caused by the encryption, and even for 16 bits of precision, the actual error is small.

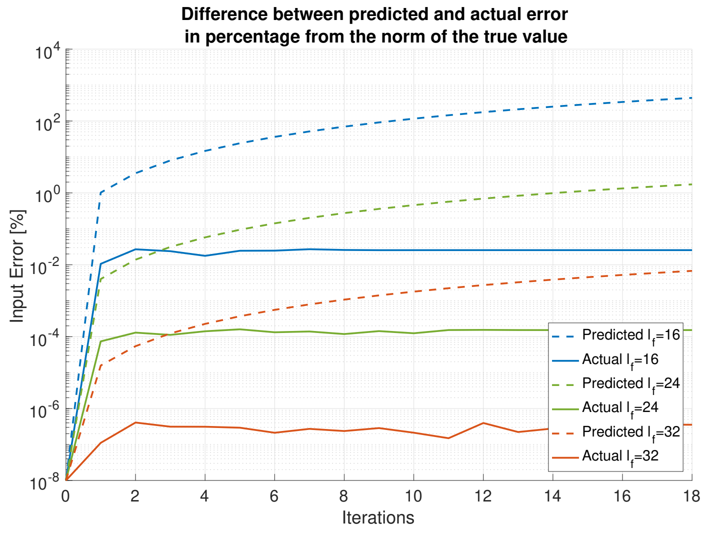


Figure 6.3: Predicted and actual errors for the control input computed in Protocol 6.2.1 for the Problem (6.2.3). The y-axis shows the errors (predicted and actual) as a percentage of the norm of the true iterate $\mathbf{U}_k$.

Furthermore, we implemented the above two protocols for various problem sizes, where n is the number of states and m is the number of control inputs, with a horizon fixed to 7. Table 6.1 shows the computation times for Protocols 6.2.1 and 6.2.2 for $l_i = 16$ integer bits and $l_f \in \{16, 32\}$ fractional bits. We fix the number of iterations to 50. The times obtained vary linearly with the number of iterations, so one can approximate how long it would take to run a different number of iterations. The size of the N_σ for the encryption is chosen to be 512 bits. The computations were effectuated on a 2.2 GHz Intel Core i7.

The larger the number of fractional bits, the larger is the execution time, because the computations involve larger numbers and more bits have to be sent from one party to another in the case of encrypted comparisons. Communication is the reason for the significant slow-down observed in the two-server architecture compared to the client-server architecture. Performing the projection requires l communication rounds, where l is the number of bits of the messages compared. Privately updating the iterates requires another communication round. However, the assumption is that the servers are powerful computers, and the execution time can be greatly improved using parallelization.

l_f	$n = 2$ $m = 2$	$n = 5$ $m = 5$	$n = 10$ $m = 10$	$n = 20$ $m = 20$	$n = 50$ $m = 30$
CS 16	1.21	4.08	13.38	39.56	84.28
CS 32	1.33	5.20	15.46	53.48	105.84
SS 16	23.27	59.81	123.19	261.75	457.87
SS 32	31.21	91.74	170.62	372.42	579.38

Table 6.1: Average execution time in seconds for Protocols 6.2.1, 6.2.2.

To summarize, privacy comes at the price of complexity: hiding the private data requires working with large encrypted numbers, and the computations on private data require communication. Furthermore, the more parties that need to be oblivious to the private data (two servers in the second architecture compared to one server in the first architecture), the more complex the private protocols become.

6.3 Secure Evaluation of MPC with private parameters

In this section, we change the problem setting from Section 6.2. The system in (6.2.1) is considered to be partitioned in subsytems for $i \in [M]$, $\mathbf{x}^i(t) \in \mathbb{R}^{n_i}$, $\sum_{i=1}^{M} n_i = n$ and $\mathbf{u}^i(t) \in \mathbb{R}^{m_i}$, $\sum_{i=1}^{M} m_i = m$, such that $\mathbf{x}(t) = \begin{bmatrix}\mathbf{x}^1(t)^\intercal & \dots & \mathbf{x}^M(t)^\intercal\end{bmatrix}$, $\mathbf{u}(t) = \begin{bmatrix}\mathbf{u}^1(t)^\intercal & \dots & \mathbf{u}^M(t)^\intercal\end{bmatrix}$.

The privacy-absent cloud-MPC problem is depicted in Figure 6.4. The system (6.2.1) is composed of M subsystems, that can be thought of as different agents, which measure their states and receive control actions, and of a setup entity which holds the system's model and parameters. The control decision problem is solved at the cloud level, by a cloud controller, which receives the system's model and parameters, the measurements, as well as the constraint sets imposed by each subsystem. The control inputs are then applied by one virtual actuator. Examples of such architecture include a smart building temperature control application, where the subsystems are apartments and the actuator is a machine in the basement, or the subsystems are robots in a swarm coordination application and the actuator is a ground control that sends them waypoints.

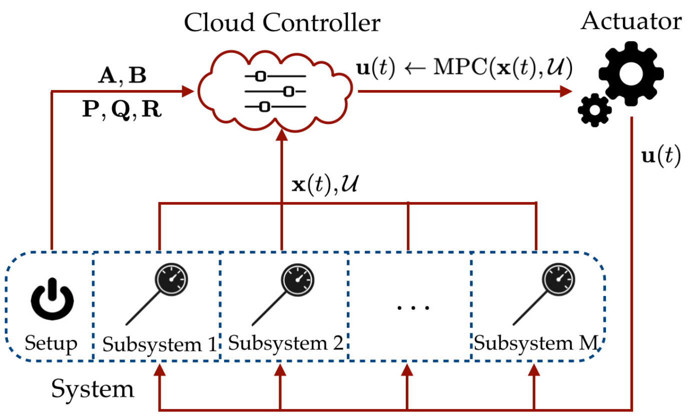

Cloud Controller
Actuator
$\mathbf{A}, \mathbf{B}$
$\mathbf{P}, \mathbf{Q}, \mathbf{R}$
$\mathbf{u}(t) \leftarrow \mathrm{MPC}(\mathbf{x}(t), \mathcal{U})$
$\mathbf{u}(t)$
$\mathbf{x}(t), \mathcal{U}$
Setup
Subsystem 1
Subsystem 2
...
Subsystem M
System

Figure 6.4: Cloud-based MPC problem for a system composed of a number of subsystems that measure their states and a setup entity which holds the system's model and parameters. The control action is computed by a cloud controller and sent to a virtual actuator.

6.3.1 Privacy objectives and overview of solution

The system model and MPC costs $\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}$ are known only to the system, but not to the cloud and actuator, hence, the matrices $\mathbf{H}, \mathbf{F}$ in (6.2.3) are also private. The measurements and constraints are not known to parties other than the corresponding subsystem and should remain private, such that the sensitive information of the subsystems is concealed. The control inputs should not be known by the cloud.

The goal of this work is to design a private cloud-outsourced version of the fast gradient method in (6.2.4) for the model predictive control problem, such that the actuator obtains the control action $\mathbf{u}^*(t)$ for system (6.2.1), without learning anything else in the process and with only a minimum amount of computation. At the same time, the cloud controller should not learn anything other than what was known prior to the computation about the measurements $\mathbf{x}(t)$, the control inputs $\mathbf{u}^*(t)$, the constraints $\mathcal{U}$, and the system model $\mathbf{H}, \mathbf{F}$. We consider that all parties are semi-honest, but they are allowed to collude. Specifically, coalitions between $M-1$ subsystems and the controller or between $M-1$ subsystems and the actuators are allowed. Then, we require the protocol that implements the above satisfy multi-party privacy, as defined in Definition 2.2.6.

As a primer to our private solution to the MPC problem, we briefly mention here the cryptographic tools used to achieve privacy. We will encrypt the data with a *labeled homomorphic encryption* scheme, which allows us to evaluate an unlimited number of additions and one multiplication over encrypted data, denoted as $\hat{\mathrm{E}}(\cdot)$. The labeled homomorphic encryption builds on top of an *additively homomorphic encryption* scheme, which allows only the evaluation of additions over encrypted data, denote as $[[\cdot]]$ or $\mathrm{E}(\cdot)$, a *secret sharing* scheme, which enables the splitting of a message into two random shares, that cannot be used individually to retrieve the message, and a *pseudorandom generator* that, given a key and a small seed, called *label*, outputs a larger sequence of bits that is indistinguishable from random. The right choice of labels is essential for a seamless application of labeled homomorphic encryption on dynamical data, and we choose the labels to be the time steps at which the data is generated. These tools ensure that we can evaluate polynomials on the private data. Furthermore, the computations for determining the control action also involve projections on a feasible hyperbox. To achieve this in a private way, we make use of *two-party private comparison* that involves exchanges of encrypted bits between two parties, and *oblivious transfer*, that allows us to choose a value out of many values when the index is secret. These cryptographic tools are be described in detail in Chapter 2, and our private cloud-based MPC solution that incorporates them will be presented in Section 6.3.2.

6.3.2 MPC with encrypted model and encrypted signals

Notice that the setting is similar to the secure evaluation of LQG, presented in Chapter 5.1, but for a non-linear control policy, given by MPC, which require extra computations. To this end, we propose a solution that uses Labeled Homomorphic Encryption to achieve encrypted multiplications and the private execution of MPC on encrypted data. LabHE has the useful property called unbalanced efficiency and we will employ this property by having the cloud perform the more complex operations and the actuator the more efficient ones.

Our protocols consist of three phases: the offline phase, in which the computations that are independent from the specific data of the users are performed, the initialization phase, in which the computations related to the constant parameters in the problem are performed, and the online phase, in which computations on the variables of the problem are performed. The initialization phase can be offline, if the constant parameters are a priori known, or online otherwise.

Figure 6.5 represents the private version of the MPC diagram from Figure 6.4, where the quantities are briefly described next and in more detail in Section 6.3.2.1. The actuator holds the MPC query functionality, denoted by f_{MPC}. Offline, the actuator generates a pair of master keys, as described in Preamble 2.5.2 and publishes the master public key. The setup and subsystems generate their secret keys and send the corresponding public keys to the actuator. Still offline, these parties generate the labels corresponding to their data with respect to the time stamp and the size of the data, similarly to the solution in Chapter 5.1. This is shown in Protocol 6.3.1.

The setup entity sends the LabHE encryptions of the state matrices and costs to the cloud controller before the execution begins. The subsystems send the encryptions of the input constraints to the cloud controller, also before the execution begins. Online, at every time step, the subsystems encrypt their measurements and send them to the cloud. After the cloud performs the encrypted MPC query for one time step, it sends the encrypted control input at the current time step to the actuator, which decrypts it and inputs it to the system. In Protocol 6.3.2, we describe how the encrypted MPC query is performed by the parties, which involves the actuator sending an encryption of the processed result back to the cloud such that the computation can continue in the future time steps.

We now show how to transform the FGM in Equation (6.2.4) into a private version, using Labeled Homomorphic Encryption and private two-party comparison. The message space $\mathcal{M}$ of the encryption schemes we use is $\mathbb{Z}_N$, the additive group of integers modulo a large value N. This means that, prior to encryption, the values have to be represented as integers. Assume that this preprocessing step has been already performed; see Remark 6.3.3.

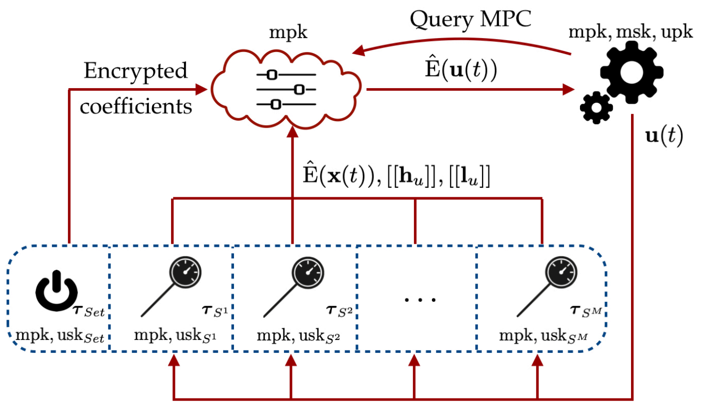


Figure 6.5: The setup and subsystems send their encrypted data to the cloud. The cloud has to run the MPC algorithm on the private measurements and the system's private matrices and send the encrypted result to the actuator. The latter then actuates the system with the decrypted inputs.

First, let us write $\mathbf{t}_k$ in (6.2.4a) as a function of $\mathbf{U}_k$ and $\mathbf{U}_{k-1}$:

$$\mathbf{t}_k = \left(\mathbf{I}_{Mm} - \frac{1}{L}\mathbf{H}\right) z_k - \frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t) = \left(\mathbf{I}_{Mm} - \frac{1}{L}\mathbf{H}\right)[(1+\eta)\mathbf{U}_k - \eta\mathbf{U}_{k-1}] - \frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t)$$
$$= \mathbf{U}_k + \eta(\mathbf{U}_k - \mathbf{U}_{k-1}) - \frac{1}{L}\mathbf{H}\mathbf{U}_k - \frac{\eta}{L}\mathbf{H}(\mathbf{U}_k - \mathbf{U}_{k-1}) - \frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t).$$

If we consider the composite variables $\frac{1}{L}\mathbf{H}$, $\frac{\eta}{L}\mathbf{H}$, $\frac{1}{L}\mathbf{F}$ and variables $\mathbf{U}_k, \mathbf{U}_{k-1}, \mathbf{x}(t)$, then $\mathbf{t}_k$ can be written as a degree-two multivariate polynomial. This allows us to compute $[[\mathbf{t}_k]]$ using LabHE. Then, one encrypted iteration of the FGM, where we assume that the cloud has access to $\hat{\mathrm{E}}\left(-\frac{1}{L}\mathbf{H}\right)$, $\hat{\mathrm{E}}\left(-\frac{\eta}{L}\mathbf{H}\right)$, $\hat{\mathrm{E}}\left(\frac{1}{L}\mathbf{F}^\intercal\right)$, $\hat{\mathrm{E}}(\mathbf{x}(t))$, $[[\mathbf{h}_u]]$, $[[\mathbf{l}_u]]$ can be written as follows. Denote the computation on the inputs mentioned previously as f_{iter}. We use both $\hat{\mathrm{Add}}$ and $\hat{\oplus}, \hat{\ominus}$, $\hat{\mathrm{Mlt}}$ and $\hat{\otimes}$ for a better visual representation.

$$[[\mathbf{t}_k - \boldsymbol{\rho}_k]] = \hat{\mathrm{Mlt}}\left(\hat{\mathrm{E}}\left(\frac{-1}{L}\mathbf{F}^\intercal\right), \hat{\mathrm{E}}(\mathbf{x}(t))\right) \hat{\oplus} \hat{\mathrm{Mlt}}\left(\hat{\mathrm{Add}}\left(\mathbf{I}_{Mm}, \hat{\mathrm{E}}\left(\frac{1}{L}\mathbf{H}\right)\right), \hat{\mathrm{E}}(\mathbf{U}_k)\right) \hat{\oplus}$$
$$\hat{\oplus} \hat{\mathrm{Mlt}}\left(\hat{\mathrm{Add}}\left(\hat{\mathrm{E}}(\eta) \otimes \mathbf{I}_{Mm}, \hat{\mathrm{E}}\left(\frac{-\eta}{L}\mathbf{H}\right)\right), (\hat{\mathrm{E}}(\mathbf{U}_k) \hat{\ominus} \hat{\mathrm{E}}(\mathbf{U}_{k-1}))\right), \tag{6.3.1}$$

where $\boldsymbol{\rho}_k$ is the secret obtained by applying f_{iter} on the LabHE secrets of the inputs of f_{iter}. When the actuator applies the LabHE decryption primitive on $[[\mathbf{t}_k - \boldsymbol{\rho}_k]]$, $\boldsymbol{\rho}_k$ is removed. Hence, for simplicity, we will write $[[\mathbf{t}_k]]$ instead of $[[\mathbf{t}_k - \boldsymbol{\rho}_k]]$.

Second, for (6.2.4b), we have to perform the projection of $\mathbf{t}_k$ over the feasible domain described by $\mathbf{h}_u$ and $\mathbf{l}_u$, where all the variables are encrypted, as well as the private update of $\mathbf{U}_{k+1}$ with the projected iterate. Since the comparison needs to be done between AHE ciphertexts and not LabHE ciphertexts, the solution for this is very similar to the solution in the first part of the chapter, given in Section 6.2.3.

Finally, because the next iteration can proceed only if the cloud has access to the full LabHE encryption $\hat{\mathrm{E}}(\mathbf{U}_{k+1})$, instead of $[[\mathbf{U}_{k+1}]]$, the cloud and actuator have to refresh the encryption. Specifically, the cloud secret-shares $[[\mathbf{U}_{k+1}]]$ in $[[\mathbf{U}_{k+1} - \mathbf{r}_{k+1}]]$ and $\mathbf{r}_{k+1}$, and sends $[[\mathbf{U}_{k+1} - \mathbf{r}_{k+1}]]$ to the actuator. The actuator decrypts it, and, using a previously generated secret, sends back $\hat{\mathrm{E}}(\mathbf{U}_{k+1} - \mathbf{r}_{k+1}) = \left(\mathbf{U}_{k+1} - \mathbf{r}_{k+1}, \left[\left[\mathbf{b}^U_{k+1}\right]\right]\right)$. Then, the cloud recovers the LabHE encryption as $\hat{\mathrm{E}}(\mathbf{U}_{k+1}) = \hat{\mathrm{Add}}(\hat{\mathrm{E}}(\mathbf{U}_{k+1} - \mathbf{r}_{k+1}), \mathbf{r}_{k+1})$. In what follows, we will outline the private protocols obtained by integrating the above observations.

6.3.2.1 Private protocol

Assume that K, N are fixed and known by all parties. Subscript S^i stands for the i-th subsystem, for $i \in [M]$, subscript Set for the Setup, subscript A for the actuator and subscript C for the cloud.

Protocol 6.3.1: Initialization of encrypted MPC

Input: Actuator: f_{MPC}; Subsystems: $\mathcal{U}^i$; Setup: $\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}$.

Output: Subsytems: $\mathrm{mpk}, \mathrm{upk}_{S^i}, \mathrm{usk}_{S^i}, \boldsymbol{\tau}_{S^i}, \mathbf{b}_{S^i}, [[\mathbf{b}_{S^i}]], \mathrm{R}_{S^i}$; Setup: $\mathbf{H}, \mathbf{F}, \eta, L$, mpk, upk_{Set}, usk_{Set}, $\boldsymbol{\tau}_{Set}, \mathbf{b}_{Set}, [[\mathbf{b}_{Set}]], \mathrm{R}_{Set}$; Actuator: $\mathrm{usk}, \mathrm{upk}, \boldsymbol{\tau}_A, \mathbf{b}_A, [[\mathbf{b}_A]], \mathrm{R}_A$; Cloud: $\mathrm{mpk}, \hat{\mathrm{E}}\left(-\frac{1}{L}\mathbf{H}\right)$, $\hat{\mathrm{E}}\left(-\frac{\eta}{L}\mathbf{H}\right), \hat{\mathrm{E}}\left(\frac{1}{L}\mathbf{F}^\intercal\right)$, $\hat{\mathrm{E}}(\eta)$, $[[\mathbf{h}_u]]$, $[[\mathbf{l}_u]]$, R_C.

Offline:

1: Actuator: Generate $(\mathrm{mpk}, \mathrm{msk}) \leftarrow \mathrm{Init}(1^\sigma)$ and distribute mpk to the others. Also generate a key usk for itself.

2: Subsystems, Setup: Each get $(\mathrm{usk}, \mathrm{upk}) \leftarrow \mathrm{KeyGen}(\mathrm{mpk})$ and send upk to the actuator.

3: Subsystems, Setup, Actuator: Allocate labels to the inputs of function f_{MPC}: $\tau_1, \ldots, \tau_v$, where v is the total number of inputs, as follows:

Subsystem i: for $\mathbf{x}^i(t)$ of size n^i, where i denotes a subsystem, generate the corresponding labels

```
   $\tau_{\mathbf{x}^i(t)} = [0 \quad 1 \quad n^i \ldots \quad n^i - 1]^\intercal$.
   Setup: for matrix $\mathbf{H} \in \mathbb{R}^{Mm \times Mm}$, set $l = 0$, generate $\tau_{\mathbf{H}}$ and update $l = M^2 m^2$, then follow the same steps for $\mathbf{F}$, starting from $l$ and updating it.
   Actuator: follow the same steps as the subsystems and setup, and then generate similar labels for the iterates $\mathbf{U}_k$ starting from the last $l$, for $k = 0, \ldots, K-1$.
4: Subsystems, Setup, Actuator, Cloud: Generate randomness for blinding and encryptions R.
5: Subsystems, Setup, Actuator: Perform the offline part of the LabHE encryption primitive. The actuator also performs the offline part for the decryption. The parties thus obtain $\mathbf{b}, [[\mathbf{b}]]$.
6: Actuator: Generate initializations for the initial iterate $\mathbf{U}_0'$.
7: Actuator: Form the program $\mathcal{P} = (f_{\mathrm{MPC}}, \tau_1, \ldots, \tau_v)$.
   Initialization:
8: Setup: Compute $\mathbf{H}$ and $\mathbf{F}$ from $\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}$ and then $L = \lambda_{\max}(\mathbf{H})$ and $\eta = (\sqrt{\kappa(\mathbf{H})} - 1)/(\sqrt{\kappa(\mathbf{H})}+1)$. Perform the online part of LabHE encryption and send to the cloud: $\hat{\mathrm{E}}\left(-\frac{1}{L}\mathbf{H}\right)$, $\hat{\mathrm{E}}\left(-\frac{\eta}{L}\mathbf{H}\right)$, $\hat{\mathrm{E}}\left(\frac{1}{L}\mathbf{F}^\intercal\right)$, $\hat{\mathrm{E}}(\eta)$.
9: Subsystems: Perform the online part of LabHE encryption and send to the cloud, which aggregates what it receives into: $[[\mathbf{h}_u]], [[\mathbf{l}_u]]$.
```

Protocol 6.3.2: Encrypted MPC step

```
Input: Actuator: $f_{\mathrm{MPC}}$; Subsystems: $\mathbf{x}^i(t), \mathcal{U}^i$; Setup: $\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}$.
Output: Actuator: $\mathbf{u}(t)$
   Offline + Initialization:
1: Subsystems, Setup, Cloud, Actuator: Run Protocol 6.3.1.
   Online:
2: Cloud: $\left[\left[\frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t)\right]\right] \leftarrow \hat{\mathrm{Mlt}}\left(\hat{\mathrm{E}}\left(\frac{1}{L}\mathbf{F}^\intercal\right), \hat{\mathrm{E}}(\mathbf{x}(t))\right)$.
3: Actuator: Send the initial iterate to the cloud: $\hat{\mathrm{E}}(\mathbf{U}_0')$.
4: Cloud: Change the initial iterate: $\hat{\mathrm{E}}(\mathbf{U}_0) = \hat{\mathrm{Add}}\left(\hat{\mathrm{E}}\left(\mathbf{U}_0'\right), \mathbf{r}_0\right)$.
5: Cloud: $\hat{\mathrm{E}}\left(\mathbf{U}_{-1}\right) \leftarrow \hat{\mathrm{E}}\left(\mathbf{U}_0\right)$.
6: for $k = 0, \ldots, K-1$ do
7:     Cloud: Compute $[[\mathbf{t}_k]]$ as in Equation (6.3.1).
8:     Cloud: $([[\mathbf{a}_k]], [[\mathbf{b}_k]]) \leftarrow \mathrm{randomize}\left([[\mathbf{h}_u]], [[\mathbf{t}_k]]\right)$.
```

```
 9:     Cloud, Actuator: Execute comparison protocol 2.7.2; Actuator obtains $\boldsymbol{\delta}_k$.
10:     Cloud, Actuator: $[[\mathbf{U}_{k+1}]] \leftarrow \mathrm{OT}'\left([[\mathbf{a}_k]], [[\mathbf{b}_k]], \boldsymbol{\delta}_k, \mathrm{msk}\right)$.
11:     Cloud: $([[\mathbf{a}_k]], [[\mathbf{b}_k]]) \leftarrow \mathrm{randomize}\left([[\mathbf{l}_u]], [[\mathbf{U}_{k+1}]]\right)$.
12:     Cloud, Actuator: Execute comparison protocol 2.7.2; Actuator obtains $\boldsymbol{\delta}_k$.
13:     if k!=K-1 then
14:         Cloud, Actuator: $[[\mathbf{U}_{k+1}]] \leftarrow \mathrm{OT}'\left([[\mathbf{a}_k]], [[\mathbf{b}_k]], \boldsymbol{\delta}_k \veebar \mathbf{1}, \mathrm{msk}\right)$. Cloud receives $[[\mathbf{U}_{k+1}]]$.
15:         Cloud: Send to the actuator $\left[\left[\mathbf{U}'_{k+1}\right]\right] \leftarrow \mathrm{Add}\left([[\mathbf{U}_{k+1}]], [[-\mathbf{r}_k]]\right)$, where $\mathbf{r}_k$ is
            randomly selected from $\mathcal{M}^{Mm}$.
16:         Actuator: Decrypt $\left[\left[\mathbf{U}'_{k+1}\right]\right]$ and send back $\hat{\mathrm{E}}(\mathbf{U}'_{k+1})$.
17:         Cloud: $\hat{\mathrm{E}}\left(\mathbf{U}_{k+1}\right) \leftarrow \hat{\mathrm{Add}}\left(\hat{\mathrm{E}}\left(\mathbf{U}'_{k+1}\right), \mathbf{r}_k\right)$.
18:     else
19:         Cloud, Actuator: $\mathbf{u}(t) \leftarrow \mathrm{OT}\left([[\mathbf{a}_k]]_{1:m}, [[\mathbf{b}_k]]_{1:m}, \{\boldsymbol{\delta}_k\}_{1:m} \veebar \mathbf{1}, \mathrm{msk}\right)$. Actuator
            receives $\mathbf{u}(t)$.
20:     end if
21: end for
22: Actuator: Input $\mathbf{u}(t)$ to the system.
```

Lines 3 and 4 ensure that neither the cloud nor the actuator knows the initial point of the optimization problem.

Assumption 6.3.1. An adversary cannot corrupt at the same time both the cloud controller and the virtual actuator or more than $M-1$ subsystems.

Theorem 6.3.2. *Under Assumption 6.3.1, the encrypted MPC solution presented in Protocol 6.3.2 achieves multi-party privacy, cf. Definition 2.2.6.*

The proof is given in Appendix E.3.

We can also have the private MPC scheme run for multiple time steps. Protocol 6.3.1 can be modified to also generate the labels and secrets for T time steps. Protocol 6.3.2 can be run for multiple time steps, and warm starts can be included by adding two lines such that the cloud obtains $\hat{\mathrm{E}}\Big(\left\{\mathbf{U}_K^t\right\}_{m+1:M}\Big)$ and sets $\hat{\mathrm{E}}(\mathbf{U}_0^{t+1}) = \left[\mathbf{U}_K^{t\,\intercal} \quad \mathbf{0}_m^\intercal\right]^\intercal$.

Remark 6.3.3 (Analysis of errors). The same discussion related to the errors arising from using a fixed-point representation as in Section 6.2.4 applies here. The bounds on these deviatiosn can be used in an offline step to choose an appropriate fixed-point precision for the desired performance of the system.

6.3.3 Discussion on efficiency

Secure multi-party computation protocols require many rounds of communication in general, and the amount of communication depends on the amount of data that needs to be concealed. This can also be observed in Protocol 6.3.2. Therefore, in order to be able to use this protocol, we need fast and reliable communication between the cloud and the actuator.

In the architecture we considered, the subsystems are computationally and memory constrained devices, hence, they are only required to generate the encryptions for their measurements. The setup only holds constant data, and it only has to compute the matrices in (6.2.4) and encrypt them in the initialization step. Furthermore, we considered the existence of the setup entity for clarity in the exposition of the protocols, but the data held by the setup could be distributed to the other participants in the computation. In this case, the cloud would have to perform some extra steps in order to aggregate the encrypted system parameters (see [6] for a related solution). Notice that the subsystems and setup do not need to generate labels for the number of iterations, only the actuator does. The actuator is supposed to be a machine with enough memory to store the labels and reasonable amount of computation power such that the encryptions and decryptions are performed in a practical amount of time (that will be dependent on the sampling time of the system), but less powerful than the cloud. The cloud controller is assumed to be a server, which has enough computational power and memory to be capable to deal with the computations on the ciphertexts, which can be large, depending on the encryption schemes employed.

If fast and reliable communication is not available or if the actuator is a highly constrained device, then a fully homomorphic encryption solution that is solely executed by the cloud might be more appropriate, although its execution can be substantially slower.

Compared to the two-server MPC with private signals but public model from Section 6.2, where only AHE is required, the MPC with private signals and private model we considered in this chapter is only negligibly more expensive. Specifically, the ciphertexts are augmented with one secret that has the number of bits substantially smaller than the number of bits in an AHE ciphertext, and each online iteration only incurs one extra round of communication, one decryption and one encryption. All the other computations regarding the secrets are done offline. This shows the efficiency and suitability of LabHE for encrypted control applications.

Chapter 7

Data-driven control

In this chapter, we investigate a Control as a Service scenario, where a client employs a specialized outsourced control solution from a service provider. Control as a Service (CaaS) is becoming a reality–particularly in the case of building automation and smart grid management. Often, the control algorithms in CaaS focus on controlling the client's system directly from input-output data; the privacy-sensitive model parameters of the client's system are either not available or variable. Therefore, large quantities of data collected from the client need to be uploaded to a cloud server. This data can be used by a malevolent cloud service provider to infer sensitive information about the client and mount attacks. Hence, we require the service provider to perform data-driven control in a privacy-preserving manner on the input-output data samples from the client. To this end, we co-design the control scheme with respect to both *control performance* and *privacy specifications*. First, we formulate our control algorithm based on recent results from the behavioral framework, and we prove closeness between the classical formulation and our formulation that accounts for noise and precision errors arising from encryption. Second, we use a state-of-the-art leveled homomorphic encryption scheme to enable the service provider to perform high complexity computations on the client's encrypted data, ensuring privacy. Finally, we streamline our solution by exploiting the rich structure of data, and meticulously employing ciphertext batching and rearranging operations to enable parallelization.

The first part of the chapter studies the data-based predictive control under an ℓ_2-regularization, while the second part of the chapter considers an ℓ_1-regularization. This slight difference leads very different solutions.

This chapter covers the work presented in [14–16].

7.1 Introduction

As cloud services become pervasive, inspiring the term "Everything as a Service" [23], privacy of the underlying data and algorithms becomes indispensable. Distributed systems coordination and process control enterprises can benefit from such services by commissioning cloud servers to aggregate and manipulate the increasing amounts of data or implement reference governors. For example, in the context of smart building automation, Control as a Service (CaaS) businesses offer high-performance algorithms to optimize a desired cost, while achieving the required control goals. Clients like hospitals, factories, commercial and residential buildings might prefer such specialized outsourced solutions rather than investing in local control solutions. However, the data that is outsourced, stored and processed at a CaaS provider's cloud server consists of privacy-sensitive measurements such as user patterns, energy and temperature measurements, occupancy information etc. Nowadays, data can be monetized and used to disrupt the normal functionality of a system, leading to frequent database leakages and cyberattacks. This compels researchers and practitioners to design decision algorithms that are optimized for both efficiency and privacy. In this chapter, we provide prototypes for such private cloud-based control algorithms.

In a client-server context, homomorphic encryption provides an appealing tool that allows one server to evaluate a functionality over the encrypted data of the client without the need to decrypt the data locally, at independently specified precision and security levels. As discussed in [215], private single client computing can be enabled using homomorphic encryption schemes. Moreover, this technology has matured enough for it to be deployed in practice, especially in the healthcare [127] and financial [161] sectors. Since its genesis in [102], fully homomorphic encryption (FHE) has been developed substantially, in terms of more theoretically efficient leveled constructions [50, 62, 103], bootstrapping methods [60, 63], computational and hardware optimizations. There are many libraries that implement various FHE schemes and capabilities [66, 115, 166, 178] and a tremendous number of papers that build on them.

Related work

Encrypted control, surveyed in [200], has been recently gaining momentum, due to the strong privacy guarantees it offers even when the controller is located on an untrusted platform. These works can be classified with respect to the type of control algorithms they implement and the cryptographic tools they use. Linear control algorithms with public model and gains are considered in most works and are implemented using partially homomorphic encryption (PHE) schemes. Nonlinear control algorithms with public model are considered in [11, 198], where PHE is combined either with secure multiparty computation schemes or the client partakes in the computation. In [6, 134], somewhat or fully homomorphic encryption schemes are preferred, either to guarantee more privacy for the system parameters (that are still known at the client) or to achieve more complex control algorithms. The work in [204] is based on data-driven techniques and uses leveled homomorphic encryption. There, the cloud computes the value function in a reinforcement learning task over multiple time steps in a one-shot way, while we collect encrypted data from the client at every time step and compute on it iteratively.

In the case of controlling a system with known model and linear controller parameters, the line of work [65, 135] has shown how to perform the computations at subsequent time steps without the need of bootstrapping or ciphertext reset. In contrast, we deal with both unknown model matrices and nonlinearities, which prevent the application of their methods.

As mentioned before, most private cloud-based control schemes assume that the model of the system is known to the client or public. However, the model might not be always available and has to be computed from data. Identification and data-driven control of unknown systems have been extensively studied in classical and recent literature [25, 35, 70, 82, 93, 154, 158, 196, 216, 226]. These methods are usually designed with the objective of sample-efficiency and control performance, without considering a private implementation. For example, the standard system identification-certainty equivalence control architecture might require solving non-convex problems [232] or might involve Singular Value Decomposition (SVD) [216], which are prohibitive from an encrypted evaluation perspective. On the other hand, the behavioral framework [35, 36, 70, 72, 82, 83, 158, 219, 226], which has received renewed interest recently, is more compatible with modern encryption tools and allows us to use less costly encrypted operations. The idea of the behavioral framework is that the state representation can be replaced by a data-based representation which only uses the trajectories of the system, bypassing the need for system identification.

Contributions

Our goal is to perform online data-based predictive control on encrypted data, while maintaining the privacy of the client's uploaded input-output data, desired setpoint and control actions. In the context of private implementation, we need to depart from the typical techniques used in the non-private versions and make compromises between tracking performance and privacy. Our controller is based on the behavioral framework and the control performance is captured by the LQR cost.

In the first part of the chapter, we consider two versions and adapt them to account for an encrypted implementation: i) an offline version, where precollected data from the system is encrypted and handed to the cloud to compute an offline feedback control law, and ii) an online version, where the cloud server computes the control at every time step based on both the offline precollected data and on the encrypted measurements received from the client. We employ a new leveled homomorphic encryption scheme optimized for efficiency and precision as the powerful base tool for encrypting and evaluating the private data on the untrusted cloud machine. To account for noisy data and changes incurred by encryption, we employ ℓ_2-regularization. At the same time, we rewrite the computation of the control input as a low-depth arithmetic circuit by using mathematical tools such as the Schur complement to facilitate matrix inversions, and carefully batching the private values into ciphertexts. Specifically, our contributions are the following:

- Present the first solution for encryption-aware control in the case of an unknown linear system model.
- Formulate a setpoint tracking problem that learns the control action from the available data at every time step and is amenable to encrypted implementation.
- Prove the closeness between this formulation and a data-driven LQR problem.
- Extend this approximation to allow the online collection and incorporation of samples to improve robustness and adaptation to new data and to errors arising from the use of encryption.
- Design storage and computation efficient encrypted versions of the offline and online data-driven problems, by carefully encoding the values to facilitate parallelization and re-engineering the way the operations are performed.
- Implement the algorithms for a standard security parameter and realistic client and server machines. Present the results for a temperature control problem and showcase the storage and complexity improvements compared to a more intuitive but less efficient implementation.

We note that the work presented in the first part of the chapter enhances and improves [15] in the following aspects: we propose a new encrypted solution that reduces more than twofold the time and the memory requirements of the previous solution, allowing us to simulate more realistic and more secure scenarios; we prove the close relationship between the standard data-driven LQR and our reformulation; and we present many more extensions.

In the second part of the chapter, we investigate a sparse data-predictive control problem, run at a cloud service to control a system with unknown model, using ℓ_1-regularization to limit the behavior complexity. Specifically, our goal is to control an unknown system using only the privacy-sensitive input-output data that are potentially noisy. In the case of noisy data, inspired by [70, 83], we reformulate the data-based predictive control as a Lasso problem. We propose:

- a distributed encrypted solution for ℓ_1-regularized data predictive control, using an optimized implementation;
- a non-homogeneous splitting of the data for better convergence;
- a customized distributed encrypted solution for this non-homogeneous splitting to a non-homogeneous set of servers: one powerful server and a few less powerful devices, added for security reasons.

We emphasize that the techniques for performing the encrypted computations efficiently are one of main contributions of this chapter. These tools can easily be generalized and used in implementing other related problems in an encryption-friendly way, such as learning and adaptive control algorithms, recursive least squares, sequential update of inverses etc.

Special notation for this chapter

For a vector $\mathbf{x} \in \mathbb{R}^n$, we denote by $\mathbf{x}_{[i]}$ the i'th element of the vector. The vector $\mathbf{e}_i$ is the vector of all zeros, except for position $i-1$ where it has 1.

We will use $+$ and $\odot$ for Single Instruction Multiple Data (SIMD, see Preamble 2.5.3) addition and multiplication between vectors and $\rho(\mathbf{x}, i)$ to denote the row vector $\mathbf{x}$ rotated to the left by i positions ($i < 0$ means rotation to the right).

We denote by $\mathcal{E}_{\text{v0}}(\mathbf{x})$ the encryption of the vector $\mathbf{x}$ followed by trailing zeros in one ciphertext and by $\mathcal{E}_{\text{v}*}(\mathbf{x})$ the encryption of the vector $\mathbf{x}$ followed by junk elements (elements whose value we do not care about, usually obtained through partial operations). We denote by $\mathcal{E}_{\text{vv}}(\mathbf{x})$ the encryption of the vector $\mathbf{x}$ repeated $[\mathbf{x}\,\mathbf{x}\ \ldots]$ in one ciphertext. Finally, we also define the encryption of a vector repeated element-wise: for $\mathbf{x} = \left[x_{[1]}\, x_{[2]}\ \ldots\ x_{[n]}\right]$, this encryption is $\mathcal{E}_{\text{vr0}}(\mathbf{x}) = \left[x_{[1]}\ \ldots\ x_{[1]}\, x_{[2]}\ \ldots\ x_{[2]}\ \ldots\ x_{[n]}\ \ldots\ x_{[n]}\, 0\, 0\ \ldots\right]$, where each element is repeated for a specific number of times. Similarly, we use the notation $\mathcal{E}_{\text{vr}*}(\cdot)$ when the repeated elements are followed by junk elements.

7.2 Data-driven LQR on encrypted data

Organization. We formulate the data-based predictive control problem and state the goals for a private solution in Section 7.2.1. In Section 7.2.2, we present the reformulations for an offline feedback and an offline feedback solutions, taking into account the encryption characteristics, and we prove that the solution to reformulation is close to the solution to the original problem. In Section 7.2.2.4 we describe the main challenges and corresponding solutions. Then, we outline the private protocol for the offline feedback control in Section 7.2.3 and for the online feedback control in Section 7.2.4, along with optimizations that make an efficient solution possible. Finally, extensive simulations are presented in Section 7.2.5.

7.2.1 Problem formulation

Let $\mathbf{x} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^p$ be the state, control input and measurement of a linear system. A client contracts a cloud service to provide the control for system (7.2.1):

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t, \quad \mathbf{y}_t = \mathbf{C}\mathbf{x}_t, \tag{7.2.1}$$

as depicted in Figure 7.1. A private CaaS should solve the control problem that minimizes the cost associated to setpoint tracking while satisfying the unknown system's dynamics (7.2.1).

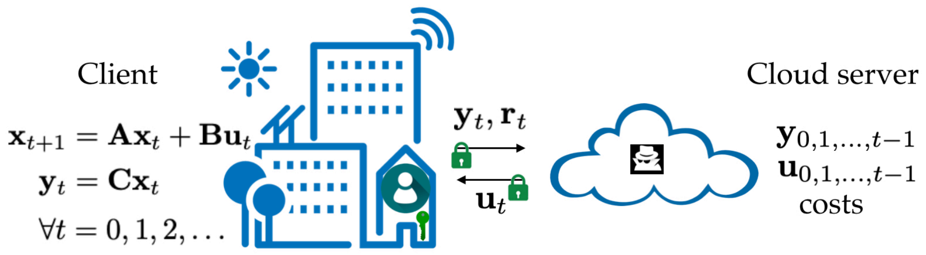

Client
$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t$
$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$
$\forall t = 0, 1, 2, \ldots$
$\mathbf{y}_t, \mathbf{r}_t$
$\mathbf{u}_t$
Cloud server
$\mathbf{y}_{0,1,\ldots,t-1}$
$\mathbf{u}_{0,1,\ldots,t-1}$
costs

Figure 7.1: Client-server diagram for data-based predictive control.

Control requirements. Assume we are given a batch of offline input-output data for system (7.2.1). Then, at every time t, given $\mathbf{u}_{0:t-1}$ and $\mathbf{y}_{0:t-1}$ we want to compute in a private and receding horizon fashion $\mathbf{u}^{*,t} \in \mathbb{R}^{Nm}$ in order to track the reference $\mathbf{r}_t$, for some costs $\bar{\mathbf{Q}}, \bar{\mathbf{R}}$, which is the solution of the LQR optimization problem:

$$\begin{aligned} &\min_{\mathbf{u},\mathbf{y}} \ \frac{1}{2}\sum_{k=t}^{N+t-1}\left(\|\mathbf{y}_k-\mathbf{r}_k\|^2_{\bar{\mathbf{Q}}}+\|\mathbf{u}_k\|^2_{\bar{\mathbf{R}}}\right) \\ &\text{s.t.} \ \ \mathbf{x}_{k+1}=\mathbf{A}\mathbf{x}_k+\mathbf{B}\mathbf{u}_k,\ \mathbf{y}_k=\mathbf{C}\mathbf{x}_k, \end{aligned} \tag{7.2.2}$$

with no prior knowledge about the system model $\mathbf{A}, \mathbf{B}, \mathbf{C}$. We also want (7.2.2) to perform well under unknown small process and measurement noise and bias.

In this section, we focus on investigating an approximation of problem (7.2.2) without hard constraints on the inputs or outputs of the system. In Section 7.2.2, we describe a data-driven reformulation of (7.2.2) that is encryption-friendly.

Privacy requirements. In the scenario we consider, the cloud service should not be able to infer anything about the client's private data, which consists of the input signals $\mathbf{u}$, the output signals $\mathbf{y}$, the model $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and state $\mathbf{x}$ (the last four quantities being unknown in a data-driven control problem), and any intermediate values. The costs $\bar{\mathbf{Q}}, \bar{\mathbf{R}}$ can be chosen by the cloud, as part of the CaaS service or chosen by the client. The cloud service is considered to be *semi-honest*, as defined in Definition 2.1.1, which means that it does not deviate from the client's specifications. We assume this to be the case because the cloud server is under contract.

The formal privacy definition is captured by Definition 2.2.8, i.e., all information obtained by the server after the execution of the protocol (while also keeping a record of the intermediate computations) on the client's private data and its own data is indistinguishable from what can be obtained solely from the inputs and outputs available to the server.

Efficiency requirements. The computation time of the control action at the current time step should not exceed the sampling time, which is the time until the next measurement is fetched. In the private case, we expect a large overhead for complex controllers and consider applications with sampling time of the order of minutes. Moreover, we require the client to be exempt from heavy computation and communication, and the bulk of the computation (in the allowable time) should be performed at the server side.

7.2.2 Encryption-aware formulation

We can formulate this control problem inspired by the behavioral and subspace identification frameworks explored in [35, 70, 82, 158, 226]. First, we introduce some preliminary concepts. A *block-Hankel matrix* for the input signal $\mathbf{u} = \begin{bmatrix}\mathbf{u}_0^\intercal & \mathbf{u}_2^\intercal & \dots & \mathbf{u}_{T-1}^\intercal\end{bmatrix}^\intercal \in \mathbb{R}^{mT}$ and a positive integer $L \leq T$ is given by the following:

$$H_L\mathbf{U} := \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \dots & \mathbf{u}_{T-L} \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_{T-L+1} \\ \vdots & & \ddots & \vdots \\ \mathbf{u}_{L-1} & \mathbf{u}_L & \dots & \mathbf{u}_{T-1} \end{bmatrix}. \tag{7.2.3}$$

Furthermore, the signal $\mathbf{u}$ is persistently exciting of order L if $H_L\mathbf{U} \in \mathbb{R}^{mL\times(T-L+1)}$ is full row rank.

Let us construct block-Hankel matrices for the "past" and "future" input and output data, $\mathbf{u} \in \mathbb{R}^{mT}$, respectively, $\mathbf{y} \in \mathbb{R}^{pT}$, for M samples for the past data and N samples for the future data, and $S := T - M - N + 1$:

$$\begin{aligned} \mathbf{U}^p &:= \begin{bmatrix}\mathbf{I}_{mM} & \mathbf{0}_{mM\times mN}\end{bmatrix} \cdot H_{M+N}\mathbf{U} \in \mathbb{R}^{mM\times S} \\ \mathbf{U}^f &:= \begin{bmatrix}\mathbf{0}_{mN\times mM} & \mathbf{I}_{mN}\end{bmatrix} \cdot H_{M+N}\mathbf{U} \in \mathbb{R}^{mN\times S} \\ \mathbf{Y}^p &:= \begin{bmatrix}\mathbf{I}_{pM} & \mathbf{0}_{pM\times pN}\end{bmatrix} \cdot H_{M+N}\mathbf{Y} \in \mathbb{R}^{pM\times S} \\ \mathbf{Y}^f &:= \begin{bmatrix}\mathbf{0}_{pN\times pM} & \mathbf{I}_{pN}\end{bmatrix} \cdot H_{M+N}\mathbf{Y} \in \mathbb{R}^{pN\times S}. \end{aligned} \tag{7.2.4}$$

7.2.2.1 Offline feedback data-based predictive control problem

Assume that we are given precollected input and output data, with $\mathbf{U}^p, \mathbf{Y}^p, \mathbf{U}^f, \mathbf{Y}^f$ the respective past and future Hankel matrices, for some past and future horizons M, N. Assume also that the precollected input is persistently exciting.

Assumption 7.2.1 (Data richness)**.** We assume the offline precollected input trajectory $\mathbf{u}$

is persistently exciting of order $M+N+n$, where n is the order of the system [226].

Informally, the *Fundamental Lemma* in [226] says that under Assumption 7.2.1, the behavior of a controllable linear system can be replaced by a data-based representation which only uses the trajectories of the system, specifically, the column span of the block-Hankel matrix for input-output trajectories, composed of $\mathbf{U}^p, \mathbf{Y}^p, \mathbf{U}^f, \mathbf{Y}^f$.

Fix a time t and let $\bar{\mathbf{u}}_t = \mathbf{u}_{t-M:t-1}$ be the batch vector of the last M inputs. The batch vector of the last M outputs $\bar{\mathbf{y}}_t$ is defined similarly. If $M \geq n$, then the standard LQR problem (7.2.2) can be re-formulated as the following data-based predictive control problem [70]:

$$\begin{aligned} \min_{\mathbf{g},\mathbf{u},\mathbf{y}} \ & \frac{1}{2} \sum_{k=t}^{N+t-1} \left(\|\mathbf{y}_k - \mathbf{r}_k\|^2_{\mathbf{Q}} + \|\mathbf{u}_k\|^2_{\mathbf{R}} \right) \\ \text{s.t.} \ & \begin{bmatrix} \mathbf{U}^p \\ \mathbf{Y}^p \\ \mathbf{U}^f \\ \mathbf{Y}^f \end{bmatrix} \cdot \mathbf{g} = \begin{bmatrix} \bar{\mathbf{u}}_t \\ \bar{\mathbf{y}}_t \\ \mathbf{u} \\ \mathbf{y} \end{bmatrix}, \end{aligned} \tag{7.2.5}$$

where the state representation has been replaced with the precollected data and $\mathbf{g}$ represents the preimage of the system's trajectory with respect to the precollected block-Hankel matrices. Note that $\mathbf{u}^{*,t}_{[1:m]}$, the first m elements of $\mathbf{u}^{*,t}$, will be input into the system in a receding horizon fashion, and $\mathbf{y}^{*,t}$ is the predicted output.

We now depart from the behavioral control problem (7.2.5) considered in the existing literature and explore a more encryption-friendly form.

First, we rewrite (7.2.5) as a minimization problem depending only on $\mathbf{g}$ by enforcing $\mathbf{u} = \mathbf{U}^f\mathbf{g}$ and $\mathbf{y} = \mathbf{Y}^f\mathbf{g}$. Second, in practice, there will be noise affecting the output measurement, as well as precision errors induced by encryption, which might prevent an exact solution to the equality constraint of (7.2.5). Hence, we prefer a least-squares penalty approach to the equality constraint in (7.2.5) with regularization weights λ_y and λ_u. Finally, to reduce overfitting and precision errors due to encryption, we also penalize the magnitude of $\mathbf{g}$ through ℓ_2-norm regularization. We opt for two-norm regularizations to obtain better efficiency for the encrypted algorithm, as well as more robustness with respect to noise, and uniqueness of the solution $\mathbf{g}^{*,t}$. We note that ℓ_2-norm regularization corresponds to the two-norm robustness with respect to the output noise and is preferred also in works that only consider the performance of the algorithm, and not its privacy, e.g. [35, 125]. (We will address a ℓ_1-norm regularization of the model surrogate in Section 7.3.) The above consideration yield the following formulation, where $\mathbf{Q} = \text{blockdiag}(\bar{\mathbf{Q}}, \ldots, \bar{\mathbf{Q}})$ and $\mathbf{R} = \text{blockdiag}(\bar{\mathbf{R}}, \ldots, \bar{\mathbf{R}})$ and we reuse the notation $\mathbf{r}_t$ for the batch reference signal:

$$\min_{\mathbf{g}} \frac{1}{2} \left(\|\mathbf{Y}^f \mathbf{g} - \mathbf{r}_t\|_{\mathbf{Q}}^2 + \|\mathbf{U}^f \mathbf{g}\|_{\mathbf{R}}^2 + \lambda_y \|\mathbf{Y}^p \mathbf{g} - \bar{\mathbf{y}}_t\|_2^2 + \lambda_u \|\mathbf{U}^p \mathbf{g} - \bar{\mathbf{u}}_t\|_2^2 + \lambda_g \|\mathbf{g}\|_2^2 \right). \quad (7.2.6)$$

Note that the resulting problem (7.2.6) is an approximation of (7.2.5). Finally, (7.2.6) can be written as a quadratic program in (7.2.7), with $\mathbf{M} \in \mathbb{R}^{S \times S}$ defined in (7.2.8):

$$\min_{\mathbf{g}} \ \frac{1}{2} \mathbf{g}^\intercal \mathbf{M} \mathbf{g} - \mathbf{g}^\intercal \left(\mathbf{Y}^{f\intercal} \mathbf{Q} \mathbf{r}_t + \lambda_y \mathbf{Y}^{p\intercal} \bar{\mathbf{y}}_t + \lambda_u \mathbf{U}^{p\intercal} \bar{\mathbf{u}}_t \right), \quad (7.2.7)$$

$$\mathbf{M} := \mathbf{Y}^{f\intercal} \mathbf{Q} \mathbf{Y}^f + \mathbf{U}^{f\intercal} \mathbf{R} \mathbf{U}^f + \lambda_y \mathbf{Y}^{p\intercal} \mathbf{Y}^p + \lambda_u \mathbf{U}^{p\intercal} \mathbf{U}^p + \lambda_g \mathbf{I}. \quad (7.2.8)$$

Since (7.2.7) is a strongly convex optimization problem (ensured by the regularization term $\lambda_g \mathbf{I}$), we can find the optimal value for $\mathbf{g}$ by zeroing the gradient of the objective function:

$$\mathbf{g}^{*,t} = \mathbf{M}^{-1} \left(\mathbf{Y}^{f\intercal} \mathbf{Q} \mathbf{r}_t + \lambda_y \mathbf{Y}^{p\intercal} \bar{\mathbf{y}}_t + \lambda_u \mathbf{U}^{p\intercal} \bar{\mathbf{u}}_t \right). \quad (7.2.9)$$

Going back to the control input for the current time step we want to compute, we obtain from (7.2.9):

$$\mathbf{u}^{*,t} = \mathbf{U}^f \mathbf{M}^{-1} \left(\mathbf{Y}^{f\intercal} \mathbf{Q} \mathbf{r}_t + \lambda_y \mathbf{Y}^{p\intercal} \bar{\mathbf{y}}_t + \lambda_u \mathbf{U}^{p\intercal} \bar{\mathbf{u}}_t \right), \quad (7.2.10)$$

from which we select the first m elements and input them to the unknown system.

As seen from (7.2.10), the controller has the form of a dynamic output-feedback law, where the feedback terms are computed using only the offline precollected data.

7.2.2.2 Closeness between approximate and original problems

We prove that if we select small enough regularization coefficient λ_g and large enough penalty coefficients λ_y, λ_u, then the solution of the approximate problem (7.2.6) is very close to the minimum norm solution of the behavioral control problem (7.2.5). The main challenge in the proof of Theorem 7.2.2 is that the past and future Hankel matrices are rank-deficient. Hence the behavioral control problem (7.2.5) involves singular matrices in both the objective and the constraints, and has multiple optimal solutions. This requires an involved analysis, where we deal with pseudo-inverses and subspaces, that we show in Appendix F.1. To formally state the result we need two definitions. The set of optimal solutions of (7.2.5) is denoted by $\mathcal{G}_{\text{opt}}$:

$$\mathcal{G}_{\text{opt}} := \{\mathbf{g} : \mathbf{g} \text{ solves } (7.2.5)\}. \tag{7.2.11}$$

The minimum norm element of $\mathcal{G}_{\text{opt}}$ is defined as

$$\mathbf{g}_{\text{min}} := \arg \min_{\mathbf{g} \in \mathcal{G}_{\text{opt}}} \|\mathbf{g}\|_2. \tag{7.2.12}$$

Theorem 7.2.2. *Consider the original behavioral control problem* (7.2.5) *and its approximation* (7.2.6)*, with penalty coefficient* $\lambda_y = \lambda_u = \lambda > 0$ *and regularization coefficient* $\lambda_g > 0$*. Let* $\mathbf{g}_{\text{min}}$ *be the minimum norm solution of the behavioral problem* (7.2.5) *as defined in* (7.2.12)*. Let* $\mathbf{g}^*$ *be the optimal solution of the approximate problem* (7.2.6)*. Then:*

$$\|\mathbf{g}_{\text{min}} - \mathbf{g}^*\|_2 \to 0, \tag{7.2.13}$$

as $(\lambda_g, \lambda) \to (0, \infty)$ *restricted on the set* $\lambda_g > 0, \lambda > 0$*.*

Although the behavioral problem has infinite solutions, due to the regularization term the approximate problem (7.2.6) will only return a solution close to the minimum norm one $\mathbf{g}_{\text{min}}$.

7.2.2.3 Online feedback data-based predictive control problem

To satisfy Assumption 7.2.1, it is necessary that the precollected input signal has length at least $(m+1)(M+N+n)-1$, cf. [70]. In practice, Assumption 7.2.1 might be violated if less precollected data is available. Another issue with the precollected data is that it can be affected by perturbations, e.g., measurement noise. To alleviate these issues, we prefer an online algorithm, where the Hankel matrices $H_{M+N}\mathbf{U}$ and $H_{M+N}\mathbf{Y}$ are updated at each time step for another $\bar{T}$ steps with the used control input and the corresponding output measured (while keeping the block Hankel form). This approach empirically ensures richness of the data and robustness to perturbations. However, it is important that $\bar{T}$ is not too large, in order to prevent overfitting.

Note that the precollected and the online data in this online algorithm belong to different trajectories. For this reason, we must compute the Hankel matrix for each data set separately, then append them in a single matrix [218]. This means that the online adaptation of the matrices $\mathbf{U}^p, \mathbf{U}^f, \mathbf{Y}^p, \mathbf{Y}^f$ can only start at time $t = M+N-1$, when we obtain enough data $\mathbf{u}_0, \mathbf{y}_0, \ldots, \mathbf{u}_{M+N-1}, \mathbf{y}_{M+N-1}$ to fill the first Hankel matrix column for the online data set. We call this phase between $t = M-1$ and $t = M+N-1$ trajectory concatenation.

The data-driven LQR algorithm is given in Algorithm 7.2.1.

Algorithm 7.2.1: Online data-based predictive control algorithm

Input: $\bar{\mathbf{u}}_t$, $\bar{\mathbf{y}}_t$, $\mathbf{U}^p$, $\mathbf{U}^f$, $\mathbf{Y}^p$, $\mathbf{Y}^f$, $\mathbf{Q}$, $\mathbf{R}$, λ_y, λ_u, λ_g, $S = T - M - N + 1$, $\bar{T}$.

Output: $\mathbf{u}_t$ for $t = 0, 1, \ldots$.

```
1: for $t = 0, 1, \ldots, M-1$ do
2:     Randomly select and input $\mathbf{u}_t$ and measure the output $\mathbf{y}_t$.
3: end for
4: Construct $\bar{\mathbf{u}}_M = \mathbf{u}_{0:M-1}$ and $\bar{\mathbf{y}}_M = \mathbf{y}_{0:M-1}$.
5: for $t = M, M+1, \ldots, M+N+\bar{T}-1$ do
6:     Solve (7.2.7) for $\mathbf{g}^{*,t}$ and obtain (7.2.9).
7:     Compute $\mathbf{u}^{*,t} = \mathbf{U}^f \mathbf{g}^{*,t}$ and obtain (7.2.10).
8:     Input to the system $\mathbf{u}_t = \mathbf{u}^{t,*}_{[1:m]}$ and measure the output $\mathbf{y}_t$.
```

```
9:      Update $\bar{\mathbf{u}}_t$ and $\bar{\mathbf{y}}_t$ to be the last $M$ components of $\begin{bmatrix}\bar{\mathbf{u}}_t^\intercal & \mathbf{u}_t^\intercal\end{bmatrix}^\intercal$ and $\begin{bmatrix}\bar{\mathbf{y}}_t^\intercal & \mathbf{y}_t^\intercal\end{bmatrix}^\intercal$, respectively.
10:     if $t = M + N - 1$ then
11:         Add $\mathbf{u}_{0:t}$ and $\mathbf{y}_{0:t}$ to the $S+1$'th column of the trajectory Hankel matrices $H_{M+N}\mathbf{U}$ and $H_{M+N}\mathbf{Y}$.
12:     else if $t > M + N - 1$ then
13:         Set $S = S + 1$. Use $\mathbf{u}_t$, respectively $\mathbf{y}_t$, to add a new column to $H_{M+N}\mathbf{U}$, respectively $H_{M+N}\mathbf{Y}$, while keeping the block Hankel matrix form.
14:     end if
15: end for
16: while $t \geq M + N + \bar{T}$ do
17:     Execute lines 6–9.
18: end while
```

7.2.2.4 Co-design of encrypted controller

According to the requirements in Section 7.2.1, the challenges for the encrypted data-based predictive control can be summarized as:

- The computations are iterative and not readily formulated as low-depth arithmetic circuits.
- The problem is computationally intensive: it requires large storage, large matrix inversions and many consecutive matrix multiplications.
- The precision loss due to encrypted computations might affect the control performance.

To deal with these challenges, we design an encrypted version of the closed-form solution (7.2.9) of the control problem stated in Section 7.2.1 and manipulate the computations involving matrix inverses to reduce the multiplicative depth needed. We employ the CKKS homomorphic scheme described in Preamble 2.5.3 to address the precision challenge.

First, we approximate problem (7.2.5) into optimization problem (7.2.7), as shown in Section 7.2.2, to simplify the encrypted computations and to avoid infeasibility due to noise and encryption errors. Second, we aim to write the computation of the solution of (7.2.7) as a low-depth arithmetic circuit. Despite being preferable to an iterative algorithm for solving the optimization problem (7.2.7) (that would increase the depth at every iteration), the closed-form solution (7.2.9) involves the encrypted inversion of a matrix, which cannot be generally written as a low-depth arithmetic circuit. This is not a problem in the offline feedback version of the problem (7.2.7), because this matrix inversion is required only once and such complex computations can be all performed offline, leaving only three encrypted matrix-vector multiplications to be performed at each time step, as shown in Section 7.2.3. However, in the online feedback version of the problem, this matrix inversion and many consecutive matrix-vector multiplications are required at every time step. Nevertheless, we leverage the special structure of the matrix to be inverted by using Schur's complement [32, Ch. 0] to reformulate the inverse computation as some lower multiplicative depth matrix-vector multiplications and one scalar division. Furthermore, to avoid performing the division, which is costly on encrypted data, the server sends to the client the denominator such that the client can return its inverse to the server, turning division into multiplication.

We also perform further valuable optimizations in terms of ciphertext packing, adding redundancy in the encoded values, and manipulating the computations to be performed in a SIMD mode, which are crucial to the tractability of the solution.

The multiplicative depth of the arithmetic circuit computing $\mathbf{u}^{*,t}_{[0:m-1]}$ can be reduced through reordering of the intermediate operations. Given a circuit that computes x and a circuit that computes y, with multiplicative depth $d(x)$ and $d(y)$, the multiplicative depth of the circuit that evaluates them in parallel and then computes their product is: $d(xy) = \max(d(x), d(y)) + 1$. The multiplicative depth should not be confounded with the number of multiplications. Judiciously choosing the order in which to perform the multiplications can reduce the multiplicative depth of the result. For example, consider we want to compute $y = x_1x_2x_3x_4$, where $d(x_i) = 0$. If we sequentially perform the multiplications, we obtain $d(y) = 3$. However, if we perform $y = (x_1x_2)(x_3x_4)$, we obtain $d(y) = 2$. In turn, this means that the ciphertext encrypting the resulting quantity y will have 2 consumed levels. Note that the fewer the number of levels necessary in the computation, the cheaper the operations on ciphertexts are. We will use this trick in Section 7.2.4.

As explained in Preamble 2.5.3, each ciphertext is created with a number of levels corresponding to the depth of multiplications it can support before the underlying plaintext is corrupted by noise. After each multiplication and rescaling, a level is removed from the ciphertext, leading to more efficient computations. We exploit this fact in our solution, by making sure we compute each operation on ciphertexts that have the minimum required number of moduli.

7.2.3 Offline feedback encrypted solution

Recall the closed-form solution of the optimization problem (7.2.7). We now compute the multiplicative depth of the quantities of interest for consecutive time steps. First, the quantities $\bar{\mathbf{u}}_0$, $\bar{\mathbf{y}}_0$, the reference signal and measurements are freshly encrypted at every time step t, hence they are inputs to the circuit that computes $\mathbf{u}_t$ and have a multiplicative depth of 0. This means, $\forall t \geq 0$, $d(\mathbf{y}_t) = d(\mathbf{r}_t) = d(\bar{\mathbf{u}}_0) = d(\bar{\mathbf{y}}_0) = 0$. Second, we assume that all the quantities obtained offline will have multiplicative depth 0. We do not address the offline computations here; since these computations depend only on the offline data and are one-time, expensive secure solutions can be used, e.g., the cloud could perform the encrypted computations, then perform bootstrapping to refresh the ciphertexts [60, 63]. Hence, the cloud has fresh encryptions of the following products: $\mathbf{A}_r := \begin{bmatrix}\mathbf{I}_m & \mathbf{0}_{(N-1)m}\end{bmatrix}\mathbf{U}^f\mathbf{M}^{-1}\mathbf{Y}^{f\intercal}\mathbf{Q} \in \mathbb{R}^{m\times pN}$, $\mathbf{A}_y := \begin{bmatrix}\mathbf{I}_m & \mathbf{0}_{(N-1)m}\end{bmatrix}\mathbf{U}^f\mathbf{M}^{-1}\lambda_u\mathbf{Y}^{p\intercal} \in \mathbb{R}^{m\times pM}$, $\mathbf{A}_u := \begin{bmatrix}\mathbf{I}_m & \mathbf{0}_{(N-1)m}\end{bmatrix}\mathbf{U}^f\mathbf{M}^{-1}\lambda_u\mathbf{U}^{p\intercal} \in \mathbb{R}^{m\times mM}$, and the underlying messages are inputs to the circuit and they have multiplicative depth 0. Then:

$$\mathbf{u}_t = \mathbf{A}_r\mathbf{r}_t + \mathbf{A}_y\bar{\mathbf{y}}_t + \mathbf{A}_u\bar{\mathbf{u}}_t. \tag{7.2.14}$$

For $t = 0$, we obtain $d(\mathbf{u}_0) = \max(d(\mathbf{r}_0) + 1, d(\bar{\mathbf{y}}_0) + 1, d(\bar{\mathbf{u}}_0) + 1) = 1$. Generalizing:

$$d(\mathbf{u}_t) = d(\bar{\mathbf{u}}_t) + 1, \quad t \geq 0. \tag{7.2.15}$$

At time $t + 1$, $\bar{\mathbf{u}}_{t+1}$ will be updated by $\mathbf{u}_t$.

We next present three encrypted methods of computing (7.2.14), that exhibit trade-offs in depth, memory and communication.

If we individually encrypt each element of the quantities in (7.2.14) in a separate ciphertext (the inputs to the circuit computing $\mathbf{u}_t$ will be all elements, rather than three vectors and three matrices), $\bar{\mathbf{u}}_{t+1}$ will have the multiplicative depth of $\mathbf{u}_t$ and $d(\mathbf{u}_t) = t+1$, but memory-wise, a step will require $(m+1)(pN+pM+mM)$ ciphertexts.

It is more efficient from both storage and computation points of view to encode a vector instead of a scalar in a ciphertext and perform the matrix-vector multiplications by a diagonal method. This vector encoding requires a different analysis.

7.2.3.1 Diagonal method

We explore a method for efficient matrix-vector multiplication where the diagonals of the matrix are encrypted in separate ciphertexts (the notion of diagonal is extended for rectangular matrices) [7, 132].

Tall matrix. Consider we want to multiply a tall matrix $\mathbf{S} \in \mathbb{R}^{u\times v}$, $u \geq v$, by a vector $\mathbf{p} \in \mathbb{R}^v$. To this end, we extract the extended diagonals $\mathbf{d}_i$ of $\mathbf{S}$ such that $\mathbf{d}_i$ will have as many elements as the number of rows u, for $i = 0, \dots, v-1$. The corresponding result $\mathbf{q} = \mathbf{Sp}$ is computed as follows, where $\tilde{\mathbf{p}}$ is $\mathbf{p}$ concatenated with itself as many times such that $\tilde{\mathbf{p}} \in \mathbb{R}^u$:

$$\mathbf{q} = \sum_{i=0}^{v-1} \mathbf{d}_i \odot \rho(\tilde{\mathbf{p}}, i).$$

This way of computing the encrypted matrix-vector multiplication is highly efficient when using ciphertext batching. Specifically, we encrypt each extended diagonal of $\mathbf{S}$ in one ciphertext, which gives us a storage of only $v \leq u$ ciphertexts and we encrypt $\mathbf{p}$ repeatedly in one ciphertext. We can compute the necessary rotations in an efficient way using hoisted rotations [116]. The encrypted result is then computed as:

$$\mathcal{E}_{\mathrm{v0}}(\mathbf{q}) = \sum_{i=0}^{v-1} \mathcal{E}_{\mathrm{v0}}(\mathbf{d}_i) \odot \rho(\mathcal{E}_{\mathrm{vv}}(\mathbf{p}), i). \tag{7.2.16}$$

Wide matrix. In the case of a wide matrix $\mathbf{S} \in \mathbb{R}^{u \times v}$ with $u \leq v$, instead of using extended diagonals, we use reduced diagonals. One reduced diagonal $\tilde{\mathbf{d}}_i$, for $i = 0, \dots, v-1$ will have as many elements as the number of rows u. The storage is not as efficient since we have to encrypt $v \geq u$ ciphertexts for the diagonals. Apart from this, the computation is identical to (7.2.16) and produces $\mathcal{E}_{\mathrm{v}0}(\mathbf{q})$.

We can improve the storage (at the cost of some extra rotations) when the number of columns divides the number of rows. We extract the extended diagonals $\mathbf{d}_i \in \mathbb{R}^v$ for $i = 0, \dots, u-1$. The computation of the matrix-vector multiplication takes the following form:

$$\mathbf{z} = \sum_{i=0}^{u-1} \mathbf{d}_i \odot \rho(\mathbf{p}, i), \qquad \mathbf{q} = \left[\mathbf{z} + \sum_{j=1}^{\lceil \log(v/u) \rceil} \rho\left(\mathbf{z}, u \cdot 2^{\lceil \log(v/u) \rceil - j}\right)\right]_{[0:u-1]},$$

and produces $\mathcal{E}_{\mathrm{v}*}(\mathbf{q})$ instead of $\mathcal{E}_{\mathrm{v}0}(\mathbf{q})$. When $u << v$, it is useful to append rows of zeros such that u divides v, and then use this method.

Let us return to (7.2.14). In general, $\mathbf{A}_r, \mathbf{A}_y, \mathbf{A}_u$ are wide matrices, but there can exist particular cases when $m >> p$ or when we want to compute all $\mathbf{u}^{*,t}$, leading to tall matrices.

We assume that at the onset of time step t, the cloud server has $\mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{u}}_t), \mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{y}}_t), \mathcal{E}_{\mathrm{v}0}(\mathbf{r}_t)$. The cloud server also has $\mathcal{E}_{\mathrm{v}0}(\mathrm{diag}_i \mathbf{A}_r)$, $\mathcal{E}_{\mathrm{v}0}(\mathrm{diag}_i \mathbf{A}_y)$, $\mathcal{E}_{\mathrm{v}0}(\mathrm{diag}_i \mathbf{A}_u)$, i.e., each extended diagonal (whose exact definition varies with the shape of the matrix: tall or wide) of the matrices $\mathbf{A}_r, \mathbf{A}_y, \mathbf{A}_u$ is encrypted in a separate ciphertext. In order to be able to use the diagonal methods, the cloud server first obtains $\mathcal{E}_{\mathrm{vv}}(\bar{\mathbf{u}}_t), \mathcal{E}_{\mathrm{vv}}(\bar{\mathbf{y}}_t), \mathcal{E}_{\mathrm{vv}}(\mathbf{r}_t)$ by rotating and adding $\mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{u}}_t), \mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{y}}_t), \mathcal{E}_{\mathrm{v}0}(\mathbf{r}_t)$. This vector packing substantially reduces memory: $3 + \max(m, pN) + \max(m, pM) + mM$ ciphertexts in the worst case and $3 + \min(m, pN) + \min(m, pM) + m$ in the best case, depending on the shape of the matrix and the type of diagonal chosen, but will require an extra level. From these quantities, the cloud server computes and sends back to the client one ciphertext containing $\mathcal{E}_{\mathrm{v}0}(\mathbf{u}_t)$ or $\mathcal{E}_{\mathrm{v}*}(\mathbf{u}_t)$, depending on the type of diagonals chosen. Assume the cloud server obtained $\mathcal{E}_{\mathrm{v}0}(\mathbf{u}_t)$. After that, the cloud server has to create $\mathcal{E}_{\mathrm{vv}}(\bar{\mathbf{u}}_{t+1})$ from $\mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{u}}_t)$ and $\mathcal{E}_{\mathrm{v}0}(\mathbf{u}_t)$, so it:

- rotates $\mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{u}}_t)$ by m positions to the left;

- rotates $\mathcal{E}_{\mathrm{v}0}(\mathbf{u}_t)$ to the right by $(M-1)m$ positions, then adds it to $\rho(\mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{u}}_t), m)$. This yields $\mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{u}}_{t+1}) + \mathcal{E}_{\mathrm{v}0}([0\,0\,\ldots\,0\,(\bar{\mathbf{u}}_t)_{[0:m-1]}])$, the last part appearing because of the rotation to the left;

- masks the result in order to truly obtain $\mathcal{E}_{\mathrm{v}0}(\bar{\mathbf{u}}_{t+1})$ (otherwise, the summing and rotation would not produce the correct result);

- repeatedly rotates and adds to obtain $\mathcal{E}_{\mathrm{vv}}(\bar{\mathbf{u}}_{t+1})$.

Because of the CKKS encoding through the Discrete Fourier Transform, (which enables SIMD multiplications, unlike other possible encodings), the masking operation needs to consume a level in order to preserve precision [62]. The reason is that, in order to not lose precision when performing the rounding of the inverse FFT in the canonical embedding, the masking vector has to also be scaled upon encoding by a large positive scaling factor, which then should be removed after the multiplication by a rescaling operation, which consumes a level. Obtaining $\mathcal{E}_{\mathrm{v}*}(\mathbf{u}_t)$ instead of $\mathcal{E}_{\mathrm{v}0}(\mathbf{u}_t)$ does not change the analysis, since the same masking applied in the third step also removes the junk elements in $\mathcal{E}_{\mathrm{v}*}$. Furthermore, the analysis (not shown here) of other less efficient matrix-vector multiplication methods, such as row and column methods, also require an extra masking.

For $t \geq 1$, equation (7.2.15) becomes: $d(\mathbf{u}_t) = d(\bar{\mathbf{u}}_t) + 1 = d(\mathbf{u}_{t-1}) + 2 = 2t + 1$. $\mathcal{E}_{\mathrm{vv}}(\bar{\mathbf{y}}_{t+1})$ can be updated the same way $\mathcal{E}_{\mathrm{vv}}(\bar{\mathbf{u}}_{t+1})$ is updated. However, it is the same cost for the client to encrypt $\mathcal{E}_{\mathrm{v}0}(\mathbf{y}_t)$ and $\mathcal{E}_{\mathrm{vv}}(\bar{\mathbf{y}}_{t+1})$, so it can encrypt and send the latter. Whenever the allocated multiplicative budget is exhausted, the server can ask the client to send a fresh encryption of $\bar{\mathbf{u}}_t$, at little extra cost.

Nevertheless, a reasonable and inexpensive option is to ask the client to send along with the encryption of $\bar{\mathbf{y}}_t$ a fresh encryption of $\bar{\mathbf{u}}_t$ at every time step. This implies that $d(\mathbf{u}_t) = 1$, i.e., a multiplicative budget of only 1 is required for computing the control input for no matter how many time steps.

Theorem 7.2.3. *The encrypted offline data-based predictive control algorithm in Section 7.2.3 achieves client privacy with respect to the server, cf. Definition 2.2.8.*

The proof is given in Appendix F.4.

7.2.4 Online feedback encrypted solution

7.2.4.1 Computing the solution using arithmetic circuits

Being capable of evaluating polynomials via homomorphic encryption theoretically gives us the possibility of evaluating any function arbitrarily close (using Taylor series for example). In practice, the multiplicative depth and loss of precision of these approximations limit the types of functions we can evaluate. Consequently, division is still prohibitive. Furthermore, large matrix inversion should be judiciously performed, in order to avoid computing products of as many factors as the number of rows.

In this section, we will use the following shorter notation:

$$H\mathbf{U} := \begin{bmatrix} \mathbf{U}^p \\ \mathbf{U}^f \end{bmatrix}, \; H\mathbf{Y} := \begin{bmatrix} \mathbf{Y}^p \\ \mathbf{Y}^f \end{bmatrix}, \; h\mathbf{u} := \begin{bmatrix} \mathbf{u}^p \\ \mathbf{u}^f \end{bmatrix}, \; h\mathbf{y} := \begin{bmatrix} \mathbf{y}^p \\ \mathbf{y}^f \end{bmatrix},$$

where $\mathbf{u}^{p,f}, \mathbf{y}^{p,f}$ are vectors added at the end of the Hankel matrices $\mathbf{U}^{p,f}, \mathbf{Y}^{p,f}$. We will drop the time subscripts t for conciseness, using prime to denote the next time step.

Let us investigate the inversion of matrix $\mathbf{M}$. For step $t = 0$, all the values involved in computing $\mathbf{u}^*$ in (7.2.10), are precollected and can be computed offline. This includes the inverse $\mathbf{M}^{-1}$ and other matrix products. However, at the next time step, cf. line 13 in Algorithm 7.2.1:

$$H\mathbf{U}' := \begin{bmatrix} H\mathbf{U} & h\mathbf{u} \end{bmatrix}, \quad H\mathbf{Y}' := \begin{bmatrix} H\mathbf{Y} & h\mathbf{y} \end{bmatrix} \tag{7.2.17}$$

The last m elements on the last column of $H\mathbf{U}'$, respectively of $h\mathbf{u}$, are the values of $\mathbf{u}_t$ at the previous time step.

Then, the matrix $\mathbf{M}' \in \mathbb{R}^{(S+1)\times(S+1)}$:

$$\mathbf{M}' := H\mathbf{Y}'^{\intercal} \begin{bmatrix} \lambda_y \mathbf{I} & \\ & \mathbf{Q} \end{bmatrix} H\mathbf{Y}' + H\mathbf{U}'^{\intercal} \begin{bmatrix} \lambda_u \mathbf{I} & \\ & \mathbf{R} \end{bmatrix} H\mathbf{U}' + \lambda_g \mathbf{I} \tag{7.2.18}$$

is a *rank-1 update* of matrix $\mathbf{M}$. Let

$$\mu := h\mathbf{y}^\intercal \begin{bmatrix} \lambda_y \mathbf{I} & \\ & \mathbf{Q} \end{bmatrix} h\mathbf{y} + h\mathbf{u}^\intercal \begin{bmatrix} \lambda_u \mathbf{I} & \\ & \mathbf{R} \end{bmatrix} h\mathbf{u} + \lambda_g, \tag{7.2.19}$$

$$\mathbf{m} := h\mathbf{y}^\intercal \begin{bmatrix} \lambda_y \mathbf{I} & \\ & \mathbf{Q} \end{bmatrix} H\mathbf{Y} + h\mathbf{u}^\intercal \begin{bmatrix} \lambda_u \mathbf{I} & \\ & \mathbf{R} \end{bmatrix} H\mathbf{U}. \tag{7.2.20}$$

Specifically, $\mathbf{M}'$ will have the following form:

$$\mathbf{M}' = \begin{bmatrix} \mathbf{M} & \mathbf{m}^\intercal \\ \mathbf{m} & \mu \end{bmatrix}. \tag{7.2.21}$$

Schur's complement [32, Ch. 0] gives an efficient way to compute $\mathbf{M}'^{-1}$ from $\mathbf{M}^{-1}$ (assuming $\mathbf{M}^{-1}$ exists), by inverting a scalar s and computing a few matrix-vector multiplications:

$$s := \mu - \mathbf{m}\mathbf{M}^{-1}\mathbf{m}^\intercal, \tag{7.2.22}$$

$$\mathbf{M}'^{-1} = \begin{bmatrix} \mathbf{M}^{-1} + \frac{1}{s}\mathbf{M}^{-1}\mathbf{m}^\intercal\mathbf{m}\mathbf{M}^{-1} & -\frac{1}{s}\mathbf{M}^{-1}\mathbf{m}^\intercal \\ -\frac{1}{s}\mathbf{m}\mathbf{M}^{-1} & \frac{1}{s} \end{bmatrix}. \tag{7.2.23}$$

To avoid performing the division $1/s$ on encrypted data, the server will ask the client to perform it on plaintext and send back an encryption of result.

7.2.4.2 Reducing memory and depth of the arithmetic circuit

In this chapter, for simplicity, we consider diagonal cost matrices $\mathbf{Q}$ and $\mathbf{R}$. Otherwise, the multiplication by these matrices requires more complicated encrypted operations.

At a given time, the cloud server has (in an encrypted form) $H\mathbf{U}, H\mathbf{Y}, \bar{\mathbf{u}}, \bar{\mathbf{y}}$ and $\mathbf{M}^{-1}$, along with the unencrypted costs $\mathbf{Q}, \mathbf{R}$, penalties $\lambda_g, \lambda_u, \lambda_y$ and reference signal $\mathbf{r}$. The cloud has to compute the equivalent formulation of (7.2.10):

$$\begin{aligned} \mathbf{u} &= \begin{bmatrix} \mathbf{0}_{mM} & \mathbf{I}_m & \mathbf{0}_{(N-1)m} \end{bmatrix} H\mathbf{U}' \mathbf{M}'^{-1}\mathbf{Z}, \\ \mathbf{Z} &= H\mathbf{Y}'^{\intercal} \begin{bmatrix} \lambda_y \mathbf{I} & \\ & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{y}} \\ \mathbf{r} \end{bmatrix} + H\mathbf{U}'^{\intercal} \begin{bmatrix} \lambda_u \mathbf{I} & \\ & \mathbf{R} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{u}} \\ \mathbf{0} \end{bmatrix}. \end{aligned} \tag{7.2.24}$$

There is a vast number of parameters to tune in the encrypted implementation, such as the packing method from the messages into the plaintexts, the storage redundancy, the order of performing the operations and choice of refreshing some ciphertexts. The trade-offs that these different versions bring are in terms of ciphertext storage, key storage (especially permutation key storage), type and number of operations, precision at the end of the operations and total multiplicative depth of the resulting circuit; these goals are intricately intertwined. For example, designing the circuit to have a lower multiplicative depth reduces the size of the ciphertexts and reduces the encryption load at the client, but might involve storing more ciphertexts and performing more computations at the server, compared to a version with a higher multiplicative depth. Thus, the main difficulty in making the computations tractable is to astutely batch the vectors and matrices into ciphertexts in order to reduce the memory, and manipulate the operations in order to reduce depth, number of operations and storage.

We describe here how to change the flow of operations in (7.2.20)–(7.2.23) in order to minimize the number of ciphertexts representing the relevant quantities: $H\mathbf{U}$, $H\mathbf{Y}$, $\mathbf{M}^{-1}$, $\mathbf{m}$, μ, $\bar{\mathbf{u}}$, $\bar{\mathbf{y}}$, $\mathbf{u}$, and the number of operations, while keeping the depth to a minimum. We make use of the feature of lattice-based homomorphic encryption schemes of encoding a vector of values into a single plaintext, which is then encrypted in a single ciphertext. Since we are also dealing with matrices, we explored several options of how to encode the matrices in ciphertexts: columns, rows, (hybrid) diagonals, vectorized matrix.

Note that the same linear algebraic operation, implemented for different encodings, leads to a different number of stored ciphertexts, multiplicative depth of circuit, and number of SIMD operations. For the problem we tackle here, we found that encoding each column of a matrix into a separate ciphertext minimizes the number of stored ciphertexts and number of operations, while keeping the same depth as when each element is encoded in a different ciphertext and the usual (unencrypted) method of performing the operations is used, as stated in Proposition 7.2.4. However, not all the matrices' columns are encoded in the same fashion, as we will see next. In summary, *having some redundancy inside the ciphertexts, i.e., having some values encoded multiple times, helps optimize the depth and complexity.*

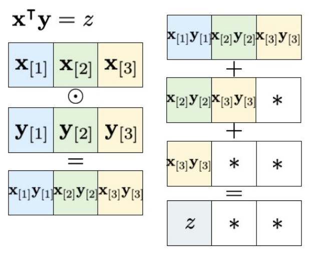
The following proposition summarizes the complexity of the online feedback algorithm, when carrying out the steps in this section. The proof of Proposition 7.2.4 is constructive and is given in the remaining of this subsection and Appendix F.2.

Proposition 7.2.4. *We evaluate the arithmetic circuit for the online feedback algorithm on encrypted data corresponding to Algorithm 7.2.1 for one time step with $O((S+t)^2)$ operations, $O(S+t)$ ciphertexts and $O(S+t)$ rotation keys, at depth $2t+4$, where S is the number of columns of the offline generated block-Hankel matrix and t is the number of online samples accumulated so far.*

Figure 7.2a shows how to obtain an inner product between two encoded vectors using SIMD multiplications, rotations and additions. Due to rotations, the resulting vector will have the relevant scalar in its first slot, and junk (partial sums) in the following slots. Figure 7.2b shows the most efficient way (in terms of required number of operations) to obtain the product between a matrix and a vector when the matrix is encoded as separate columns. Specifically, we need the elements of the vector to be repeated and separately encoded.

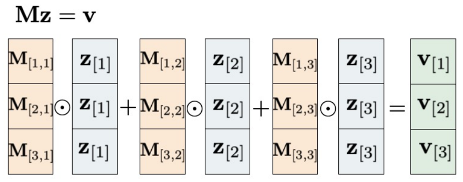
Once we have each column of $\mathbf{M}^{-1}$ encoded in a ciphertext, we want to also obtain the columns of $\mathbf{M}'^{-1}$ from (7.2.23) encoded each in a ciphertext. To minimize the multiplicative depth and number of operations, this suggests that we need to obtain $\mathbf{M}^{-1}\mathbf{m}^\intercal$ and $\mathbf{M}^{-1}\mathbf{m}^\intercal\mathbf{m}\mathbf{M}^{-1}$ encoded as columns.

Let us look first at $\mathbf{M}^{-1}\mathbf{m}^\intercal$. In order to use the efficient method outlined in Figure 7.2b, we need to have a separate ciphertext that encodes each element of $\mathbf{m}$, repeated as many times as the number of columns in $\mathbf{M}^{-1}$. In turn, this suggests that we should use the method outlined in Figure 7.2c when computing $\mathbf{m}$ as in (7.2.20), rather than the method outlined in Figure 7.2a which would require extra masking and rotations afterwards. Specifically, we



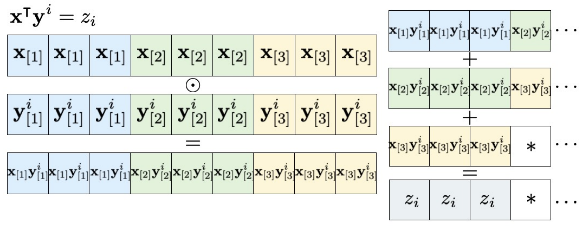
$\mathbf{x}^\intercal\mathbf{y} = z$
$\mathbf{x}_{[1]}$ $\mathbf{x}_{[2]}$ $\mathbf{x}_{[3]}$
$\odot$
$\mathbf{y}_{[1]}$ $\mathbf{y}_{[2]}$ $\mathbf{y}_{[3]}$
$=$
$\mathbf{x}_{[1]}\mathbf{y}_{[1]}$ $\mathbf{x}_{[2]}\mathbf{y}_{[2]}$ $\mathbf{x}_{[3]}\mathbf{y}_{[3]}$
$+$
$\mathbf{x}_{[2]}\mathbf{y}_{[2]}$ $\mathbf{x}_{[3]}\mathbf{y}_{[3]}$ $*$
$+$
$\mathbf{x}_{[3]}\mathbf{y}_{[3]}$ $*$ $*$
$=$
z $*$ $*$

(a) Inner product of two vectors.



$\mathbf{M}\mathbf{z} = \mathbf{v}$
$\mathbf{M}_{[1,1]}$ $\mathbf{M}_{[2,1]}$ $\mathbf{M}_{[3,1]}$ $\odot$ $\mathbf{z}_{[1]}$ $\mathbf{z}_{[1]}$ $\mathbf{z}_{[1]}$ $+$
$\mathbf{M}_{[1,2]}$ $\mathbf{M}_{[2,2]}$ $\mathbf{M}_{[3,2]}$ $\odot$ $\mathbf{z}_{[2]}$ $\mathbf{z}_{[2]}$ $\mathbf{z}_{[2]}$ $+$
$\mathbf{M}_{[1,3]}$ $\mathbf{M}_{[2,3]}$ $\mathbf{M}_{[3,3]}$ $\odot$ $\mathbf{z}_{[3]}$ $\mathbf{z}_{[3]}$ $\mathbf{z}_{[3]}$ $=$
$\mathbf{v}_{[1]}$ $\mathbf{v}_{[2]}$ $\mathbf{v}_{[3]}$

(b) Inner product of a matrix and a vector.



$\mathbf{x}^\intercal\mathbf{y}^i = z_i$
$\mathbf{x}_{[1]}$ $\mathbf{x}_{[1]}$ $\mathbf{x}_{[1]}$ $\mathbf{x}_{[2]}$ $\mathbf{x}_{[2]}$ $\mathbf{x}_{[2]}$ $\mathbf{x}_{[3]}$ $\mathbf{x}_{[3]}$ $\mathbf{x}_{[3]}$
$\odot$
$\mathbf{y}^i_{[1]}$ $\mathbf{y}^i_{[1]}$ $\mathbf{y}^i_{[1]}$ $\mathbf{y}^i_{[2]}$ $\mathbf{y}^i_{[2]}$ $\mathbf{y}^i_{[2]}$ $\mathbf{y}^i_{[3]}$ $\mathbf{y}^i_{[3]}$ $\mathbf{y}^i_{[3]}$
$=$
$\mathbf{x}_{[1]}\mathbf{y}^i_{[1]}$ $\mathbf{x}_{[1]}\mathbf{y}^i_{[1]}$ $\mathbf{x}_{[1]}\mathbf{y}^i_{[1]}$ $\mathbf{x}_{[2]}\mathbf{y}^i_{[2]}$ $\mathbf{x}_{[2]}\mathbf{y}^i_{[2]}$ $\mathbf{x}_{[2]}\mathbf{y}^i_{[2]}$ $\mathbf{x}_{[3]}\mathbf{y}^i_{[3]}$ $\mathbf{x}_{[3]}\mathbf{y}^i_{[3]}$ $\mathbf{x}_{[3]}\mathbf{y}^i_{[3]}$
$\mathbf{x}_{[1]}\mathbf{y}^i_{[1]}$ $\mathbf{x}_{[1]}\mathbf{y}^i_{[1]}$ $\mathbf{x}_{[1]}\mathbf{y}^i_{[1]}$ $\mathbf{x}_{[2]}\mathbf{y}^i_{[2]}$ $\cdots$
$+$
$\mathbf{x}_{[2]}\mathbf{y}^i_{[2]}$ $\mathbf{x}_{[2]}\mathbf{y}^i_{[2]}$ $\mathbf{x}_{[2]}\mathbf{y}^i_{[2]}$ $\mathbf{x}_{[3]}\mathbf{y}^i_{[3]}$ $\cdots$
$+$
$\mathbf{x}_{[3]}\mathbf{y}^i_{[3]}$ $\mathbf{x}_{[3]}\mathbf{y}^i_{[3]}$ $\mathbf{x}_{[3]}\mathbf{y}^i_{[3]}$ $*$ $\cdots$
$=$
z_i z_i z_i $*$ $\cdots$

(c) Inner product of two vectors with repeated result.

Figure 7.2: Inner product methods for encrypted data.

have to repeat each element of $h\mathbf{y}$ for S times, encode the resulting vector in a ciphertext and do the same for each column of $H\mathbf{Y}$ and the elements in the diagonal matrix $\mathbf{Q}$ and λ_y. Then, we perform an element-wise multiplication between the ciphertext encoding $h\mathbf{y}$ and the ciphertext encoding the elements of $\mathbf{Q}$ and λ_y, and then perform the inner product with the encoded columns of $H\mathbf{Y}$. The same steps are taken for $h\mathbf{u}, H\mathbf{U}, \mathbf{R}$ and λ_u.

Remark 7.2.5. To account for all the time steps where new samples will be collected, whenever we encode values repeatedly, we need to encode from the beginning $S+\bar{T}$ copies of the elements, where S is the initial number of columns in $\mathbf{M}$ and $\bar{T}$ is the total number of new samples intended to be collected.

One of the main realizations that makes this more efficient computation possible is how to compute each column $\mathrm{col}_i((\mathbf{m}\mathbf{M}^{-1})^\intercal(\mathbf{m}\mathbf{M}^{-1}))$ at a depth $d(\mathbf{M}^{-1})+2$ (when $d(\mathbf{M}^{-1}) \neq 0$), with $O(S)$ storage and $O(S^2)$ operations. Define $\tilde{\mathbf{M}} := (\mathbf{m}\mathbf{M}^{-1})^\intercal(\mathbf{m}\mathbf{M}^{-1})$. Then:

$\forall i = 0, \dots, S-1$:

$$\mathbf{z} := \sum_{k=0}^{S-1} \mathrm{col}_k(\mathbf{M}^{-1}) \odot \begin{bmatrix} \mathbf{m}_{[k]} \\ \vdots \\ \mathbf{m}_{[k]} \end{bmatrix}, \quad \mathbf{w}_i := \sum_{k=0}^{S-1} \mathrm{col}_k(\mathbf{M}^{-1}) \odot \left(\begin{bmatrix} \mathbf{m}_{[k]} \\ \vdots \\ \mathbf{m}_{[k]} \end{bmatrix} \odot \mathbf{e}_i \right),$$
$$\mathrm{col}_i(\tilde{\mathbf{M}}) = \mathbf{z} \odot \sum_{j=0}^{S-1} \rho\left(\mathbf{w}_i, i-j\right). \tag{7.2.25}$$

The next piece is to compute s as follows:

$$s = \mu - \sum_{i=0}^{S-1} \begin{bmatrix} \mathbf{m}_{[i]} & \dots & \mathbf{m}_{[i]} \end{bmatrix}^{\intercal} \odot \rho(\mathbf{M}^{-1}\mathbf{m}^{\intercal}, i).$$

The server asks the client to invert s. After the server receives from the client $1/s$, it can compute $\frac{1}{s}\tilde{\mathbf{M}}$, as required in (7.2.23), by replacing $\mathbf{e}_i$ in (7.2.25) by $1/s\,\mathbf{e}_i$.

Finally, we construct the columns of $\mathbf{M}'^{-1}$ by adding at the end of columns of $\frac{1}{s}\tilde{\mathbf{M}}$ the corresponding extracted elements of $-\frac{1}{s}\mathbf{M}^{-1}\mathbf{m}^{\intercal}$. The new column that expands the matrix is obtained by adding $1/s$ at the end of the column $-\frac{1}{s}\mathbf{M}^{-1}\mathbf{m}^{\intercal}$.

Regarding the computation of the control input $\mathbf{u}$, we again need to reorder the operations in order to have the smallest depth possible, i.e., $d(\mathbf{M}')+1$. For that, we also encode $\bar{\mathbf{u}}$ and $\bar{\mathbf{y}}$ with repeated elements, so we can compute the inner product as in Figure 7.2c, for the elements of $\mathbf{Z}$ in (7.2.24) stored repeatedly. We also prefer some redundancy in storage to avoid online processing, by having a ciphertext for the first m rows of $\mathbf{U}^{f'}$, encoded without repetition. This allows us to compute, for $i = 0, \dots, m-1$:

$$\boldsymbol{v}_i = \sum_{j=0}^{S} \sum_{k=0}^{S} \mathrm{col}_k \mathbf{M}'^{-1} \odot \left(\mathrm{row}_i \mathbf{U}^{f'} \odot \mathbf{Z}_{[j]} \right)$$
$$\mathbf{u}_{[i]} = (\mathbf{U}^{f'}\mathbf{M}'^{-1}\mathbf{Z})_{[i]} = \mathbf{1}^{\intercal}\boldsymbol{v}_i. \tag{7.2.26}$$

To send only one ciphertext for the result $\mathbf{u}$ (instead of m ciphertexts), while avoiding an extra masking, the server will pack the vectors $\boldsymbol{v}_i$ (which have an encoding with trailing zeros, whereas the encoding of $\mathbf{u}_{[i]}$ does not) into one ciphertext and ask the client to perform the corresponding summation.

The cloud service performs the encrypted version of (7.2.20)–(7.2.23), taking the steps outlined in this subsection, which we describe in detail in Appendix F.2 in equation (F.2.2)–(F.2.4), along with the specific encoding of each quantity.

After the $\bar{T}$ steps elapsed, we finish adapting and the server stores the encrypted values of $\mathbf{M}_t^{-1}, H\mathbf{U}_t, H\mathbf{Y}_t$. The server can store the encryptions of $\bar{\mathbf{y}}_t, \bar{\mathbf{u}}_t$ and update them at the subsequent time steps with the received encryptions of $\mathbf{y}_t, \mathbf{u}_t$, as done in the adaptation phase. However, we prefer the client to send $\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{y}}_t), \mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{u}}_t)$ instead of $\mathcal{E}_{\mathrm{vr0}}(\mathbf{y}_t), \mathcal{E}_{\mathrm{vr0}}(\mathbf{u}_t)$, in order to avoid future refreshing, see Remark F.2.1.

7.2.4.3 Precision discussion

Scaling. As described in Preamble 2.5.3, every encrypted operation introduces some noise that corrupts the least significant bits of the encrypted values, which accumulates with the depth of the circuit. This implies that values with a smaller magnitude are affected more by this noise. In the problem we address, the smallest values are found in $\mathbf{M}_t^{-1}$, hence scaling it by a positive integer factor α and accordingly changing the computations such that at every time step we obtain $\alpha\mathbf{M}_t^{-1}$ and $\alpha\mathbf{u}_t$ (such that the client can remove the scaling) allows us to increase the precision of the result. In order to incorporate this scaling without increasing the depth, the server has to perform more operations than before, e.g., recomputation from scratch of some quantities. The scaling factor should be selected by the client, and if its magnitude is sensitive, it can be sent as a ciphertext to the cloud server. We can modify the circuit such that this scaling does not affect the depth.

Precision loss when computing the Schur complement. Given the particularity of the matrix that has to be inverted, at each time step, some precision bits of the Schur complement are lost (depending on the regularization parameter and the magnitude of the measurements), which incurs a loss in precision in the subsequent computations.

Without the regularization term $\lambda_g\mathbf{I}$ added to the objective function of (7.2.6), the resulting matrix $\mathbf{M}$ does not have full rank in the noiseless case (noise helps, but the inverse is not numerically stable). This suggests that the Schur complement (7.2.22) will have the same order of magnitude as λ_g, regardless of the values in $\mathbf{M}$, which we show in the following. Let $\mathbf{H}^\intercal := \begin{bmatrix} H\mathbf{Y}^\intercal & H\mathbf{U}^\intercal \end{bmatrix}$ and $\mathbf{h}^\intercal := \begin{bmatrix} h\mathbf{y}^\intercal & h\mathbf{u}^\intercal \end{bmatrix}$ be a new column obtained by adding new samples, as described in Section 7.2.4.1, and $\mathbf{P} := \text{blkdiag}(\lambda_y\mathbf{I}, \mathbf{Q}, \lambda_u\mathbf{I}, \mathbf{R})$. Then,

$$s = \mu - \mathbf{m}\mathbf{M}^{-1}\mathbf{m}^\intercal = \lambda_g + \mathbf{h}^\intercal\mathbf{P}\mathbf{h} - \mathbf{h}^\intercal\mathbf{P}\mathbf{H}(\mathbf{H}^\intercal\mathbf{P}\mathbf{H} + \lambda_g\mathbf{I})^{-1}\mathbf{H}^\intercal\mathbf{P}\mathbf{h}. \tag{7.2.27}$$

We now show that, in the noiseless case, the Schur complement has the same order of magnitude as λ_g. Formally:

Lemma 7.2.6. *The following statements characterize the relation between s and λ_g:*

(a) $\lim_{\lambda_g \to 0} s = 0.$

(b) $\lim_{\lambda_g \to 0} \frac{s}{\lambda_g} = 1 + \mathbf{h}^\intercal\mathbf{P}^{1/2}(\mathbf{H}^\intercal\mathbf{P}^{1/2})^\dagger(\mathbf{P}^{1/2}\mathbf{H})^\dagger\mathbf{P}^{1/2}\mathbf{h}.$

(c) $\lim_{\lambda_g \to \infty} \frac{s}{\lambda_g} = 1.$

(d) The function $f(\lambda_g) = \frac{s}{\lambda_g}$ *is monotonously decreasing on* $[0, \infty)$.

The proof is given in Appendix F.3. The term in (b) $\mathbf{h}^\intercal\mathbf{P}^{1/2}(\mathbf{H}^\intercal\mathbf{P}^{1/2})^\dagger(\mathbf{P}^{1/2}\mathbf{H})^\dagger\mathbf{P}^{1/2}\mathbf{h}$ is small for slowly varying systems. Under small random noise, the results in Lemma 7.2.6 are not exact, but empirically follow closely.

This analysis is important from an encrypted implementation perspective. Lemma 7.2.6 shows that, for fixed cost values in $\mathbf{P}$ but regardless of the values in $\mathbf{H}$ and $\mathbf{h}$, i.e., the values of the $\mathbf{y}$ measurements and of the $\mathbf{u}$ control actions, s will be close to the regularization parameter λ_g. In the encrypted computations, we will compute $\mu = \lambda_g + \mathbf{h}^\intercal\mathbf{P}\mathbf{h}$ and $\mathbf{m}\mathbf{M}^{-1}\mathbf{m}^\intercal$ each with a fixed precision of x bits. For large values of the measurements, we obtain that $s \approx \lambda_g << \mathbf{h}^\intercal\mathbf{P}\mathbf{h}$, which means that there will be a cancellation in the most significant bits (e.g., the first y MSBs), leading to a loss of precision in s, which will now have only $x - y$ bits of precision. This is particularly crucial since such a cancellation of the MSBs will happen at every time step we accumulate new data samples, leading to a cascading loss of precision. Furthermore, due to using the Residue Number System implementation of the encryption scheme, which improves the efficiency by orders of magnitude by using native sizes of 64 bits to store the “residue ciphertexts” rather than multi-precision arithmetic of arbitrary size, the precision cannot be increased indefinitely (see [64, 117] for technical details).

As a side note, decreasing or increasing the values in the cost matrix $\mathbf{P}$ independently from λ_g does not solve this precision issue, because it implies changing the problem (it is equivalent to increasing or decreasing λ_g).

The conclusion of this discussion is that, apart from the role it has for regularization and noise robustness, the parameter λ_g also affects the precision of the solution (meaning the difference between the encrypted solution and the unencrypted solution), which suggests we need to pick λ_g to not be too small. This introduces a precision versus accurate convergence trade-off. We will comment more on this trade-off in Section 7.2.5.2.

7.2.4.4 Considerations for continuous running

Accumulating new samples serves to robustify the algorithm and adapt to new disturbances. However, an important caveat is that adding too many new samples damages the performance of the algorithm, both because of overfitting and because of the problem becoming intractable as the number of variables in (7.2.7) grows. In future work, we will investigate this issue more, along with having a sliding window of sample collection.

Nevertheless, there are cases where the number of new samples to be collected leads to a circuit of a depth higher than the preselected multiplication budget. Our options for continuing the computations are:

(i) Restore the initial precollected Hankel matrices which bypasses the refreshing step altogether. Advantages: no extra computations needed. Disadvantages: this causes oscillations in the control actions.

(ii) Stop adding new information to the matrices after the multiplication budget is exhausted. Advantages: no extra computations needed. Disadvantages: the multiplicative budget has to be large enough such that *enough* samples are collected.

(iii) Pack the matrix into a single ciphertext as described in (7.2.28) and ask the client to refresh it.

$$\mathcal{E}_{\mathrm{v}0}(\mathbf{M}_t^{-1}) = \sum_{i=0}^{S+t-1} \rho\left(\mathcal{E}_{\mathrm{v}0}(\mathrm{col}_i\mathbf{M}_t^{-1}), -i(S+t-1)\right). \tag{7.2.28}$$

Advantages: the server can continue collecting values for any desired time, without extra multiplication depth. Disadvantages: the client has to decrypt, encrypt and send another ciphertext; the rotation keys necessary for packing can occupy a lot of storage (but compared to [15], here we only need $S+t$ such rotations, not $(S+t)^2/2$).

(iv) Bootstrap the ciphertext of $\mathbf{M}_t^{-1}$. The computation advancements [60] regarding the bootstrapping procedure suggest that it is likely to locally resolve the refreshing step. Advantages: the server can continue collecting values for any desired time without the client's intervention. Disadvantages: the initial multiplication budget has to be larger to also allow for the bootstrapping circuit.

Increasing the multiplicative depth of the circuit also increases the ring dimension in order to ensure a fixed security level. Capping the maximum circuit depth (encouraging refreshing or bootstrapping) is desirable. In the solution we implemented, we chose option (iii). This gives us flexibility on the maximum multiplicative depth of the circuit, which will now be $2t_{refresh}+4$, at little extra cost for the client.

We can pack as many values as half the *ring dimension* ringDim in one ciphertext. Then, the number of ciphertexts the server can pack $\mathbf{M}_t^{-1}$ into is $\lceil (S+t)(S+t)/(\mathrm{ringDim}/2)\rceil$. This is viable since the ring dimension is in general large (e.g. ≥ 2048), in order to accommodate a reasonable multiplicative budget, plaintext precision and standard security parameter. The client only has to decrypt, re-encrypt and send back this number of ciphertexts. The server then uses one extra level to perform the reverse of (7.2.28) to unpack $\mathcal{E}_{\mathrm{v}0}(\mathbf{M}_t^{-1})$ into ciphertexts $\mathcal{E}_{\mathrm{v}0}(\mathrm{col}_i(\mathbf{M}_t^{-1}))$. These multiplications can be absorbed in the same initial multiplicative depth.

Theorem 7.2.7. *The encrypted online data-based predictive control algorithm in Section 7.2.4 achieves client privacy with respect to the server, cf. Definition 2.2.8.*

The proof is given in Appendix F.4.

Extensions

We now briefly describe extensions and possible modifications to the online algorithm.

7.2.4.5 Inputting blocks of control inputs

For simplicity of exposition, we illustrated the encrypted computations outline for when the server sends only the first component of $\mathbf{u}^{*,t}$ to the client. In practice, it is common that more consecutive components from the N computed control inputs are applied. This is also beneficial for the encrypted solution, since the most expensive computations, the inversion of $\mathbf{M}_t$ and its subsequent update, need to be computed only once for multiple time steps. Similarly, the communication rounds between the client and the server would be reduced. We recommend this for larger systems.

7.2.4.6 Batch collection of new samples

We can still use the Schur complement to compute the inverse of a rank-x update of the matrix $\mathbf{M}_t$, where x is how many new data samples we want to accumulate at once. The server then packs the matrix representing the Schur complement into one ciphertext and asks the client to invert it and send it back. The circuit depth will increase accordingly.

7.2.4.7 Sliding window of sample collection

In order to manage the growth in the number of variables, i.e., number of columns of $H\mathbf{U}_t$ and $H\mathbf{Y}_t$, after collecting a sample and updating $\mathbf{M}_t^{-1}$, we would like to remove the first sample collected. We can again achieve this by using Schur's complement and the Woodbury matrix inversion lemma. We revert to the notation used in Section 7.2.2.4 for simplicity of the exposition and we show how to remove the first sample, i.e., the first row and first column of $\mathbf{M}'$ (for removing rows and columns inside the matrix, we will have to multiply by permutation matrices). Given $\mathbf{M}'^{-1}$, we would like to extract $\mathbf{N}^{-1}$:

$$\mathbf{M}' = \begin{bmatrix} \mathbf{M} & \mathbf{m}^\intercal \\ \mathbf{m} & \mu \end{bmatrix} =: \begin{bmatrix} \nu & \mathbf{n} \\ \mathbf{n}^\intercal & \mathbf{N} \end{bmatrix}.$$

The Schur complements expressions for the two matrices are:

$$\mathbf{M}'/\mathbf{M} := \mu - \mathbf{m}\mathbf{M}^{-1}\mathbf{m}^\intercal = m_S, \quad \mathbf{M}'/\mathbf{N} := \nu - \mathbf{n}\mathbf{N}^{-1}\mathbf{n}^\intercal, \quad \mathbf{M}'/\nu := \mathbf{N} - \mathbf{n}^\intercal\nu^{-1}\mathbf{n}.$$

This gives the expression in (7.2.23) and the following equal expression for $(\mathbf{M}')^{-1}$:

$$\mathbf{M}'^{-1} = \begin{bmatrix} (\mathbf{M}'/\mathbf{N})^{-1} & -(\mathbf{M}'/\mathbf{N})^{-1}\mathbf{n}\mathbf{N}^{-1} \\ -\mathbf{N}^{-1}\mathbf{n}^\intercal(\mathbf{M}'/\mathbf{N})^{-1} & (\mathbf{M}'/\nu)^{-1} \end{bmatrix} =: \begin{bmatrix} l_1 & \mathbf{L}_2 \\ \mathbf{L}_2^\intercal & \mathbf{L}_3. \end{bmatrix}.$$

By the matrix inversion lemma,

$$\begin{aligned} &\mathbf{L}_3 = \mathbf{N}^{-1} + \mathbf{N}^{-1}\mathbf{n}^\intercal(\mathbf{M}'/\mathbf{N})^{-1}(\mathbf{M}'/\mathbf{N})(\mathbf{M}'/\mathbf{N})^{-1}\mathbf{n}\mathbf{N}^{-1} = \mathbf{N}^{-1} + \mathbf{L}_2^\intercal l_1^{-1}\mathbf{L}_2 \\ &\mathbf{N}^{-1} = \mathbf{L}_3 - \mathbf{L}_2^\intercal l_1^{-1}\mathbf{L}_2. \end{aligned}$$

Removing a sample after collecting one sample incurs an increase by two in the total depth of the computation and asking the client to perform the division of l_1.

7.2.4.8 Reducing accumulated noise

As discussed in Section 7.2.4.3, noise accumulates in the stored ciphertexts of $\mathbf{M}_t^{-1}$ at the server, because of the fixed precision imposed by the native dimension of 64 bits (in the Residue Number System implementation of the encryption scheme). However, before the noise grows too much, the server could try to keep track of it and reduce it by using the fact it can also easily compute $\mathbf{M}_t$ as in (7.2.18). Denote the error on $\mathbf{M}_t^{-1}$ by $\mathbf{X}$. Then:

$$\begin{aligned} \mathbf{\Lambda} &:= \mathbf{M}_t(\mathbf{M}_t^{-1} + \mathbf{X}) = \mathbf{I} + \mathbf{M}_t\mathbf{X} \\ (\mathbf{M}_t^{-1} &+ \mathbf{X})(\mathbf{\Lambda} - \mathbf{I}) = \mathbf{X} - \mathbf{X}\mathbf{M}_t\mathbf{X}. \end{aligned} \tag{7.2.29}$$

It is safe to assume $\|\mathbf{X}\|_\infty \leq \epsilon\|\mathbf{M}_t\|_\infty$, with $\epsilon < 1$. If it is true that ϵ satisfies the relation $\epsilon\|\mathbf{M}_t\|_\infty^2 \leq 1$, then $\|\mathbf{X}\mathbf{M}_t\mathbf{X}\|_\infty \leq \|\mathbf{X}\|_\infty$. This would mean that the server can compute a better approximation of $\mathbf{M}_t^{-1}$ than $\mathbf{M}_t^{-1} + \mathbf{X}$ as $\mathbf{M}_t^{-1} + \mathbf{X}\mathbf{M}_t\mathbf{X}$. However, this second approximation would imply a larger depth for $\mathbf{M}_t^{-1}$, so the best option is for the server to compute this approximation right before it asks the client to refresh the ciphertext.

7.2.4.9 Function privacy

The algorithm run at the CaaS service provider might be proprietary. In that case, the client should not get extra information about that algorithm, apart from what it agreed to receive. This notion is captured by the *function privacy* property of a homomorphic encryption scheme, which states that a ciphertext should not expose information about the function that was evaluated in order to create that ciphertext. Inherently, the CKKS scheme does not preserve function privacy because of the partial information stored in the junk elements. The common ways to achieve function privacy is for the server to mask out the irrelevant slots before handing a ciphertext to the client to decrypt, either by a multiplicative mask (zeroing out the junk elements) or by adding large enough randomness to them. The multiplicative mask increases the depth by one, whereas the randomness masking can lead to an increase in the plaintext modulus since the randomness has to be large enough to drown the relevant information.

In our scenario, the client is allowed to obtain the numerator and denominator of the control input at every time step and also the inverse matrix $\mathbf{M}^{-1}$, which is only determined from the inputs and outputs that the client knows. The server can zero out the extra slots in $\mathcal{E}_{\mathrm{vr}*}(s_t)$ and $\mathcal{E}_{\mathrm{v}*}(\boldsymbol{v}_t)$. This masking would add one extra level to the respective ciphertext, but the level increase would not build up, since these masks are not used in posterior computations. Further investigation on keeping the service provider's algorithm and costs private from the client will be subject to future work.

7.2.5 Implementation and evaluation

We use the version of the CKKS scheme [62], optimized to run on machine word size of 64-bit integer arithmetic [64, 117] instead of multiprecision integer arithmetic.

We focused on testing out our proposed algorithms on a temperature control problem, because first, smart buildings can support flexible encryption methods for the collected data, and second, the sampling time is of the order of minutes, which allows for complex computations and communication between the client and the server to take place.

7.2.5.1 Comparison with [15]

We first want to underline the improvements in both runtime and memory that the algorithm presented here brings compared to the algorithm in our previous work [15], that used a "hybrid" encoding and is briefly described in Appendix F.2. For the same parameters considered in the zone temperature problem exemplified in [15], which had one input and one output, used a ring dimension of 2^{12} and 21 levels in the ciphertext, we used the same commodity laptop with Intel Core i7, 8 GB of RAM and 8 virtual cores at 1.88 GHz frequency. We obtained an improvement of 2.4x in terms of memory: 2.25 GB compared to 5.48 GB. We obtained an improvement of 4x in terms of total running time, and specifically, the maximum runtime per step dropped from 308.8 s (previous algorithm) to 75 s (current algorithm). Some of the changes (e.g., in terms of better thread parallelization) we applied to our current algorithm can also be applied to the algorithm in [15], but the difference is made mostly by a more efficient encoding and computation reorganization. These improvements allowed us to simulate the encrypted control for larger systems with better security parameters, as described in the next parts.

7.2.5.2 2x2 system with 73 bits of security

Setup. We considered a temperature control problem for a building with two zones. The sampling time is of $T_s = 420$ s. The system parameters are: $n = 2, m = 2, p = 2, M = 4, N = 4, T = 32$ and the system is stable. To mimic a realistic example, we considered two types of disturbances: unknown, characterized by zero mean Gaussians for the process noise (covariance $0.001\mathbf{I}$) and for the measurement noise (covariance $0.01\mathbf{I}$), and known, slowly varying disturbances caused by the exterior temperature and the ground temperature, which we simulate as varying uniformly between $[24.5°, 25.5°]$ and $[9.5°, 10.5°]$, respectively. We choose the cost matrices and regularization terms $\mathbf{Q} = \mathbf{I}, \mathbf{R} = 10^{-3}\mathbf{I}, \lambda_g = 5, \lambda_y = \lambda_u = 1$.

For the offline data collection, we assume a different initial point than for the online computation, and randomly sample the offline input trajectory so that the corresponding output trajectory lies in the interval $[10°, 40°]$. The M, N and T parameters mean that we start from 25 columns in the Hankel matrices. We note that, due to noise, the offline feedback control algorithm has suboptimal performance for these parameters–see Figure 7.3. Thus, we collect data online for 15 more time steps, which means adding another 15 columns to the Hankel matrices. Afterwards, we run the system with fixed gains. Figure 7.3 shows the performance of the setpoint tracking with these parameters and under noise and disturbances. To make the comparison meaningful, we use the same noise sequence for both schemes, in both the encrypted and unencrypted runs of the algorithms.

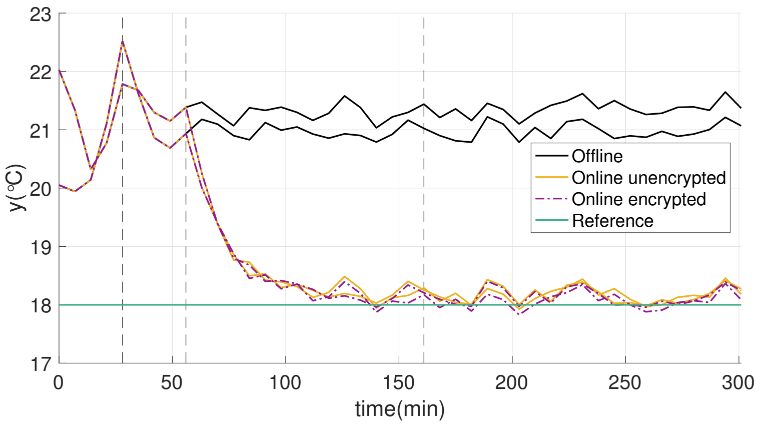


Figure 7.3: Tracking performance of the online versus offline feedback in the presence of noise. The first vertical dashed line marks the first M time steps, corresponding to the initial offline data, the second vertical dashed line marks the following N time steps, corresponding to the trajectory concatenation, and the last vertical dashed line marks the end of sample collection.

Implementation details. For better benchmarking capabilities (less variability between runs), for these experiments we used an AWS EC2 c5.2xlarge machine, with 8 virtual cores, 3.4GHz frequency and 16 GB of RAM. We implemented the encrypted solutions using the PALISADE library [179].

We use a ring dimension of 2^{14} and 15 levels (for a refresh step after 5 collected samples, corresponding to depth 14). The resulting ciphertext modulus of a fresh ciphertext is of 760 bits (the first plaintext modulus has 60 bits and the following plaintext moduli have 50 bits). This gives a security size of 73 bits, according to the LWE estimator [4]. Choosing a smaller refreshing time increases the communication rounds between the client and the server (one extra exchange at every 35 minutes in this case), but at the same time decreases the multiplicative depth of the circuit, reducing the computational complexity and the size of the parameters.

Precision. Figure 7.3 shows the tracking performance for the encrypted and unencrypted versions of the online and offline feedback algorithms. Due to the precision loss incurred by working with a deep circuit on encrypted data, we see a small offset in the measured output compared to the unecrypted measured output for the online feedback algorithm. Since the offline feedback algorithm is a very shallow circuit (depth 1) when the client returns a fresh encryption of the control input to the server, it incurs no precision loss.

We obtain a maximum magnitude of 0.129°C and an average of 0.045°C for the difference between the unencrypted measured output and the encrypted output during the online simulation of a noisy process. For the control input, the maximum difference is 0.248 kWatts and an average difference of 0.053 kWatts. A typical such sample is depicted in Figure 7.3. Notice that the maximum magnitude of the difference is smaller than the variation in the disturbances and does not affect the performance and stability of the control algorithm. One can increase the plaintext modulus in order to reduce the error introduced by encryption (e.g., 53 bits instead of 50 bits in the moduli will lead to an almost tenfold decrease in the differences). However, if we keep the ring dimension constant, this reduces the security level to 69 bits. In Section 7.2.5.3, we show results for better precision at a standard security level. As described in Section 7.2.4.3, the loss of precision can be alleviated by a larger regularization parameter λ_g. For example, keeping all other parameters the same but increasing $\lambda_g = 10$ will lead to a maximum magnitude of the difference in the measurements of 0.037° and the maximum magnitude of the difference in the inputs of 0.043 kWatts, but the tracking performance is slightly reduced (from 0.06°C to 0.12°C deviation from the reference in the noiseless case).

Offline. The encrypted offline feedback algorithm is very fast, since it has only depth 1 when the client sends to the server encryptions of $\bar{\mathbf{u}}_t$ and $\bar{\mathbf{y}}_t$. A ring dimension of 2^{12} gives a security parameter of 126 bits, and a ciphertext has 110 KB. The peak online RAM is 42 MB. The average runtime for a time step is 57 ms for the server and 12 ms for the client.

Online. We now examine the encrypted online control algorithm simulation adding details that complement the information in Section 7.2.4.

Online - Memory. The peak RAM for both the client and the server incurred during the three phases for 45 time steps was 6.26 GB, out of which, 3.9 MB is the public key, 2 MB is the secret key, 15.7 MB is the relinearization key and 4.6 GB are the evaluation rotation keys (corresponding to a key for each of the 336 rotation indices), generated offline. The client discards the evaluation keys after it sends them to the server. At every iteration, the client receives two ciphertexts from the server (the Schur complement and control action) and sends three ciphertexts (inverse of the Schur complement, measurement, control action– we assumed here that the setpoint is constant). During the refresh time steps, one extra ciphertext is sent and received (the inverse matrix). The size of these ciphertexts depends on the level where the ciphertext is at. Specifically, for our selected parameters described in the beginning of the subsection, the ciphertexts are 2.77 MB at maximum size and 240 KB at the minimum size.

Online - Time. The runtimes for the encrypted computation are given in Figure 7.4, where three different phases are depicted: trajectory concatenation, online sample collection and static update. The total offline initialization time for key generation and ciphertext encryption is 60 s for both the client and the server. In all online phases, the client only performs cheap and fast computations that take less that 0.3 s.

The first online phase is the trajectory concatenation phase from $t = M - 1 = 3$ to $t = M + N - 1 = 7$, where the server computes the encrypted control action only with the precollected data, as described in Section 7.2.2.3. This is very efficient, despite computing on the maximum ciphertext sizes, and results in a total computation time for one time step of 2 s. Before $t = M - 1$, the client applies random inputs.

The second phase is the online collection of new input-output samples, which implies modifying the Hankel matrices and computing the inverse matrix via the Schur complement at every time step $\mathbf{M}_t^{-1}$ from step 8 to step 22. This phase is split in as many parts as the refreshing time dictates. In the simulation depicted in Figure 7.4, this corresponds to

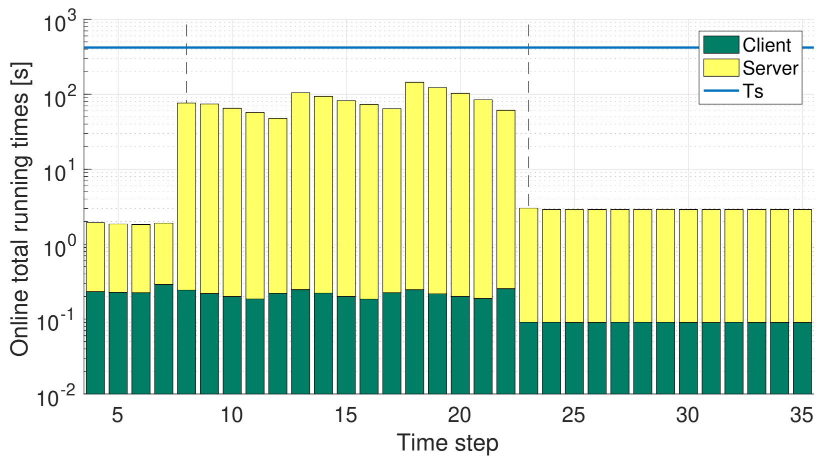


Figure 7.4: Running times for the computations performed at the client and the cloud server for the encrypted online control algorithm. The plot is semi-logarithmic and the amounts of time required by the client and the cloud server are stacked. The first M time steps are not depicted, because the server has no computational load. The first vertical dashed line marks the following N time steps, corresponding to the trajectory concatenation, and the second vertical dashed line marks the end of sample collection.

three parts. At the established refresh times ($t = 12, 17$), the server packs the matrix $\mathbf{M}_t^{-1}$ into one ciphertext, sends it to the client to re-encrypt it with the maximum number of moduli, and unpacks it back into component-wise ciphertexts. First, the increase in runtime between the subsequent time steps 13 and 12, respectively 18 and 17, is given by the fact the ciphertexts are returned to the maximum number of moduli. Second, the increase in computation time from the first refresh at time 12 and 13 (47.23 s and 104.6 s) to the second refresh at time 17 and 18 (63.9 s and 144.2 s) is given by the fact that the server has to deal with more collected samples ciphertexts than in the beginning of phase two. The intermediate decrease in the computation time is given by the decrease in the ciphertext size (we make sure to compute with only the minimal number of moduli required).

The third phase corresponds to the computations after stopping the collection of new samples, which starts from time step 23 and can go for the rest of the desired simulation time (accounting for Remark F.2.1 in Appendix F.2). The third phase is only slightly more computationally intensive than the first. The time for the client halves compared to the previous two phases, because the client has to decrypt and encrypt ciphertexts with two moduli. The running time for the server substantially decreases compared to phase two but doubles compared to phase one, because we use a less efficient matrix-vector multiplications in order to minimize the multiplicative depth. Nevertheless, the running time required for computing the control input at one time step is around 3 s.

7.2.5.3 More security

In practice, the desired security parameter for a cryptographic application is at least 100 bits. The refresh time after 5 time steps (35 minutes) gives a ciphertext modulus of 802 bits and the refresh time after 8 time steps (56 minutes), which means a deeper circuit, gives a ciphertext modulus of 1120 bits. In Table 7.1, we present simulation results where we choose the ring dimension 2^{15} such that we satisfy this security requirement for the chosen ciphertext modulus. We simulate for the above 2x2 system and for a 4x4 system (where $M = 4, N = 4, T = 64$ and we again collect 15 more online samples, which gives a maximum number of 72 unknowns). We use 53 bits of precision for the plaintext moduli, such that we obtain 0.034°C and 0.008°C maximum, respectively average difference between the encrypted and unencrypted measurements, and 0.05 kWatts and 0.008 kWatts maximum, respectively average difference for the input.

We use different AWS cloud machines in order to satisfy the RAM and sampling time requirements. The c5.2xlarge machine (16 GB RAM) used 8 threads, the c5.4xlarge machine (32 GB RAM) used 13 threads and the c5.9xlarge machine (72 GB RAM) used 28 threads.

In Table 7.1 we see the trade-off between a longer refresh time (the client refreshes a ciphertext after more steps, which implies a deeper circuit and less security for the same ring dimension) and the memory consumption and computation time. The conclusion is that *the refresh time is a tuning knob between security level, communication and computational complexity which should be determined according to the application's specifications.*

Note that as the performance of the underlying homomorphic encryption library improves, these improvements will reflect in our computation times as well.

Experiment id.	System dim.	# of steps until refresh	Security level [bits]	Ciphertext modulus	Max. RAM [GB]	Server machine
1	2×2	8	100	1120	17.19	c5.4xlarge
2	2×2	5	142	802	12.57	c5.2xlarge
3	4×4	8	100	1120	34.32	c5.9xlarge
4	4×4	5	142	802	27.29	c5.9xlarge

Experiment id.	Max. runtime phase I [s]		Max. runtime phase II [s]		Max. runtime phase III [s]	
	Client	Server	Client	Server	Client	Server
1	0.88	3.48	0.84	220.2	0.23	4.36
2	0.6	3.23	0.5	298.1	0.17	6.21
3	0.88	4.06	0.84	413.07	0.23	4.94
4	0.6	2.95	0.5	325.72	0.17	4.42

Table 7.1: Simulation results for four experiments of the encrypted online feedback algorithm on systems of two different sizes, at different security levels and different refresh times. In all entries, the ring dimension is 2^{15}, to ensure a security level of at least 100 bits. The online times for the client were obtained from a commodity laptop-like machine, c5.2xlarge.

7.3 Sparse data-based predictive control on encrypted data

In Section 7.2.2, we used an ℓ_2-regularization for the preimage in (7.2.5). We made this choice for two reasons: first, it is an accepted and used solution to reduce overfitting due to noisy data [35, 125], and second, it gives us a ridge regression formulation, that has a closed-form solution, making it appealing from an encrypted implementation perspective.

On the other hand, ℓ_1-regularization is also preferred in the data-based predicted control literature [70, 71, 83], thanks to a motivation coming from direct system identification techniques, and in a hybrid regularization setting, where it is used alongside ℓ_2. More generally, ℓ_1-regularization is widely employed for sparse learning, see [123]. Hence, in this section, we investigate a private solution to the sparse data-based predictive control.

To summarize, in order to avoid overfitting due to noisy data, we penalize the magnitude of $\mathbf{g}$ through an ℓ_1-regularization with penalty parameter λ_g in (7.3.1). The intuition behind this choice comes from the fact that in the noiseless data predictive control formulation, the block-Hankel matrix of the trajectory data has an inherent low-rank structure. Choosing an ℓ_1-regularization acts like a convex relaxation of imposing a low-rank constraint–see [83, Thm. 4.6] for more details.

$$\min_{\mathbf{g}} \frac{1}{2}\left(\|\mathbf{Y}^f\mathbf{g}-\mathbf{r}_t\|^2_{\mathbf{Q}}+\|\mathbf{U}^f\mathbf{g}\|^2_{\mathbf{R}}\right)+\lambda_y\|\mathbf{Y}^p\mathbf{g}-\bar{\mathbf{y}}_t\|_2^2+\lambda_u\|\mathbf{U}^p\mathbf{g}-\bar{\mathbf{u}}_t\|_2^2+\lambda_g\|\mathbf{g}\|_1. \quad (7.3.1)$$

Notice that (7.3.1) is a Lasso problem:

$$\min_{\mathbf{g}} \frac{1}{2}\|\mathbf{H}\mathbf{g}-\mathbf{J}\mathbf{f}_t\|_2^2+\lambda_g\|\mathbf{g}\|_1, \quad (7.3.2)$$

with $\mathbf{J} := \mathrm{blkdiag}\left(2\lambda_y\mathbf{I},\, \mathbf{Q},\, 2\lambda_u\mathbf{I},\, \mathbf{R}\right)^{1/2}, \mathbf{f}_t := \left[\bar{\mathbf{y}}_t^\intercal\, \mathbf{r}_t^\intercal\, \bar{\mathbf{u}}_t^\intercal\, \mathbf{0}^\intercal\right]^\intercal, \mathbf{H} := \mathbf{J}\left[\mathbf{Y}^{p\intercal}\, \mathbf{Y}^{f\intercal}\, \mathbf{U}^{p\intercal}\, \mathbf{U}^{f\intercal}\right]^\intercal$. In the case a hybrid regularization $\lambda_g\|\mathbf{g}\|_1+\mu_g\|\mathbf{g}\|_2^2$ is preferred, we can use the same formulation (7.3.2) and appropriately modify $\mathbf{H}$ and $\mathbf{f}_t$.

Our goal is to provide a solution that outsources to a cloud service the computation of the optimal solution $\mathbf{g}^{*,t}$ of (7.3.2) and of $\mathbf{u}^{*,t} = \mathbf{U}^f\mathbf{g}^{*,t}$, while ensuring client data confidentiality for all time steps, as described in Section 7.2.1.

In the rest of this chapter, we apply our cloud-based sparsity framework for encrypted Lasso problems, developed in Chapter 3.3, to the problem of data-based predictive control. This framework requires multiple servers for both efficiency and security reasons, compared to the solution in the first part of the chapter, that used a single server (but asked the client for refreshing). To alleviate the cost, we will present a solution that involves only one powerful server, and a few computationally-weaker devices.

7.3.1 Encrypted sparse regularized data predictive control

Following the discussion in Section 3.3.3, we write the problem (7.3.2) as a distributed problem with split variables, in order to use a distributed ADMM algorithm:

$$\begin{aligned}&\min_{\mathbf{g}_1,\ldots,\mathbf{g}_K,\mathbf{z}} \frac{1}{2}\sum_{i=1}^{K}\|\mathbf{H}_i\mathbf{g}_i-(\mathbf{J}\mathbf{f})_i\|_2^2+\lambda_g\|\mathbf{z}\|_1\\ &\text{s.t.}\;\; \mathbf{g}_i-\mathbf{z}=\mathbf{0}, \quad i=1,2,\ldots,K.\end{aligned} \quad (7.3.3)$$

In the context of this data-predictive problem, when we perform a homogenous split of the data (a split of equal size), the distributed solution converges very slowly to the global optimal solution. The reason for that is that the homogeneous sub-problems have a different optimal solution than the global solution. To gain intuition, consider splitting the component matrices $\mathbf{U}^p, \mathbf{U}^f, \mathbf{Y}^p, \mathbf{Y}^f$ of $\mathbf{H}$ equally between the K servers. This means that each server solves a local optimization problem for the same system (7.2.1) that generated the values, but being given fewer samples than necessary to characterize the behaviour of the system, i.e., losing persistency of excitation (Assumption 7.2.1). The problem remains when allocating a random set of rows of the equal size to the servers.

To avoid this issue, we prefer to unequally split the problem. Specifically, we designate Server 1 to have most of the rows and the rest of the servers to hold fewer. Because the local solution of Server 1 is close to the central solution, the empirical convergence to the optimal solution is much faster. However, the more servers we add, the slower the convergence (if we do not weight contributions differently). A valid option is to have only one server do all the computation, i.e., central ADMM, and request help only for the distributed bootstrapping from the rest of the servers (recall that we require multiple servers both for an efficient distributed bootstrapping and for security of the private key). But since the rest of the servers would be idle while the central server performs the computation, we prefer to distribute some of the computations to them as well.

Let the matrix $\mathbf{H}_1$ denote the first $(m+p)M + pN$ rows of matrix $\mathbf{H} \in \mathbb{R}^{(m+p)(N+M)\times S}$. We split the remaining rows of $\mathbf{H}$ into blocks of $mN/(K-1)$ rows, denoted $\mathbf{H}_i$ for $i = 2, \dots, K$. We similarly split $\mathbf{H}^\intercal \mathbf{J} \in \mathbb{R}^{S\times(m+p)(N+M)}$ into $\bar{\mathbf{H}}_1$ and $\bar{\mathbf{H}}_2 \dots, \bar{\mathbf{H}}_K$ and $\mathbf{f}_t \in \mathbb{R}^m$ into $\mathbf{f}_{1,t}$ and $\mathbf{f}_{2,t}, \dots, \mathbf{f}_{K,t}$. We prefer to use more of less powerful devices in order to increase the security threshold (by splitting the secret key into more values) and reduce the cost of operating the cloud service. To this end, we shift some of the computations from the less powerful servers to the more powerful Server 1 and remove online communication between the client and the less powerful servers. Protocol 7.3.1 differs from Protocol 3.3.1 in this different allocation of computation, described below.

First, we split problem (7.3.3) such that $\mathbf{f}_{i,t} = 0$, for $i = 2, \dots, K$, see (7.3.4). Second, Servers $2, \dots, K$ have an easier offline computation, since they have to invert substantially smaller matrices than Server 1, using the matrix inversion lemma. The bootstrapping step is done the same as in Protocol 3.3.1, after all parties broadcast their local sums. However, we let only the more powerful Server 1 perform the summation $\sum_{i=1}^{K} \mathbf{g}_i^{k+1} + \mathbf{w}_i^k$ and the evaluation of the soft thresholding approximation, and then send the result $\mathbf{z}^{k+1}$ to the other less powerful servers (the ciphertext $\mathbf{z}^{k+1}$ will have only $l_B + 1$ levels so communication is cheap; recall l_B is the number of levels necessary for a statistically secure distributed bootstrapping). Then, all servers continue with the computation of $\mathbf{w}_i^{k+1}$ and finish the iteration.

$$\begin{aligned}
\mathbf{g}_1^{k+1} &= (\mathbf{H}_1^\intercal \mathbf{H}_1 + \rho \mathbf{I})^{-1} \left(\bar{\mathbf{H}}_1^\intercal \mathbf{f}_1 + \rho(\mathbf{z}^k - \mathbf{w}_1^k) \right) \\
\mathbf{g}_i^{k+1} &= \rho \, (\mathbf{H}_i^\intercal \mathbf{H}_i + \rho \mathbf{I})^{-1} \, (\mathbf{z}^k - \mathbf{w}_i^k), \ i = 2, \dots, K \\
\mathbf{z}^{k+1} &= \frac{1}{K} S_{\lambda_g/\rho} \left(\sum_{i=1}^{K} \mathbf{g}_i^{k+1} + \sum_{i=1}^{K} \mathbf{w}_i^k \right) \\
\mathbf{w}_i^{k+1} &= \mathbf{w}_i^k + \mathbf{g}_i^{k+1} - \mathbf{z}^{k+1}, \ i = 1, \dots, K.
\end{aligned} \tag{7.3.4}$$

Moreover, because of the way we split the time-varying vector $\mathbf{f}_t$, such that the elements corresponding to Servers $2, \dots, K$ are 0, there is no need for them to update with the latest values of $\mathbf{u}_t$ and $\mathbf{y}_t$. This way, only Server 1 needs to have a connection with the client. The ciphertexts communicated to the client are on level 0 (the predicted input $\mathbf{u}^{*,t}$), while the ciphertexts communicated from the client ($\mathbf{u}_t$ and $\mathbf{y}_t$ for assembling $\mathbf{f}_{1,t}$) are on level $l_B + 2$.

Nevertheless, if the servers have different capacity, the more powerful server will likely have to wait on the other servers for the bootstrapping synchronization (requiring completion of the computation for $\mathbf{g}_i^{k+1}$). In the idle time, the more powerful Server 1 can perform multiple local updates, which heuristically helps with convergence in our problem.

Theorem 7.3.1. *Protocol 7.3.1 achieves client data confidentiality with respect to semi-honest servers, assuming at least one of the servers is honest.*

The proof follows from the proof of Proposition 3.3.4, regardless of having the servers perform different tasks, since all tasks involve computations only on encrypted data.

Protocol 7.3.1: Distributed encrypted protocol for (7.3.3) with *unequal servers* and *unequal data split* for one time step t

Input: Public parameters: public key pk, parameters of the system and offline trajectory m, p, N, M, S, the number of servers K, number of maximum iterations K_{iter}. C: $(\mathbf{u}_\tau, \mathbf{y}_\tau)_{\tau=0,\dots,t}$. S_1: encryption of $\mathbf{M}_1 = \rho(\mathbf{H}_1^\intercal \mathbf{H}_1 + \rho \mathbf{I})^{-1}$, encryption of $\mathbf{F}_1 = \frac{1}{\rho}(\mathbf{M}_1 \mathbf{H}_1^\intercal \mathbf{J})$, encryption of $\mathbf{U}^f$, encryption of $\bar{\mathbf{y}}_t, \mathbf{r}_t, \mathbf{u}_t$, share of the secret key sk_1, the Chebyshev coefficients for evaluating the soft threshold function for a given interval and bias λ_g/ρ. $S_2, \dots, S_K$: encryption of $\mathbf{M}_i = \rho(\mathbf{H}_i^\intercal \mathbf{H}_i + \rho \mathbf{I})^{-1}$, share of the secret key sk_i, for $i = 2, \dots, K$.

Output: C: $\mathbf{u}_{t+1}$

```
1: $C$: send to $S_1$ the ciphertexts $\mathcal{E}_{\text{v0}}(\mathbf{u}_t)$, $\mathcal{E}_{\text{v0}}(\mathbf{y}_t)$, $\mathcal{E}_{\text{v0}}(\mathbf{r}_t)$;
2: $S_1$: assemble the ciphertext $\mathcal{E}_{\text{v0}}(\mathbf{f}_{1,t}) = \mathcal{E}_{\text{v0}}([\bar{\mathbf{y}}_t^\intercal\ \mathbf{r}_t^\intercal\ \bar{\mathbf{u}}_t^\intercal]^\intercal)$;
3: $S_{i=1,\dots,K}$: set initial values $\mathcal{E}_{\text{v0}}(\mathbf{g}_i^0)$, $\mathcal{E}_{\text{v0}}(\mathbf{w}_i^0)$, $\mathcal{E}_{\text{v0}}(\mathbf{z}^0)$ (the value of $\mathbf{z}^k$ is previously agreed upon);
4: **for** $k = 0, \dots, K_{\text{iter}} - 1$ **do**
5:     $S_1$: compute $\mathcal{E}_{\text{v0}}(\mathbf{g}_1^k) = \text{MultDiag}(\mathbf{F}_1, \mathbf{f}_{1,t}) + \text{MultDiag}(\mathbf{M}_1, \mathbf{z}^k - \mathbf{w}_1^k)$;
6:     $S_{i=2,\dots,K}$: compute $\mathcal{E}_{\text{v}*}(\mathbf{g}_i^k) = \text{MultDiag}(\mathbf{M}_i, \mathbf{z}^k - \mathbf{w}_i^k)$;
7:     $S_{i=1,\dots,K}$: compute and send to the other servers the rotation of the sum $\mathcal{E}_{\text{v0}}(\mathbf{v}_i) := \rho(\mathbf{g}_i^{k+1} + \mathbf{w}_i^k, -(i-1)S)$;
8:     $S_{i=1,\dots,K}$: assemble $\mathcal{E}_{\text{v}*}(\mathbf{v}) := \mathcal{E}_{\text{v}*}([\mathbf{v}_1\, \mathbf{v}_2 \dots \mathbf{v}_K])$ by summing own ciphertext and all received shifted ciphertexts;
9:     $S_{i=1,\dots,K}$: perform part in the distributed boostratpping to get $\mathcal{E}_{\text{v}*}(\mathbf{v}^b) := \text{DBoot}(\mathcal{E}_{\text{v}*}(\mathbf{v}))$;
10:    $S_{i=1,\dots,K}$: extract its refreshed sum of local iterates $\mathcal{E}_{\text{v}*}(\mathbf{v}_i) = \rho(\mathcal{E}_{\text{v}*}(\mathbf{v}), (i-1)S)$;
11:    $S_1$: rotate and sum the refreshed $\mathcal{E}_{\text{v}*}(\mathbf{v}^b)$ to obtain $\mathcal{E}_{\text{v}*}(\sum_{i=1}^K \mathbf{g}_i^{k+1} + \mathbf{w}_i^k)$, then compute $\mathcal{E}_{\text{v0}}(\mathbf{z}^k) = \text{EvalApproxSoftT}(\frac{1}{K}\sum_{i=1}^K \mathbf{g}_i^{k+1} + \mathbf{w}_i^k, \lambda_g/\rho)$;
12:    $S_1$: send to all the other servers $\mathcal{E}_{\text{v0}}(\mathbf{z}^k)$;
13:    $S_{i=1,\dots,K}$: compute $\mathcal{E}_{\text{v0}}(\mathbf{w}_i^{k+1}) = [\mathbf{1}_S^\intercal\, \mathbf{0}^\intercal]^\intercal \odot \mathcal{E}_{\text{v}*}(\mathbf{x}_i) - \mathcal{E}_{\text{v0}}(\mathbf{z}^k)$;
14: **end for**
15: $S_1$: compute $\mathcal{E}_{\text{v0}}(\mathbf{u}^*) = \text{MultDiag}(\mathbf{U}^f, \mathbf{g}_1^K)$ and send the result to the client $C$;    ▷ or directly obtain only the first $m$ components by multiplying by $[\mathbf{I}_m\ \mathbf{0}]\mathbf{U}^f$
16: $C$: decrypt $\mathbf{u}_{t+1}$, plug it in the system to measure $\mathbf{y}_{t+1}$.
```

Remark 7.3.2. An interesting remark is related to [134], where the authors propose to use multiple controllers in parallel that perform asynchronous local bootstrapping to ensure that at least one has a control input ready at each time step. In our case, we prefer multiple servers to perform a distributed bootstrapping in order to reduce the time it takes to refresh the ciphertexts.

7.3.2 Numerical results

We consider a data-driven temperature control of a 4x4 stable system representing a building with four zones, with sampling time 300 seconds, and $M = 4$, $N = 8$, $T = 84$. We add process noise and measurement noise, both zero mean Gaussian with covariance $0.01\mathbf{I}$. We choose the cost matrices and regularization terms $\mathbf{Q} = 300\mathbf{I}, \mathbf{R} = \mathbf{I}, \lambda_g = 300, \lambda_y = \lambda_u = 3000$. The data was distributed among 3 servers: Server 1 holds 64 rows and Servers 2 and 3 hold 16 rows each. Convergence for the Lasso problem associated to one time step of the above problem occurred after 20 ADMM iterations, for $\rho = 1200$. Figure 7.5 reflects the tracking performance of the data predictive control problem with these parameters.

We evaluate Protocol 7.3.1 on Ubuntu 18.04 on a commodity laptop with 8 GB of RAM and Intel Core i7, 1.88 GHz, implemented using the PALISADE library [178], using 8 threads. We set the parameters such that we get a security level of 128 bits, i.e., we use a ciphertext modulus of 436 bits and a ring dimension of 2^{14}. We obtain 6 decimal places precision for the results. The average time for the first iteration is 2.75 seconds and for any iteration afterwards is 1.98 seconds. The time for Server 1 to assemble the vector $\mathbf{f}_{1,t}$ from the client and to compute the prediction is 0.61 seconds. The client needs 0.07 seconds to decrypt the control input and to encrypt the new measurement and the input. This gives a total computation time of 38.44 seconds per solving the optimization problem, not taking communication into account. The setup takes 4.5 seconds, and is performed once for all subsequent iterations.

If we artificially add a 150 ms delay of communication (serialization/deserialization and transport) and assume the machines send the messages sequentially to the other machines, then the total computation time increases by 6.45 seconds (450 ms delay per iteration, and

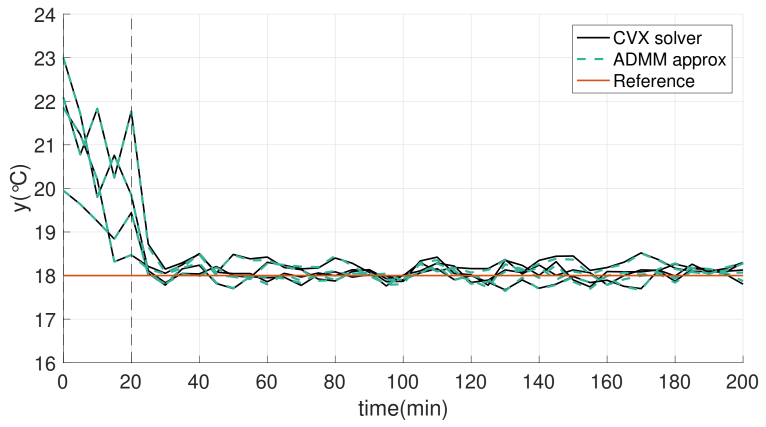


Figure 7.5: Comparison between the tracking performance of the data predictive controller solving (7.3.2) exactly via the CVX solver, and solving (7.3.2) via distributed ADMM with 3 servers and approximating the soft thresholding function with a degree-11 polynomial. The vertical dashed line marks the first M time steps, corresponding to the initial offline data. The curves represent the temperature measurements in the four rooms of the system.

450 ms delay for communication between Server 1 and Client).

Overall, the maximum amount of memory Server 1 needs to have is 1.22 GB, while Server 2 and 3 need 0.52 GB.

To simulate less powerful devices, we run Servers 2 and 3 on 2 threads instead of 8. The total time necessary for the 20 iterations increases to 49.96 seconds. The majority of the difference comes from the final operation of bootstrapping (which can be done asynchronously, i.e., it occurs after the communication so servers do not have to wait for each other): the total bootstrapping time increases from 0.91 seconds to 1.48 seconds. The rest of the difference comes from the fact that computing $\mathbf{g}_2^{k+1}$ and $\mathbf{g}_3^{k+1}$ takes 1.06 seconds compared to the 0.83 seconds that Server 1 needs to compute $\mathbf{g}_1^{k+1}$.

Because the servers only need to synchronize in order to bootstrap (Server 2 and 3 also wait for $\mathbf{z}^{k+1}$ from Server 1, but in our instance this does not create idle time since Server 1 is more powerful), Server 1 can either wait for the other servers to finish the computation or can perform two local updates of $\mathbf{g}_1$. Heuristically, since the local solution of Server 1 is closer to the global optimal solution, performing more local iterations helps convergence (in our example, by needing 18 iterations instead of 20). Nevertheless, the computation of the control input is still ready in one sixth of the sampling time.

Chapter 8

Conclusion

Encrypted control is a young research area. In this thesis, I focused on fundamental control algorithms and on providing proof of concept solutions that achieve both privacy and efficiency. In a sense, these solutions are only scratching the surface. There are many remaining challenges ranging from the structured nature of the data in control and optimization problems, to the stability and performance requirements that are potentially disrupted under encryption, to the possibility of malicious attacks over the controller, the sensors or the actuator, and to scaling the solutions up to large systems, both in terms of data size and in terms of number of participating actors. In the following, I will outline some of the future research areas that complement the problems and solutions covered in this thesis. We have collected more research directions in the tutorial on encrypted control in [200].

Parameter estimation for encrypted experiments

In this work, as well as in most of encrypted protocols for control and optimization, we assumed knowledge of some bounds on the variables, which can themselves leak information. In fact, an overarching problem when dealing with algorithms is figuring out the optimal parameters for a correct and efficient execution: e.g., step size, number of iterations, normalization, ranges etc. In some cases, it is reasonable to consider that all the algorithms have been run on cleartext data in advance, so the parameters were determined with confidence. In other cases, proprietary algorithms and proprietary data can be involved in all steps of the process, including what we usually call "the offline phase". In such scenarios, one can still make use of encrypted solutions to obliviously run the algorithms on data and cross-validate.

An important avenue of research is to design encrypted and differentially private prior experiments in order to compute differentially private bounds on parameters such as costs, penalties and number of iterations. The loss in accuracy should not matter for a conservative choice of parameters. Further, when a company desires to tune its algorithm on representative offline data from the client, we recommend an approach where the key is shared, as described in Chapters 3.3 and 7.3. We can involve the client in these experiments, by having it control a small device with one share of the secret key. In this way, the client can assess whether the query is too revealing and has to agree on the computation that is being effectuated to let the experiment continue and decrypt the result.

More general aggregation

In Chapter 4, we discussed an elementary type of aggregation, specifically weighted affine combinations of private local data, with private weights. However, there are interesting applications involving more complex aggregation schemes and it is worthwhile to explore the efficiency of private nonlinear aggregation schemes in the future. For example, the underlying leveled homomorphic encryption scheme used in Section 4.5 can support more complex operations, at the cost of larger parameters. More specifically, we could perform computations such as:

$$\mathbf{x}_a(t) = \sum_{i\in[M]} \varphi\left(\mathbf{W}_i \mathbf{x}_i(t)\right),$$

for a nonlinear function $\varphi(\cdot)$.

Beyond passive adversaries

My previous work assumed passive adversaries, i.e., adversaries that do not tamper with the data, but try to infer information from it. Many of the agents and IoT devices are mobile and autonomous and are soon intended to be deployed as delivery, exploration, information-providing and first-responder agents. Apart from the crucial aspects of pre-emptively guaranteeing safety in human-robot and robot-robot interaction, there is also the aspect of accounting for malicious behavior. One of the significant challenges is that non-industrial unmanned aerial vehicles (UAVs) and robots can simply be programmed to ignore any protocol and they can be made non-identifiable (by 3D printing components), thereby transcending the passive adversary model. This requires setting up an extrinsic software and hardware infrastructure to deal with otherwise impossible goals.

Dependable cyber-physical systems

A critical avenue of exploration is security research for verification and accountability under privacy requirements and abuse prevention, for future networks of autonomous agents, such as UAVs and robots that will carry out daily tasks and can be corrupted or misbehave.

A first part of the solution needs to design a system of granting and revoking privileges to agents for their tasks, that can be privately locally verified, inspired from e.g., [18], but accounting for faults and removing the reliance on a trusted certificate provider. Another part of the solution needs to deal with anonymous reporting, provability and traceability of unlawful actions, e.g., using tools from [143], without requiring publishing location, identity information or other privacy sensitive-information related to the tasks carried out. A third dimension of the solution should focus on competitive-collaborative tasks, where swarms of autonomous agents need to collaborate to achieve a common goal, while possibly satisfying their own private local goal. In such scenarios, communicating only encrypted data prevents unintended privacy loss of the local data and also protects the data of the other participants when an agent is captured in an adversarial environment and its memory seized. However, encryption hampers the detection of incorrect and adversarially supplied data, which might lead to failure in achieving the goal. My aim is to exploit the tight interconnection between the cyber and physical parts in such systems and use physical information, such as admissible velocity and location, to bound malicious behavior.

Zero-knowledge of correct execution in control

Synthesis of control algorithms with provable guarantees is a research field in itself. However, there are very few works that deal with the verification of a malicious controller that cheats during the execution of the protocol. A first step would be to address computations performed unencrypted at a user or a server, through a verifier that checks in zero-knowledge that the computation was correctly executed. Such a use case is key in industry, where a company wants to protect its proprietary data and algorithms, but a regulatory agency or clients should check that laws are being followed (e.g., emission control in autovehicles) or perform quality control (e.g., concentration of chemical products, faults in batch devices), without learning proprietary details. In the context of statistical tests, optimization and control, the challenges lie in designing and adapting the required checks in order to be efficiently and correctly performed in zero-knowledge, as well as in dealing with the multi-dimensional nature of the data, such that the verification step does not become a computational burden. The next steps would be to incorporate malicious adversaries and formal verification in the setting of control algorithms, in order to ensure that the protocols behave correctly even under the misbehavior of the other participants in the computation, and crucially, under privacy requirements and in the required sampling time.

Internet of secure multi-party computation

More than simply being connected and gathering data, the trend for the "things" in the Internet of Things is to also be computing devices in a heterogeneous computation network. Blockchains are establishing themselves as distributed solutions for reliable computations, and new proposals achieve secure, private and anonymous computation on blockchains, even under malicious faults, e.g., [34]. An interesting research direction is the design of protocols for mapping computations to networks of computing devices and reusing computations necessary for transaction posting to compute useful functions, while avoiding DDoS attacks.

Appendices

Appendix A

More background details

A.1 Decisional Composite Residuosity

Consider the additive group of integers modulo N, $\mathbb{Z}_N$, where $N = pq$ is a large modulus composed of two prime numbers of equal bit-length, p and q, such that $\gcd(\phi(N), N) = 1$. $\phi(N) = (p-1)(q-1)$ is the order of $\mathbb{Z}_N^*$. Now consider the multiplicative group of integers modulo N^2, $\mathbb{Z}_{N^2}^*$. The order of $\mathbb{Z}_{N^2}^*$ is $N\phi(N)$. An important subgroup of $\mathbb{Z}_{N^2}^*$ is:

$$\Gamma_N := \{(1+N)^\alpha \bmod N^2 | \alpha \in \{0,\ldots,N-1\}\} = \{1+\alpha N | \alpha \in \{0,\ldots,N-1\}\}, \quad \text{(A.1.1)}$$

where the equality follows from the binomial theorem: $(1+N)^\alpha = 1 + \alpha N \bmod N^2$. Computing discrete logarithms in Γ_N is easy [131, 177]: given $x, y \in \Gamma_N$, we can find β such that $y = x^\beta \bmod N^2$ by $\beta = (y-1)/(x-1) \bmod N$.

Another important subgroup in $\mathbb{Z}_{N^2}^*$ is:

$$\mathfrak{G}_N := \{x^N \bmod N^2 | x \in \mathbb{Z}_N^*\}. \quad \text{(A.1.2)}$$

$\mathfrak{G}_N$ has order $\phi(N)$. Computing discrete logarithms in $\mathfrak{G}_N$ is as hard as computing discrete logarithms in $\mathbb{Z}_N^*$ [131].

We also have the modular equalities for $x \in \mathbb{Z}_{N^2}^*$:

$$x^{\phi(N)} = 1 \bmod N, \quad x^{N\phi(N)} = 1 \bmod N^2. \tag{A.1.3}$$

Definition A.1.1. Let $N = pq$ be a product of two large primes. The **Decisional Composite Residuosity** (DCR) problem in $\mathbb{Z}_{N^2}^*$ is to distinguish among the following two distributions given $N = pq$:

$$D_0 = \{x^N \bmod N^2 | x \in_R \mathbb{Z}_N\}, \quad D_1 = \{x \in_R \mathbb{Z}_{N^2}^*\},$$

where $\in_R$ means drawn at random. The DCR assumption states that the advantage of any distinguisher $\mathcal{D}$, defined as the distance:

$$\mathbf{Adv}^{\mathrm{DCR}}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}(y, N) = 1 | y = x^N \bmod N^2, x \in_R \mathbb{Z}_N\right] - \Pr\left[\mathcal{D}(y, N) = 1 | y \in_R \mathbb{Z}_{N^2}^*\right]\right|$$

where probabilities are taken over all coin tosses, is a negligible function.

The name of the assumption comes from the fact a value $y = x^N \bmod N^2$ is called an N'th residue $\bmod N^2$, for a composite modulus $N = pq$.

Under the DCR assumption (i.e., distinguishing between an element from $\mathfrak{G}_N$ and an element from $\mathbb{Z}_{N^2}^*$ is hard), the Paillier scheme is semantically secure.

A.2 Ring Learning with Errors

Let κ denote the security parameter and $q = q(\kappa)$ a prime. Consider the ring $\mathcal{R} = \mathbb{Z}[X]/\langle\Phi(X)\rangle$, where $\Phi(x) = x^N + 1$ is a cyclotomic polynomial with $N = 2^r$, for $r > 2$, and the quotient ring $\mathcal{R}_q = \mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[X]/\langle\Phi(X)\rangle$. A Ring Learning with Errors (R-LWE) term is composed of:

$$(\mathbf{a}_i, \mathbf{b}_i) \in \mathcal{R}_q \times \mathcal{R}_q, \text{ where } \mathbf{b}_i = \mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i, \tag{A.2.1}$$

with $\mathbf{a}_i \overset{\$}{\leftarrow} \mathcal{R}_q$ a polynomial from $\mathcal{R}_q$ and the secret $\mathbf{s} \overset{\$}{\leftarrow} \mathcal{R}_q$. The error term $\mathbf{e}_i \in \mathcal{R}_q$ is sampled independently according to a discretized Gaussian distribution.

Given polynomially many pairs $(\mathbf{a}_i, \mathbf{b}_i)$, the *Decisional R-LWE* problem asks to determine whether $\mathbf{b}_i$ were constructed as in (A.2.1) or were randomly sampled from $\mathcal{R}_q$. Informally, given these pairs, it is infeasible to recover the secret $\mathbf{s}$, see [156].

A.3 Proof of Proposition 2.6.1

We will show in the following proof how to construct the simulators in Definition 2.2.5 in order to prove the privacy of the oblivious transfer variant we use.

The public key pk and secret key sk of the Paillier scheme are fixed, because the inputs to party A are Paillier encryptions under these keys. Let us construct the view of A, with inputs $\mathrm{pk}, [[\sigma_0]], [[\sigma_1]]$ and output $[[\sigma_i]]$:

$$V_A(\mathrm{pk}, [[\sigma_0]], [[\sigma_1]]) = (\mathrm{pk}, [[\sigma_0]], [[\sigma_1]], r_0, r_1, [[i]], [[v_i]], [[\sigma_i]], \mathrm{coins}),$$

where coins are the random values used for encrypting r_0 and r_1 and $[[-1]]$. The view of party B, that has inputs $i, \mathrm{pk}, \mathrm{sk}$ and no output, is:

$$V_B(i, \mathrm{pk}, \mathrm{sk}) = (i, \mathrm{pk}, \mathrm{sk}, [[v_0]], [[v_1]], \mathrm{coins}),$$

where coins are the random values used for encrypting v_i, i and 0.

Now let us construct a simulator S_A that generates an indistinguishable view from party A. S_A takes as inputs $\mathrm{pk}, [[\sigma_0]], [[\sigma_1]], [[\sigma_i]]$ and generates $\widetilde{r_0}$ and $\widetilde{r_1}$ as random values in $\mathcal{M}$. It then selects a random bit $\widetilde{i}$ and encrypts it with pk and computes $[[\widetilde{v_i}]] = \mathrm{Add}\left([[\sigma_i]], \mathrm{cMlt}(-\widetilde{r_0}, \mathrm{Add}([[\widetilde{i}]], [[-1]])), \mathrm{cMlt}(\widetilde{r_1}, [[\widetilde{i}]])\right)$. It also generates $\widetilde{\mathrm{coins}}$ for three encryptions. S_A outputs:

$$S_A(\mathrm{pk}, [[\sigma_0]], [[\sigma_1]], [[\sigma_i]]) = \left(\mathrm{pk}, [[\sigma_0]], [[\sigma_1]], \widetilde{r_0}, \widetilde{r_1}, [[\widetilde{i}]], [[\widetilde{v_i}]], [[\sigma_i]], \widetilde{\mathrm{coins}}\right).$$

First, $\widetilde{r_0}, \widetilde{r_1}$ and $\widetilde{\text{coins}}$ are statistically indistinguishable from r_0, r_1 and coins because they were generated from the same distributions. Second, $[[\widetilde{i}]]$ and $[[\widetilde{v_i}]]$ are indistinguishable from $[[i]]$ and $[[v_i]]$ because AHE is semantically secure and has indistinguishable encryptions, and because $[[\sigma_i]]$ is a refreshed value of $[[\sigma_0]]$ or $[[\sigma_1]]$. This means A cannot learn any information about i, hence $[[i]]$ looks like the encryption of a random bit, i.e., like $[[\widetilde{i}]]$. Thus, $V_A(\text{pk}, [[\sigma_0]], [[\sigma_1]]) \overset{c}{\equiv} S_A(\text{pk}, [[\sigma_0]], [[\sigma_1]], [[\sigma_i]])$.

A simulator S_B for party B takes as inputs i, pk, sk and generates two random values from $\mathcal{M}$, names them $\widetilde{v_0}$ and $\widetilde{v_1}$ and encrypts them. It then generates $\widetilde{\text{coins}}$ as random values for three encryptions. S_B outputs:

$$S_B(i, \text{pk}, \text{sk}) = \Big(i, \text{pk}, \text{sk}, [[\widetilde{v_0}]], [[\widetilde{v_1}]], \widetilde{\text{coins}}\Big) .$$

First, $\widetilde{\text{coins}}$ are statistically indistinguishable from coins because they were generated from the same distribution. Second, $\widetilde{v_0}$ and $\widetilde{v_1}$ are also statistically indistinguishable from each other and from v_0 and v_1 due to the security of the one-time pad. Their encryptions will also be indistinguishable. Thus, $V_B(i, \text{pk}, \text{sk}) \overset{c}{\equiv} S_B(i, \text{pk}, \text{sk})$. $\square$

Appendix B

Technical details for Chapter 3

B.1 Proof of Theorem 3.2.2

For simplicity of the exposition, we avoid mentioning the public data (public keys $\mathrm{pk}_{\mathcal{R}}$, $\mathrm{pk}_{\mathrm{DGK}}$, number of iterations K and number of bits l) in the views, since all parties have access to them. In what follows, we will successively discuss the views of each type of party participating in the protocol: agents, cloud and requester. As mentioned in Definitions 2.2.5 and 2.2.6, the views of the parties during the execution of Protocol 3.2.4 are constructed on all the inputs of the parties involved and symbolize the real values the parties get in the execution of the protocol. We will denote by $\mathcal{I}$ the inputs of all the parties:

$$\mathcal{I} = \{\mathbf{b}_{\mathcal{A}}, \mathbf{c}_{\mathcal{A}}, \mathbf{A}_{\mathcal{C}}, \mathbf{Q}_{\mathcal{C}}, \mathrm{sk}_{\mathcal{R}}, \mathrm{sk}_{\mathrm{DGK}}\}.$$

Furthermore, in order to construct a simulator that simulates the actions of a party, we need to feed into it the inputs and outputs of that respective party.

Simulator for one agent $\mathcal{A}_i$

Agent $\mathcal{A}_i$, for every $i = 1, \ldots, p$, has the following inputs $I_{\mathcal{A}_i} = (\{\mathbf{b}_j\}_{j=1,\ldots,m_i}, \{\mathbf{c}_j\}_{j=1,\ldots,n_i})$. To avoid cluttering, we will drop the subscripts $j = 1, \ldots, m_i$ and $j = 1, \ldots, n_i$. Then agent

$\mathcal{A}_i$ has the following view:

$$V_{\mathcal{A}_i}(\mathcal{I}) := (\mathbf{b}_j, \mathbf{c}_j, [[\mathbf{b}_j]], [[\mathbf{c}_j]], \text{coins}),$$

where coins represent the random values used for encryption.

The agents are only online to send their encrypted data to the cloud, and they do not have any role and output afterwards. Hence, a simulator $S_{\mathcal{A}_i}$ would simply generate the random values necessary to encrypt its inputs as specified by the protocol and output the view obtained such:

$$S_{\mathcal{A}_i} := (\mathbf{b}_j, \mathbf{c}_j, \widetilde{[[\mathbf{b}_j]]}, \widetilde{[[\mathbf{c}_j]]}, \widetilde{\text{coins}}),$$

where by $\widetilde{(\cdot)}$ we denote the quantities obtained by the simulator, which are different than the quantities of the agents, but follow the same distribution. Hence, it is trivial to see that the protocol is secure in the semi-honest model from the point of view of the interaction with the agents.

Next, we want to prove the privacy of the protocol from the point of view of the interactions with the cloud and the requester. We will construct a sequence of algorithms such that we obtain that the view of the real parties after the execution of K iterations is the same as the view of simulators that simply execute K iterations with random exchanged messages. For the simplicity of the exposition, we will treat all our variables as scalars μ, $b_{\mathcal{A}}$, $c_{\mathcal{A}}$, $A_{\mathcal{C}}, Q_{\mathcal{C}}$, x^* as scalars. The same steps are repeated for every element in the vectors, when we consider multi-dimensional variables (recall that some of the protocols are performed element-wise).

Simulator for the cloud $\mathcal{C}$

For clarity, we will form the view of the cloud in steps, using pointers to the lines in Protocol 3.2.4. The view of the cloud during the execution of lines 5-7 is:

$$V_{\mathcal{C}}^{-1}(\mathcal{I}) = \big(A_{\mathcal{C}}, Q_{\mathcal{C}}, \eta, [[b_{\mathcal{A}}]], [[c_{\mathcal{A}}]], [[\mu_0]], \text{coins}\big) =: I_{\mathcal{C}}^{-1}, \tag{B.1.1}$$

where coins represent the random values generated for the Paillier encryption. Furthermore, we construct the view of the cloud at iteration $k = 0, \dots, K-1$ during the execution of an instance of Protocol 3.2.3, which will be constructed on the inputs of all parties: the inputs $\mathcal{I}$ and the data the parties had at iteration $k-1$. We denote the view of the cloud at iteration $k-1$ by $I_{\mathcal{C}}^{k-1}$ which, along with $\mathcal{I}$ and the view of the requester at iteration $k-1$, denoted by $I_{\mathcal{R}}^{k-1}$ (B.1.5), will be what the view is constructed on at iteration k.

$$\bar{\mathcal{I}}^{k-1} := \mathcal{I} \cup I_{\mathcal{C}}^{k-1} \cup I_{\mathcal{R}}^{k-1}, \tag{B.1.2}$$

$$I_{\mathcal{C}}^{k} := V_{\mathcal{C}}^{k}(\bar{\mathcal{I}}^{k-1}) = \big(I_{\mathcal{C}}^{k-1}, [[\mu_k]], [[\bar{\mu}_k]], \underbrace{\pi_k, \text{coins}_{1k}}_{\textit{Protocol } 3.2.1}, \tag{B.1.3}$$

$$\underbrace{\rho_k, [[\delta_k]], \text{m}^{\text{comp}_k}, \text{coins}_{2k}}_{\textit{Protocol } 2.7.2}, \underbrace{r_k, s_k, [[v_k]], [[\mu_{k+1}]], \text{coins}_{3k}}_{\textit{Protocol } 3.2.2}\big),$$

where m^{comp_k} are messages exchanged in the comparison protocol, and coins_{jk}, for $j = 1, 2, 3$ are the random numbers generated in Protocol j. Finally, the view of the cloud after the execution of line 11 in Protocol 3.2.4 is:

$$V_{\mathcal{C}}^{K}(\bar{\mathcal{I}}^{K-1}) = \big(I_{\mathcal{C}}^{K-1}, [[x^*]]\big). \tag{B.1.4}$$

Therefore, the view of the cloud during the whole execution of Protocol 3.2.4 is:

$$V_{\mathcal{C}}(\mathcal{I}) := V_{\mathcal{C}}^{K}(\bar{\mathcal{I}}^{K-1}).$$

We first construct a simulator on inputs $I_{\mathcal{C}} = \{A_{\mathcal{C}}, Q_{\mathcal{C}}\}$ that mimics $V_{\mathcal{C}}^{-1}(\mathcal{I})$ in (B.1.1):

(i) Generate $n + m$ random numbers of l bits $\widetilde{b_{\mathcal{A}}}, \widetilde{c_{\mathcal{A}}}$; (ii) Generate a random positive initial value $\widetilde{\mu_0}$; (iii) Generate $n+m+1$ uniformly random numbers for the Paillier encryption and denote them $\widetilde{\text{coins}}$; (iv) Compute $[[\widetilde{b_{\mathcal{A}}}]], [[\widetilde{c_{\mathcal{A}}}]], [[\widetilde{\mu_0}]]$; (v) Compute η following line 6; (vi) Output $\widetilde{I_{\mathcal{C}}^{-1}} := S_{\mathcal{C}}^{-1}(I_{\mathcal{C}}) = \big(A_{\mathcal{C}}, Q_{\mathcal{C}}, [[\widetilde{b_{\mathcal{A}}}]], [[\widetilde{c_{\mathcal{A}}}]], \eta, [[\widetilde{\mu_0}]], \widetilde{\text{coins}}\big)$.

Since Protocol 3.2.3 is secure in the semi-honest model (Proposition 3.2.1), we know that there exists a probabilistic polynomial-time (ppt) simulator for the functionality of

Protocol 3.2.3 on inputs $(A_\mathcal{C}, Q_\mathcal{C}, [[b_\mathcal{A}]], [[c_\mathcal{A}]], \eta, [[\mu_k]])$ and output $[[\mu_{k+1}]]$. However, we need to show that we can simulate the functionality of consecutive calls of Protocol 3.2.3, or, equivalently, on one call of Protocol 3.2.3 but on the augmented input that contains the data of the cloud in the previous iterations. Call such a simulator $S_\mathcal{C}^k$, that on the input $I_\mathcal{C}^{k-1}$ mimics $V_\mathcal{C}^k(\bar{\mathcal{I}}^{k-1})$ in (B.1.3), for $k = 0, \dots, K-1$:

(i) Compute $[[\nabla g(\mu_k)]]$ and $[[\bar{\mu}_k]]$ as in lines 1-2 of Protocol 3.2.3 from $[[\mu_k]], [[b_\mathcal{A}]], [[c_\mathcal{A}]]$ which are included in $I_\mathcal{C}^{k-1}$; (ii) Generate a random permutation $\widetilde{\pi_k}$ and apply it on $([[0]], [[\bar{\mu}_k]])$ as in Protocol 3.2.1; (iii) Follow Protocol 2.7.2 and replace the incoming messages by DGK encryptions of appropriately sized random values to obtain $\widetilde{\rho_k}, \widetilde{\mathrm{m}^{\mathrm{comp}_k}}$; (iv) Generate random bit $\widetilde{\delta_k}$ and its encryption $[[\widetilde{\delta_k}]]$; (v) Generate random values $\widetilde{r_k}$ and $\widetilde{s_k}$ as in line 1 in Protocol 3.2.2 and their encryptions $[[\widetilde{r_k}]], [[\widetilde{s_k}]]$; (vi) Obtain $[[\widetilde{v_k}]]$ by choosing between the elements of $\widetilde{\pi_k}([[0]], [[\bar{\mu}_k]]) + (\widetilde{r_k}, \widetilde{s_k})$ according to the generated $\widetilde{\delta_k}$; (vii) Compute $[[\widetilde{\mu_{k+1}}]]$ as in line 8 of Protocol 3.2.2; (viii) Denote the rest of the random values used for encryption and blinding by $\widetilde{\mathrm{coins}_k}$; (ix) Output $\widetilde{I_\mathcal{C}^k} := S_\mathcal{C}^k(I_\mathcal{C}^{k-1}) = (I_\mathcal{C}^{k-1}, [[\bar{\mu}_k]], [[\widetilde{\pi_k}]], [[\widetilde{z_k}]], \widetilde{\mathrm{m}^{\mathrm{comp}_k}}, [[\widetilde{\delta_k}]], [[\widetilde{r_k}]], [[\widetilde{s_k}]], [[\widetilde{v_k}]], [[\widetilde{\mu_{k+1}}]], \widetilde{\mathrm{coins}_k})$.

Finally, a trivial simulator $\widetilde{I_\mathcal{C}^K} := S_\mathcal{C}^K(I_\mathcal{C}^{K-1})$ for $V_\mathcal{C}^K(\bar{\mathcal{I}}^{K-1})$ is obtained by simply performing line 11 on the inputs.

Proposition B.1.1. *$S_\mathcal{C}^k(I_\mathcal{C}^{k-1}) \overset{c}{\equiv} V_\mathcal{C}^k(\bar{\mathcal{I}}^{k-1})$, for $k = -1, \dots, K$, where $I_\mathcal{C}^{k-2} := I_\mathcal{C}$.*

Proof. For $k = -1$, due to the fact that the coins and the initial iterate are generated by the simulator via the same distributions as specified in the protocol, and due to the semantic security of the Paillier encryption, which guarantees indistinguishability of encryptions, we obtain that $V_\mathcal{C}^{-1}(\mathcal{I}) \overset{c}{\equiv} S_\mathcal{C}^{-1}(I_\mathcal{C})$.

For $0 \leq k \leq K-1$, $(\mathrm{coins}_{1k}, \mathrm{coins}_{2k}, \mathrm{coins}_{3k}) \overset{s}{\equiv} \widetilde{\mathrm{coins}_k}$ because they are generated from the same distributions. Similarly, the ensemble distributions of $(\widetilde{\pi_k}, \widetilde{\pi_k}([[0]], [[\bar{\mu}_k]]))$ and $(\pi_k, \pi_k([[0]], [[\bar{\mu}_k]]))$ are the same and $\widetilde{\pi_k}, \pi_k$ are independent of the other parameters. The quantities from the DGK protocol $\widetilde{\rho_k} \overset{s}{\equiv} \rho_k$ and $\widetilde{\mathrm{m}^{\mathrm{comp}_k}} \overset{c}{\equiv} \mathrm{m}^{\mathrm{comp}_k}$ either due to the semantic security of the DGK scheme, e.g. $[\widetilde{t'}], [\widetilde{\delta_\mathcal{R}}]$, or due to having the same distributions, e.g. $\widetilde{\rho_k}, \widetilde{\delta_\mathcal{C}}, \widetilde{\alpha}$. The values in the update step $\widetilde{r_k}, \widetilde{s_k}$ are sampled from the same distribution as

r_k, s_k, and, finally, $[[\widetilde{\delta_k}]], [[\widetilde{v_k}]], [[\widetilde{\mu_{k+1}}]] \overset{c}{\equiv} [[\delta_k]], [[v_k]], [[\mu_{k+1}]]$ due to the semantic security of the Paillier encryption. Thus, $S^k(I_{\mathcal{C}}^{k-1}) \overset{c}{\equiv} V_{\mathcal{C}}^k(\bar{\mathcal{I}}^{k-1})$.

For $k = K$, $S_{\mathcal{C}}^K(I_{\mathcal{C}}^{K-1}) \overset{c}{\equiv} V_{\mathcal{C}}^K(\bar{\mathcal{I}}^{K-1})$ follows from the fact that the simulator simply executes the last part of the protocol and from the Paillier scheme's semantic security. □

Hence, we obtained that $I_{\mathcal{C}}^k \overset{c}{\equiv} \widetilde{I}_{\mathcal{C}}^k$, for $k = -1, \ldots, K$. The essence of the proof of Proposition B.1.1 is that all the messages the cloud receives are encrypted. Then, thanks to the semantic security of the Paillier and DGK schemes, the extra information included in $\bar{\mathcal{I}}^{k-1}$ from the previous iterations cannot be used to extract other information about the values at iteration k. From this, we also have the next corollary:

Corollary B.1.2. *$S_{\mathcal{C}}^k(\widetilde{I}_{\mathcal{C}}^{k-1}) \overset{c}{\equiv} S_{\mathcal{C}}^k(I_{\mathcal{C}}^{k-1})$, equivalent to $S_{\mathcal{C}}^{k+1}(S_{\mathcal{C}}^k(I_{\mathcal{C}}^{k-1})) \overset{c}{\equiv} S_{\mathcal{C}}^{k+1}(V_{\mathcal{C}}^k(\bar{\mathcal{I}}^{k-1}))$, for any $k = 0, \ldots, K-1$.*

Finally, we construct a simulator $S_{\mathcal{C}}(I_{\mathcal{C}})$ for the execution of Protocol 3.2.4 and we will show that its view will be computationally indistinguishable from $V_{\mathcal{C}}(\mathcal{I})$. To this end, we define the following sequence of views–obtained as hybrids between the real views and the views of the simulators:

$$\begin{aligned}
V_{\mathcal{C}}(\mathcal{I}) = H_{-1}(\mathcal{I}) &= V_{\mathcal{C}}^K(\bar{\mathcal{I}}^{K-1}) \\
H_0(\mathcal{I}) &= S_{\mathcal{C}}^K \circ V_{\mathcal{C}}^{K-1}(\bar{\mathcal{I}}^{K-2}) \\
H_1(\mathcal{I}) &= S_{\mathcal{C}}^K \circ S_{\mathcal{C}}^{K-1} \circ V_{\mathcal{C}}^{K-2}(\bar{\mathcal{I}}^{K-3}) \\
&\vdots \\
H_K(\mathcal{I}) &= S_{\mathcal{C}}^K \circ S_{\mathcal{C}}^{K-1} \circ \ldots \circ S_{\mathcal{C}}^0 \circ V_{\mathcal{C}}^{-1}(\mathcal{I}) \\
S_{\mathcal{C}}(I_{\mathcal{C}}) = H_{K+1}(I_{\mathcal{C}}) &= S_{\mathcal{C}}^K \circ S_{\mathcal{C}}^{K-1} \circ \ldots \circ S_{\mathcal{C}}^0 \circ S_{\mathcal{C}}^{-1}(I_{\mathcal{C}}).
\end{aligned}$$

By transitivity, H_{-1} and H_{K+1} are computationally indistinguishable if:

$$H_{-1} \overset{c}{\equiv} H_0 \overset{c}{\equiv} \ldots \overset{c}{\equiv} H_k \overset{c}{\equiv} H_{k+1} \overset{c}{\equiv} H_{k+2} \overset{c}{\equiv} \ldots \overset{c}{\equiv} H_{K+1}.$$

This result follows from induction on Corollary B.1.2. In conclusion, we obtain that $S_{\mathcal{C}}(I_{\mathcal{C}}) \overset{c}{\equiv}$

$V_{\mathcal{C}}(\mathcal{I})$, which verifies that Protocol 3.2.4 achieves privacy with respect to the cloud.

Simulator for the requester $\mathcal{R}$

We proceed with the same steps in order to show that the consecutive K iterations form a protocol that is secure in the semi-honest model from the point of view of the requester. The main difference is that for the cloud we used the semantic security of the Paillier and DGK cryptosystems, while for the requester we will use the secrecy of the one-time pad variant used for blinding. The symmetric encryption used in this chapter, as discussed in Preamble 2.4, to which we refer as blinding, guarantees that a value of l bits additively blinded by a random value of $l+\lambda$ bits is statistically indistinguishable from a random value of $l+\lambda+1$ bits.

The inputs and output of the requester in Protocol 3.2.4 are $I_{\mathcal{R}} = (\mathrm{sk}_{\mathcal{R}}, \mathrm{sk}_{\mathrm{DGK}}, x^*)$. As in (B.1.2), $\bar{\mathcal{I}}^{k-1}$ represents the inputs of all the parties at iteration k, with $\bar{\mathcal{I}}^{-1} = \mathcal{I}$. Then, the view of the requester during iteration $k = 0, \ldots, K-1$ is:

$$I^k_{\mathcal{R}} := V^k_{\mathcal{R}}(\bar{\mathcal{I}}^{k-1}) = (\mathrm{sk}_{\mathcal{R}}, \mathrm{sk}_{\mathrm{DGK}}, \underbrace{z_k, \delta_k, \mathrm{m}^{\mathrm{comp}_k}, \mathrm{coins}_{2k}}_{Protocol\ 2.7.2}, \underbrace{\bar{a}_k, \bar{b}_k, v_k, \mathrm{coins}_{3k}}_{Protocol\ 3.2.2}). \tag{B.1.5}$$

The view of the requester during the last step of the protocol is:

$$V^K_{\mathcal{R}}(\bar{\mathcal{I}}^{K-1}) := (I^{K-1}_{\mathcal{R}}, [[x^*]]). \tag{B.1.6}$$

As before, the view of the requester during the execution of Protocol 3.2.4 is:

$$V_{\mathcal{R}}(\mathcal{I}) := V^K_{\mathcal{R}}(\bar{\mathcal{I}}^{K-1}).$$

In order to be able to construct a simulator with indistinguishable view from the view of the requester, we need to show that the requester is not capable of inferring new relevant information about the private data (other than what can be inferred solely from its inputs and outputs) although it knows the optimal solution x^* and has access to the messages from multiple iterations.

Apart from the last message, which is the encryption of the optimization solution $[[x^*]]$, and the comparison results δ_k, all the values the requester receives are blinded, with different sufficiently large values at each iteration. The requester knows that $Q_{\mathcal{C}}x^* = A_{\mathcal{C}}^\intercal \mu_K - c_{\mathcal{A}}$ and that $\mu_K = v_K - \bar{r}_K$, for some random value $\bar{r}_K$. However, from these two equations, although it has access to x^* and v_K, the requester cannot infer any new information about the private data of the agents or about μ_K, even when $Q_{\mathcal{C}}, A_{\mathcal{C}}$ are public, thanks to the large enough randomization.

Let $a_k, b_k = \pi_k(0, \bar{\mu}_k)$, where π_k is a random permutation. From Protocol 2.7.2, the requester receives z_k which is the additively blinded value of $b_k - a_k + 2^l$, and other blinded values, denoted in (B.1.5) as $\mathrm{m}^{\mathrm{comp}_k}$. Hence, provided the blinding noises are refreshed at every iteration, the requester cannot infer any information, as follows from Preamble 2.4. At the end of the protocol, it receives the bit δ_k which is 1 if $a_k \leq b_k$ and 0 otherwise. However, due to the uniform randomization between the order of $\bar{\mu}_k$ and 0 at every iteration for assigning the values of a_k and b_k, described in Protocol 3.2.1, the requester cannot identify the sign of $\bar{\mu}_k$, and by induction, the sign of $\bar{\mu}_{K-1}$ and magnitude of μ_K. This means that having access to $x^* = x_K$ does not bring more information about the blinded values from the intermediary steps.

We now investigate the relation between the messages from consecutive iterations. From the update protocol 3.2.2, we know:

$$\begin{aligned} v_k &= (a_k + r_k)(1-\delta_k) + (b_k + s_k)\delta_k \\ \mu_{k+1} &= v_k - r_k(1-\delta_k) - s_k\delta_k = a_k(1-\delta_k) + b_k\delta_k, \end{aligned}$$

The requester knows the values of v_k and δ_k, but $\pi_k, \bar{\mu}_k, r_k, s_k$ are unknown to it. Furthermore, let π_{k+1} be the permutation applied by the cloud at step $k+1$ in Protocol 3.2.1, unknown to the requester. Take for example the case when $\delta_k = 0$ and $\delta_{k+1} = 0$. Let $\tilde{Q} = I - \eta A_{\mathcal{C}} Q_{\mathcal{C}}^{-1} A_{\mathcal{C}}^\intercal$ and $E = [I\ 0]$. Then:

$$v_{k+1} = a_{k+1} + r_{k+1} = E\pi_{k+1}(0, \tilde{Q}(v_k - r_k) - \eta A_{\mathcal{C}} Q_{\mathcal{C}}^{-1} c_{\mathcal{A}} - \eta b_{\mathcal{A}}) + r_{k+1}.$$

The above equation shows the requester cannot construct v_{k+1} from v_k. Similar equations arise when considering the other values for δ_k and δ_{k+1}. This guarantees that an integer $\tilde{v}_k^i$ obtained by selecting uniformly at random from $(2^{l+\lambda}, 2^{l+\lambda+1}) \cap \mathbb{Z}_N$ will have the distribution statistically indistinguishable from v_k^i. Moreover:

$$\begin{aligned}
\mu_{k+2} &= \max\{0, \tilde{Q}\mu_{k+1} - \eta A_{\mathcal{C}} Q_{\mathcal{C}}^{-1} c_{\mathcal{A}} - \eta b_{\mathcal{A}}\} \\
&= \max\{0, \tilde{Q}(v_k - r_k(1-\delta_k) - s_k\delta_k) - \eta A_{\mathcal{C}} Q_{\mathcal{C}}^{-1} c_{\mathcal{A}} - \eta b_{\mathcal{A}}\} \\
&= v_{k+1} - r_{k+1}(1-\delta_{k+1}) - s_{k+1}\delta_{k+1}.
\end{aligned}$$

Since the blinding noise is different at each iteration and uniformly sampled, the requester cannot retrieve μ_{k+2} from v_k and v_{k+1}. In short, if the requester receives some random values $\widetilde{\bar{a}}_k, \widetilde{\bar{b}}_k \in (2^{l+\lambda}, 2^{l+\lambda+1}) \cap \mathbb{Z}_N$ instead of $a_k + r_k$ and $b_k + s_k$ respectively, it would not be able to distinguish the difference. Similar arguments hold for the blinded messages c_i from the comparison protocol 2.7.1.

Hence, by processing multiple iterations, the requester can only obtain functions of the private data that involve at least one large random value, which does not break privacy.

We now build a simulator $S_{\mathcal{R}}$ that applies the steps of the protocol on randomly generated values. As before, since Protocol 3.2.3 is secure in the semi-honest model (Proposition 3.2.1), we know that there exists a ppt simulator for the functionality of Protocol 3.2.3 on inputs $I_{\mathcal{R}}^{-1} = \{\mathrm{sk}_{\mathcal{R}}, \mathrm{sk}_{\mathrm{DGK}}\}$. However, we need to show that we can simulate the functionality of consecutive calls of Protocol 3.2.3, or, equivalently, on one call of Protocol 3.2.3 but on inputs $(I_{\mathcal{R}}^k, x^*)$. Call such a simulator $S_{\mathcal{R}}^k$, that on the inputs $(I_{\mathcal{R}}^k, x^*)$ should output a view that is statistically indistinguishable from $V_{\mathcal{R}}^k(\bar{\mathcal{I}}^{k-1})$ in (B.1.5), for $k = 0, \ldots, K-1$. We already showed that although the requester has access to the output x^* and to blinded messages from all the iterations of Protocol 3.2.4, it cannot extract information from them or correlate the messages to the iteration they arise from, so x^* will only be relevant in the last simulator:

(i) Generate a $\lambda + l$-length random integer $\widetilde{\rho}_k$ and add 2^l and obtain $\widetilde{z}_k$; (ii) Generate a random bit $\widetilde{\delta_k}$; (iii) Choose a random bit $\widetilde{\delta_{\mathcal{R}}}$. If it is 0, then generate l nonzero values of $2t$ bits, otherwise generate $l-1$ nonzero random values and one 0 value. Those will be the $\widetilde{\mathrm{m}^{\mathrm{comp}_k}}$ (see Protocol 2.7.1); (iv) Generate random integers of length $l+\lambda+1$ $\widetilde{\bar{a}_k}$ and $\widetilde{\bar{b}_k}$; (v) Compute $\widetilde{v_k}$ according to $\widetilde{\delta_k}$; (vi) Denote all Paillier and DGK generated coins by $\widetilde{\mathrm{coins}}$; (vii) Output $\widetilde{I}^k_{\mathcal{R}} := S^k_{\mathcal{R}}(I^{k-1}_{\mathcal{R}}, x^*) = (I^{k-1}_{\mathcal{R}}, \widetilde{z_k}, \widetilde{\delta_k}, \widetilde{\mathrm{m}^{\mathrm{comp}_k}}, \widetilde{\bar{a}_k}, \widetilde{\bar{b}_k}, \widetilde{v_k}, \widetilde{\mathrm{coins}})$.

Finally, a trivial simulator $\widetilde{I}^K_{\mathcal{R}} := S^K_{\mathcal{R}}(I^{K-1}_{\mathcal{R}}, x^*)$ for $V^K_{\mathcal{R}}(\bar{\mathcal{I}}^{K-1})$ is obtained by simply generating an encryption of x^* and outputting: $(I^{K-1}_{\mathcal{R}}, \widetilde{[[x^*]]})$.

Proposition B.1.3. *$S^k_{\mathcal{R}}(I^{k-1}_{\mathcal{R}}, x^*) \overset{c}{\equiv} V^k(\bar{\mathcal{I}}^{k-1})$, for $k = 0, \dots, K$.*

Proof. For $0 \le k \le K-1$, $(\mathrm{coins}_{2k}, \mathrm{coins}_{3k}) \overset{s}{\equiv} \widetilde{\mathrm{coins}_k}$ because they are generated from the same distributions. Similarly, $\widetilde{z_k} \overset{s}{\equiv} z_k$ because $\widetilde{\rho_k} \overset{s}{\equiv} b_k - a_k + \rho_k$. From the discussion above, the same holds for $\widetilde{\mathrm{m}^{\mathrm{comp}_k}}$ and their counterparts $\mathrm{m}^{\mathrm{comp}_k}$. Furthermore, $(\widetilde{\delta_k}, \widetilde{\bar{a}_k}, \widetilde{\bar{b}_k}, \widetilde{v_k})$ are statistically indistinguishable from $(\delta_k, \bar{a}_k, \bar{b}_k, v_k)$ due to the way they are generated, and $\widetilde{v_k}$ being consistent with $\widetilde{\delta_k}$. Thus, $S^k(I^{k-1}_{\mathcal{R}}, x^*) \overset{c}{\equiv} V^k(I^{k-1}_{\mathcal{R}} \cup \mathcal{I})$.

For $k = K$, $S^K(I^{K-1}_{\mathcal{R}}, x^*) \overset{c}{\equiv} V^K(\bar{\mathcal{I}}^{K-1})$ trivially follows from the fact that both $[[x^*]]$ and $\widetilde{[[x^*]]}$ are decrypted in x^*. □

Hence, we obtained that $\widetilde{I}^k_{\mathcal{R}} \overset{c}{\equiv} I^k_{\mathcal{R}}$, for $k = 0, \dots, K$. The essence of the proof of Proposition B.1.3 is that all the messages the requester receives are blinded with large enough random noise and the comparison bits are randomized. This results in the messages being statistically indistinguishable from random values of the same length, which means that the extra information included in $(I^{k-1}_{\mathcal{R}}, x^*)$ from the previous iterations cannot be used to extract other information about the current values at iteration k. The next corollary then follows from Proposition B.1.3:

Corollary B.1.4. *$S^k_{\mathcal{R}}(\widetilde{I}^{k-1}_{\mathcal{R}}, x^*) \overset{c}{\equiv} S^k_{\mathcal{R}}(I^{k-1}_{\mathcal{R}}, x^*)$, equivalent to $S^{k+1}_{\mathcal{R}}(S^k_{\mathcal{R}}(I^{k-1}_{\mathcal{R}}, x^*), x^*) \overset{c}{\equiv} S^{k+1}_{\mathcal{R}}(V^k(\bar{\mathcal{I}}^{k-1}), x^*)$, for any $k = 1, \dots, K-1$.*

Finally, we construct a simulator $S_{\mathcal{R}}(I_{\mathcal{R}})$ for the execution of Protocol 3.2.4 and we show that its view will be statistically indistinguishable from $V_{\mathcal{R}}(\mathcal{I})$. We define the following sequence, from which we drop the input x^* to the simulators to not burden the notation:

$$\begin{aligned}
V_{\mathcal{R}}(\mathcal{I}) = H_0(\mathcal{I}) &= V_{\mathcal{R}}^{K}(\bar{\mathcal{I}}^{K-1}) \\
H_1(\mathcal{I}, x^*) &= S_{\mathcal{R}}^{K} \circ V_{\mathcal{R}}^{K-1}(\bar{\mathcal{I}}^{K-2}) \\
H_2(\mathcal{I}, x^*) &= S_{\mathcal{R}}^{K} \circ S_{\mathcal{R}}^{K-1} \circ V_{\mathcal{R}}^{K-2}(\bar{\mathcal{I}}^{K-3}) \\
&\vdots \\
H_K(\mathcal{I}, x^*) &= S_{\mathcal{R}}^{K} \circ S_{\mathcal{R}}^{K-1} \circ \ldots \circ S_{\mathcal{R}}^{1} \circ V_{\mathcal{R}}^{0}(\mathcal{I}) \\
S_{\mathcal{R}}(I_{\mathcal{R}}) = H_{K+1}(I_{\mathcal{R}}) &= S_{\mathcal{R}}^{K} \circ S_{\mathcal{R}}^{K-1} \circ \ldots \circ S_{\mathcal{R}}^{1} \circ S_{\mathcal{R}}^{0}(I_{\mathcal{R}})
\end{aligned}$$

By transitivity, H_0 and H_{K+1} are statistically indistinguishable if:

$$H_0 \overset{c}{\equiv} H_1 \overset{c}{\equiv} \ldots \overset{c}{\equiv} H_k \overset{c}{\equiv} H_{k+1} \overset{c}{\equiv} H_{k+2} \overset{c}{\equiv} \ldots \overset{c}{\equiv} H_{K+1}.$$

The result follows from induction on Corollary B.1.4. In conclusion, we obtain that $S_{\mathcal{R}}(I_{\mathcal{R}}) \overset{c}{\equiv} V_{\mathcal{R}}(\mathcal{I})$ which verifies that Protocol 3.2.4 achieves privacy with respect to the requester.

The proof of Theorem 3.2.2 is now complete. $\square$

B.2 Proof of Theorem 3.2.3

We will now show that any coalition consistent with the setup of Propositions 3.2.6, 3.2.7 and with the assumption that the cloud and requester do not collude does not gain any new information about the private data of the honest agents, other than what can be inferred solely from the inputs and outputs of the coalition. As mentioned in Section 3.2.3.6, the agents in a coalition only add their inputs to the view of the coalition, but do not have any messages in the protocol.

Simulator for the cloud $\mathcal{C}$ and $\bar{p}$ agents $\mathcal{A}_i$

Consider the coalition between a set of agents $\mathcal{A}_{i=1,\ldots,\bar{p}}$ and the cloud $\mathcal{C}$, which has the inputs $(\{b_i\}_{i=1,\ldots,\bar{m}}, \{c_i\}_{i=1,\ldots,\bar{n}}, \mathbf{A}_{\mathcal{C}}, \mathbf{Q}_{\mathcal{C}})$ and no output from the execution of Protocol 3.2.4. Since $\bar{p} < p$, the coalition is not able to compute $\boldsymbol{\mu}_1$ by itself, and the semantic security of the

Paillier cryptosystem is again enough to ensure the privacy of the rest of the sensitive data. A simulator for this coalition can be constructed following the same steps as in the simulator for the cloud on the augmented inputs defined above, from the fact that every value the cloud receives is encrypted by a semantically secure cryptosystem.

Simulator for the requester $\mathcal{R}$ and $\bar{p}$ agents $\mathcal{A}_i$

Consider the coalition between a set of agents $\mathcal{A}_{i=1,\dots,\bar{p}}$ and the requester $\mathcal{R}$, which has the inputs $(\{b_i\}_{i=1,\dots,\bar{m}}, \{c_i\}_{i=1,\dots,\bar{n}}, \mathrm{sk}_{\mathcal{R}}, \mathrm{sk}_{\mathrm{DGK}})$ and output $(\mathbf{x}^*)$ from the execution of Protocol 3.2.4. If both matrices $\mathbf{Q}_{\mathcal{C}}, \mathbf{A}_{\mathcal{C}}$ are public, if there exists $i \in \{1, \dots, \bar{p}\}$ such that $\mathbf{a}_i^\intercal \mathbf{x}^* < b_i$, the coalitions finds out $\mu_i^* = 0$, which comes from public knowledge in the KKT conditions (3.2.15). From this, the coalition is able to find some coins of the cloud: r_K and s_K associated to element i. However, these values are independent from the rest of the parameters and do not reveal any information about the private inputs of the parties. Apart from this, the coalition is not able to compute any private data of the honest parties from the execution of the protocol, due to the secure blinding. A simulator for this coalition can be build by following the same steps as described in the simulator for the requester on the augmented inputs defined above.

The proof is now complete. □

B.3 Proof of Proposition 3.2.6

The coalition has access to the following data, which is fixed: $\mathbf{A}_{\mathcal{C}}, \mathbf{Q}_{\mathcal{C}}, \mathbf{x}^*, \{b_i\}_{1,\dots,\bar{m}}, \{c_i\}_{i,\dots,\bar{n}}$.

Proof of (1). We need to address two cases: the non-strict satisfaction of the constraints and the equality satisfaction of the constraints.

(I) Suppose there exists a solution $\{b_i\}_{\bar{m}+1,\dots,m}$ and $\{c_i\}_{\bar{n}+1,\dots,n}$ and $\boldsymbol{\mu}$ to the KKT conditions such that $\mathbf{a}_i^\intercal \mathbf{x}^* < b_i$ for some $\bar{m}+1 \leq i \leq m$. In particular this implies that $\mu_i = 0$. Define $\mathbf{c}' := \mathbf{c}_{\mathcal{A}}$, $\boldsymbol{\mu}' := \boldsymbol{\mu}$ and $\mathbf{b}'$ such that $b_j' := b_j$ for all $j \neq i$ and b_i' to take any value $b_i' \geq \mathbf{a}_i^\intercal \mathbf{x}^*$. The new set of points $(\mathbf{b}', \mathbf{c}', \boldsymbol{\mu}')$ is also a solution to the KKT conditions, by construction.

(II) Alternatively, suppose there exists a solution $\{b_i\}_{\bar{m}+1,\dots,m}$ and $\{c_i\}_{\bar{n}+1,\dots,n}$ and $\boldsymbol{\mu}$ to the KKT conditions such that $\mathbf{a}_j^\intercal \mathbf{x}^* = b_j$ for all $j = \bar{m}+1, \dots, m$. Consider there exists a vector

$\boldsymbol{\gamma}$ that satisfies $\boldsymbol{\gamma} \succeq \mathbf{0}$ and $\mathbf{A}_{21}^\intercal \boldsymbol{\gamma} = \mathbf{0}$. Compute $\epsilon \geq 0$ as: $\epsilon = \min \left(\frac{\mu_k}{\gamma_k}\right)_{\gamma_k > 0, k = \bar{m}+1,\ldots,m}$. Then, we construct $\boldsymbol{\mu}' := \boldsymbol{\mu} - \epsilon \left[\begin{smallmatrix}\mathbf{0}\\ \boldsymbol{\gamma}\end{smallmatrix}\right]$ that satisfies $\boldsymbol{\mu}' \succeq \mathbf{0}$ and $\mu_i' = 0$ for some $\bar{m}+1 \leq i \leq m$ that is the argument of the above minimum.

Furthermore, define $\mathbf{c}' := \mathbf{c}_\mathcal{A} + \epsilon \left[\begin{smallmatrix}\mathbf{0}\\ \mathbf{A}_{22}^\intercal \boldsymbol{\gamma}\end{smallmatrix}\right]$ and $\mathbf{b}'$ such that $b_j' := b_j$ for all $j \neq i$ and b_i' to be any value $b_i' > b_i$. Then $(\mathbf{b}', \mathbf{c}', \boldsymbol{\mu}')$ is also a solution to the KKT conditions. More specifically, the complementarity slackness condition holds for all $j = \bar{m}+1, \ldots, m$:

$$\begin{aligned} &\mu_j' \; (a_j^\intercal x^* - b_j') = \mu_j' \underbrace{(a_j^\intercal x^* - b_j)}_{=0} = 0, \quad j \neq i \\ &\underbrace{\mu_i'}_{=0} (a_i^\intercal x^* - b_i') = 0, \qquad\qquad\qquad j = i. \end{aligned} \tag{B.3.1}$$

Then we can check the gradient condition:

$$\begin{aligned} &\mathbf{Q}_\mathcal{C} \mathbf{x}^* + \mathbf{A}_\mathcal{C}^\intercal \boldsymbol{\mu}' + \mathbf{c}' = \mathbf{Q}_\mathcal{C} \mathbf{x}^* + \mathbf{A}_\mathcal{C}^\intercal \left(\boldsymbol{\mu} - \epsilon \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\gamma} \end{bmatrix} \right) + \left(\mathbf{c} + \epsilon \begin{bmatrix} \mathbf{0} \\ \mathbf{A}_{22}^\intercal \boldsymbol{\gamma} \end{bmatrix} \right) \\ &= \mathbf{Q}_\mathcal{C} \mathbf{x}^* + \mathbf{A}_\mathcal{C}^\intercal \boldsymbol{\mu} + \mathbf{c}_\mathcal{A} = 0. \end{aligned} \tag{B.3.2}$$

Hence, $\mathbf{b}' \neq \mathbf{b}_\mathcal{A}$ satisfies the KKT conditions and the coalition cannot uniquely determine $\mathbf{b}_\mathcal{A}$.

Proof of (2). Consider a solution $\{b_i\}_{\bar{m}+1,\ldots,m}$ and $\{c_i\}_{\bar{n}+1,\ldots,n}$ and $\boldsymbol{\mu}$ to the KKT conditions. For some $\epsilon > 0$ define $\boldsymbol{\mu}' := \boldsymbol{\mu} + \epsilon \left[\begin{smallmatrix}\mathbf{0}\\ \boldsymbol{\gamma}\end{smallmatrix}\right]$ and $\mathbf{c}' := \mathbf{c}_\mathcal{A} - \epsilon \left[\begin{smallmatrix}\mathbf{0}\\ \mathbf{A}_{22}^\intercal \boldsymbol{\gamma}\end{smallmatrix}\right]$. Define $\mathbf{b}'$ such that for all j, it holds that $b_j' = \mathbf{a}_j^\intercal \mathbf{x}^*$. Then $(\mathbf{b}', \mathbf{c}', \boldsymbol{\mu}')$ is also a solution to the KKT conditions. Specifically, it follows that $\boldsymbol{\mu}' \geq \mathbf{0}$. Moreover the complementarity slackness condition holds by construction of $\mathbf{b}'$, and as before the gradient condition holds. Hence, $\mathbf{c}' \neq \mathbf{c}_\mathcal{A}$ satisfies the KKT solution, and the coalition cannot uniquely determine $\mathbf{c}_\mathcal{A}$. □

B.4 Proof of Lemma 3.3.3

Let $\mathbf{y}^k := \mathbf{x}^k + \mathbf{w}^{k-1}$. Manipulating (3.3.3), we get:

$$\mathbf{y}^{k+1} = (\mathbf{I} - \rho \mathbf{M}^{-1}) \mathbf{y}^k + (2\rho \mathbf{M}^{-1} - \mathbf{I}) \mathbf{z}^k + \mathbf{n} = \rho \mathbf{M}^{-1} \mathbf{y}^k + (2\rho \mathbf{M}^{-1} - \mathbf{I})(\mathbf{z}^k - \mathbf{y}^k) + \mathbf{n}.$$

The expression of the thresholding operator gives the following bound: $-\lambda/\rho \leq \mathbf{z}_i^k - \mathbf{y}_i^k \leq \lambda/\rho$, for $i = 1, \dots, n$. Then, using the triangle inequality and submultiplicative property:

$$
\begin{aligned}
\|\mathbf{y}^{k+1}\|_2 &\leq \|\rho\mathbf{M}^{-1}\mathbf{y}^k\|_2 + \|(2\rho\mathbf{M}^{-1} - \mathbf{I})(\mathbf{z}^k - \mathbf{y}^k)\|_2 + \|\mathbf{n}\|_2 \\
&\leq \|\rho\mathbf{M}^{-1}\|_2\|\mathbf{y}^k\|_2 + \sqrt{n}\lambda/\rho\|2\rho\mathbf{M}^{-1} - \mathbf{I}\|_2 + \|\mathbf{n}\|_2.
\end{aligned}
$$

We select $\mathbf{z}^0 = \mathbf{w}^0 = \mathbf{0}$, compress the geometric progression and use $\|\mathbf{y}^{k+1}\|_\infty \leq \|\mathbf{y}^{k+1}\|_2$, getting the bounds in (3.3.4). □

B.5 Proof of Theorem 3.3.4

We use two theoretical results in the proof. First, we need the underlying homomorphic encryption scheme, CKKS, to be semantically secure, in order to ensure that an adversary that does not have access to the private key of the scheme cannot decrypt or even distinguish ciphertexts encrypting different values. This has been proven in [62] and assumes that the Decisional Ring Learning with Errors problem is computationally hard [156]. We select the scheme parameters to ensure that this problem is hard in practice, specifically that it achieves a security level of 128 bits, according to the Learning with Errors estimator of [4]. Second, we need that the interactive part of the multi-party CKKS scheme, in our case, the distributed bootstrapping protocol, preserves the indistinguishability of the ciphertexts and does not reveal the private key. This has been proven in [96] and assumes that at least one servers is honest and that the masks used in DBoot are statistically hiding. Our adversarial model indeed considers that at most $K-1$ servers can be corrupted and we choose the masks to be 80 bits larger than the messages, while ensuring enough levels such that the result does not overflow.

In Protocol 3.3.1, the data of the client is sent encrypted to the servers. Since the servers cannot all collude in order to reveal the private key, and no information about the messages underlying is leaked by viewing or computing on the respective ciphertexts, Protocol 3.3.1 achieves client data confidentiality. □

Appendix C

Technical details for Chapter 4

C.1 Proof of Theorem 4.3.2

We are going to treat two cases: I, the adversary does not corrupt the aggregator and II, the adversary corrupts the aggregator:

$$\Pr[b' = b] = \frac{1}{2}\Pr[b' = b | i \notin \mathcal{C}] + \frac{1}{2}\Pr[b' = b | i \in \mathcal{C}].$$

We will consider the stronger case where $\mathcal{S}^* = \mathcal{U}^*$; the weaker case where $\mathcal{S}^* \subseteq \mathcal{U}^*$ follows.

I. $a \notin \mathcal{C}$. From the compromise queries, the adversary holds the following information $\{\kappa, \mathrm{pk}, \{s_i(t)\}_{i\in\mathcal{C}}, \{w_i\}_{i\in\mathcal{C}}\}_{t\in[T]}$ and $\sum_{i\in\mathcal{U}} s_i(t) = -\sum_{i\in\mathcal{C}} s_i(t)$, for all $t \in [T]$. From the encryption queries at time t, the adversary knows $\{c_i(t) = \mathrm{E}(w_i^{\mathcal{A}} x_i(t)) + s_i(t))\}_{i\in\mathcal{E}(t)}$. Then, the adversary chooses $t^* \in T$ and a series of $\{x_i^0(t^*)\}_{i\in\mathcal{U}^*}$ and $\{x_i^1(t^*)\}_{i\in\mathcal{U}^*}$ and receives from the challenger $\{c_i(t^*) = \mathrm{E}(w_i^{\mathcal{A},b} x_i^b(t^*)) + s_i(t^*))\}_{i\in\mathcal{U}^*}$.

Because the adversary doesn't have the secret key of the Paillier scheme and does not have the individual secrets of the uncorrupted agents, the following holds, where $\eta_1(\kappa), \eta_2(\kappa)$

are negligible functions, according to Theorems 2.5.2 and 2.4.1:

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ breaks Paillier scheme}] \leq \eta_1(\kappa),\\
&\Pr[\mathcal{A} \text{ breaks secret sharing}] \leq \eta_2(\kappa),\\
&\Pr[b' = b | i \notin \mathcal{C}] \leq \frac{1}{2} + \eta_1(\kappa)\eta_2(\kappa).
\end{aligned} \tag{C.1.1}$$

II. $a \in \mathcal{C}$. From the compromise queries, the adversary holds the following information $\forall t \in [T]$: $\{\kappa, \mathrm{pk}, \{s_i(t)\}_{j\in\mathcal{C}}, \{w_i\}_{i\in\mathcal{C}}, \mathrm{sk}\}_{t\in[T]}$, and $\sum_{i\in\mathcal{U}} s_i(t) = -\sum_{i\in\mathcal{C}} s_i(t)$. From the encryption queries, and after using sk to decrypt, the adversary knows $\{p_i(t) = w_i^{\mathcal{A}} x(t) + s_i(t)\}_{i\in\mathcal{E}(t)}$. Then, the adversary chooses $t^* \in T$ and a series of $\{x_i^0(t^*)\}_{i\in\mathcal{U}^*}$ and $\{x_i^1(t^*)\}_{i\in\mathcal{U}^*}$, such that $\sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},0} x_i^0(t^*) = \sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},1} x_i^1(t^*)$ and receives from the challenger $\{c_i(t^*) = \mathrm{E}(w_i^{\mathcal{A},b} x_i^b(t^*)) + s_i(t^*)\}_{i\in\mathcal{U}^*}$. The adversary uses the secret key of the Paillier scheme to decrypt the individual ciphertexts and obtains $p_i(t^*) = w_i^{\mathcal{A},b} x_i^b(t^*) + s_i(t^*) \mod N$, for $i \in \mathcal{U}^*$. Because the secret shares of zero are different for each time $t \neq t^*$, the adversary cannot infer information about the challenge query from the previous encryption queries.

Then, the probability that the adversary wins is the probability that the adversary breaks secret sharing:

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ breaks secret sharing}] \leq \eta_2(\kappa),\\
&\Pr[b' = b | i \in \mathcal{C}] \leq \frac{1}{2} + \eta_2(\kappa).
\end{aligned} \tag{C.1.2}$$

From (C.1.1) and (C.1.2): $\mathbf{Adv}^{\mathrm{pWSAh}}(\mathcal{A}) \leq \eta_2(\kappa)$. □

C.2 Further discussion on the secret shares in pWSAh

In this part, we will discuss some details from the pWSAh scheme in Section 4.3.2. There, we require that $s_i(t) \in \mathbb{Z}_N$. With a small probability, $s_i(t) \in \mathbb{Z}_N \backslash \mathbb{Z}_N^*$, i.e., it is a multiple of p or q. Say $s_i(t)$ is a multiple of q. This means that $ps_i(t) = 0 \mod N$. In that case, a strategy that an adversary that has corrupted the aggregator can try is to multiply $w_i x_i(t) + s_i(t)$ by p and then try to find if there is an element in $\mathbb{Z}_N$ which multiplied by p gives the same answer, which has complexity approximately $O(p)$:

$$\gamma_i(t) := w_i x_i(t) + s_i(t) \bmod N$$
$$p\gamma_i(t) = p w_i x_i(t) \bmod N, \text{ if } s_i(t) \bmod q = 0.$$

Then, the probability that the adversary decrypts $w_i x_i(t)$ is:

$$\Pr\left[b' = b | i \in \mathcal{C}\right] = \frac{N - \phi(N)}{N} \frac{1}{O(p) + O(q)} = \frac{p+q}{N} \frac{1}{O(p) + O(q)} \leq \frac{1}{N},$$

which is a negligible function in the security parameter κ.

The above bound holds when plaintexts are from $\mathbb{Z}_N$, and not only from a smaller subgroup. In this section, we made no assumptions about the local data. However, in Section 4.4.2.3 we need plaintexts to be on l bits, where l will almost never be larger than 128, which is less than the number of bits of p and q. Then, solving $pz = \gamma \bmod N$ for some z on l bits and known γ, is easy – it can essentially be solved in $\mathbb{Z}$, as there is no wrap-around. In this case:

$$\Pr\left[b' = b | i \in \mathcal{C}\right] \leq \frac{p+q}{N} \approx \frac{2}{\min(p, q)}.$$

The aggregator could try the same trick on all combinations of partial sums from the agents. In this case, the probability that the aggregator learns any partial information is $2/(\min(p, q) - M)$. Nevertheless, for the current acceptable value of bit size for N, which is 2048 bits, this scheme would still give around 1000 bits of security, which is substantially larger than what the Decisional Composite Residue gives, or even the possible values the local data can take.

Nevertheless, we can introduce an extra round of communication to ensure $s_i(t) \in \mathbb{Z}_N^*$: agent $i \in [M]$ verifies if $\gcd(s_i(t), N) = 1$ and if not, it changes the values $\sigma_{ii}(t)$, $\sigma_{ai}(t)$ so that (4.3.4) is still satisfied and $\gcd(s_i(t), N) = 1$, then sends the new $\sigma_{ai}(t)$ to the aggregator. This works because $s_a(t)$ is not used for masking, so it is not required to be in $\mathbb{Z}_N^*$.

C.3 Technical details for packed pWSA

We detail the steps for packing the pWSA scheme from Section 4.4.2.5 and depict the bit gains due to the operations performed on the packed columns in Figure C.1.

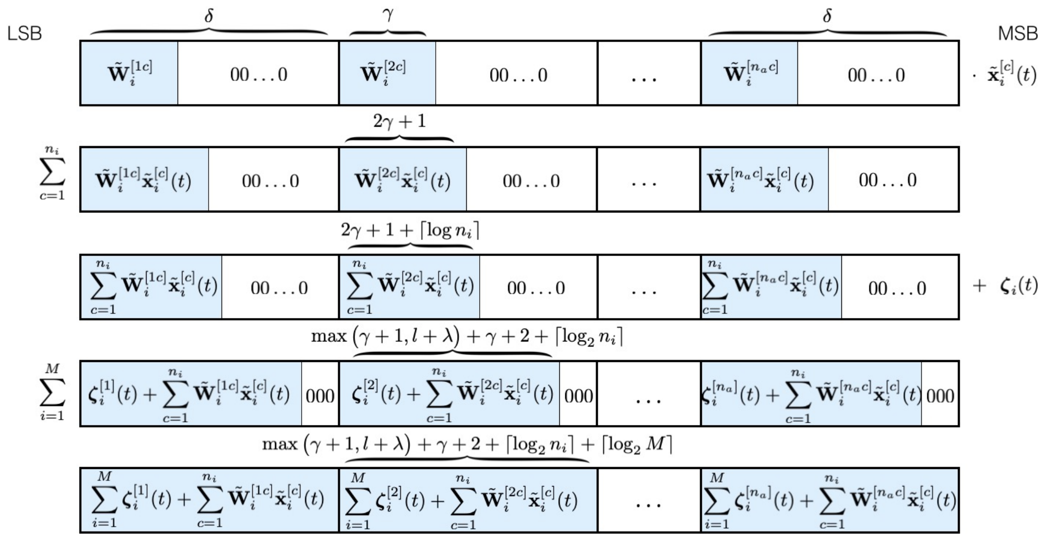

LSB
MSB
δ
γ
δ
$\tilde{\mathbf{W}}_i^{[1c]}$
$00\dots0$
$\tilde{\mathbf{W}}_i^{[2c]}$
$\tilde{\mathbf{W}}_i^{[n_a c]}$
$\cdot\ \tilde{\mathbf{x}}_i^{[c]}(t)$
$2\gamma+1$
$\sum_{c=1}^{n_i}$
$\tilde{\mathbf{W}}_i^{[1c]}\tilde{\mathbf{x}}_i^{[c]}(t)$
$2\gamma+1+\lceil\log n_i\rceil$
$\sum_{c=1}^{n_i}\tilde{\mathbf{W}}_i^{[1c]}\tilde{\mathbf{x}}_i^{[c]}(t)$
$+\ \boldsymbol{\zeta}_i(t)$
$\max(\gamma+1, l+\lambda)+\gamma+2+\lceil\log_2 n_i\rceil$
$\sum_{i=1}^{M}$
$\boldsymbol{\zeta}_i^{[1]}(t)+\sum_{c=1}^{n_i}\tilde{\mathbf{W}}_i^{[1c]}\tilde{\mathbf{x}}_i^{[c]}(t)$
000
$\max(\gamma+1, l+\lambda)+\gamma+2+\lceil\log_2 n_i\rceil+\lceil\log_2 M\rceil$
$\sum_{i=1}^{M}\boldsymbol{\zeta}_i^{[1]}(t)+\sum_{c=1}^{n_i}\tilde{\mathbf{W}}_i^{[1c]}\tilde{\mathbf{x}}_i^{[c]}(t)$
$\dots$

Figure C.1: The operations performed on the packed columns and the corresponding number of bits of the result, where on the third line $\boldsymbol{\zeta}_i(t) = \left[\boldsymbol{\zeta}_i^{[1]}(t)\, 0\dots0\, \boldsymbol{\zeta}_i^{[2]}(t)\, \dots\, \boldsymbol{\zeta}_i^{[n_a]}(t)\, 0\dots0\right]$ and $\boldsymbol{\zeta}_i^{[k]}(t) = \mathbf{s}_i^{[k]}(t) + 2^\gamma \mathbf{z}_i^{[k]}(t)$, for $k \in [n_a]$.

1) Prepare the values to be packed as per (4.4.2), (4.4.3), with γ to be determined later.

2) Construct the packed plaintext p_i^c; encrypt it in $\mathrm{E}(\mathbf{W}_i^c)$:

$$p_i^c = \sum_{k=1}^{n_a} \tilde{\mathbf{W}}_i^{[kc]} 2^{(k-1)\delta}.$$

3) Multiply the encrypted column $\mathrm{E}(\mathbf{W}_i^c)$ by a pre-processed scalar $\tilde{\mathbf{x}}_i^{[c]}(t)$. One slot of the ciphertext will now contain (C.3.1) and will be represented on $2\gamma+2$ bits:

$$\left(\overline{\mathbf{W}}_i^{[kc]} + 2^\gamma\right)\left(\overline{\mathbf{x}}_i^{[c]}(t) + 2^\gamma\right) = \overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t) + 2^{2\gamma} + 2^\gamma\left(\overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t)\right). \tag{C.3.1}$$

From (C.3.1), one can retrieve the desired result $\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$ by taking the lefthand side modulo 2^γ. But one can also obtain:

$$\overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t) = \left\lfloor (\tilde{\mathbf{W}}_i^{[kc]}\tilde{\mathbf{x}}_i^{[c]}(t) - 2^{2\gamma})//2^\gamma \right\rfloor, \tag{C.3.2}$$

which can reveal intermediate information to the decryptor. In order to avoid this information leakage, we need to artificially add some noise $\mathbf{z}_i^{[kc]}(t)$ that still allows retrieving $\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$. It is more efficient to add this noise in step 5), after summing over n_i.

4) Sum the n_i ciphertexts to output a ciphertext that contains the vector result of the matrix-vector multiplication product.

5) For each slot $k \in [n_a]$, an agent $i \in [M]$ selects $\mathbf{z}_i^{[k]}(t) \in (0, 2^{l+1+\lambda+\lceil \log_2 n_i \rceil})$, such that a statistical security of λ bits is guaranteed for the private value $\sum_{c=1}^{n_i} \overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t)$. Then, each agent constructs its ciphertext $c_i(t)$ by adding the secret shares of zero such that the remaining private value $\sum_{c=1}^{n_i} \overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$ is concealed:

$$\begin{aligned} c_i(t) := \mathbf{s}_i^{[k]}(t) + 2^\gamma \mathbf{z}_i^{[k]}(t) + \sum_{c=1}^{n_i} \tilde{\mathbf{W}}_i^{[kc]}\tilde{\mathbf{x}}_i^{[c]}(t) \overset{\text{(C.3.1)}}{=} n_i 2^{2\gamma} + \mathbf{s}_i^{[k]}(t) + \\ + \sum_{c=1}^{n_i} \overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t) + 2^\gamma \Big(\mathbf{z}_i^{[k]}(t) + \sum_{c=1}^{n_i} \overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t) \Big). \end{aligned} \tag{C.3.3}$$

Due to the masking with $2^\gamma z_i^{[k]}(t)$, we can reduce the size of the mask $\mathbf{s}_i^{[k]}(t)$. More specifically, $\mathbf{s}_i^{[k]}(t)$ can be sampled uniformly at random from $(0, 2^\gamma)$, because it acts like a one-time pad (perfect secrecy) on $\sum_{c=1}^{n_i} \overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$ once the decryptor takes equation (C.3.3) modulo 2^γ. The whole quantity $\mathbf{s}_i^{[k]}(t) + 2^\gamma \mathbf{z}_i^{[k]}(t)$ is used for masking, but we only need to ensure that $\sum_{i \in [M] \cup a} \mathbf{s}_i^{[k]}(t) = 0$.

6) The aggregator obtains $c(t)$ by taking the product of all ciphertexts $c_i(t)$ it received.

7) The aggregator decrypts the ciphertext $c(t)$, adds its own secret share of zero $\mathbf{s}_a(t)$. It then retrieves the n_a elements of $\tilde{\mathbf{x}}_a(t)$ by recursively taking the quotient and rest by 2^δ, and from each resulting element, it obtains the elements of $\mathbf{x}_a(t)$ by taking modulo 2^γ and dividing by 2^{2l_f}.

We now compute the number of bits and corresponding padding one slot can have such that no overflow occurs during the private weighted sum aggregation. We assume that the values of n_i, for $i \in [M]$ are similar and define $n := \max\limits_{i \in [M]} n_i$. The value that is packed in slot $k \in [n_a]$ after step 6) is:

$$\sum_{i=1}^{M} \left(\mathbf{s}_i^{[k]}(t) + 2^{\gamma} \mathbf{z}_i^{[k]}(t) + \sum_{c=1}^{n_i} \tilde{\mathbf{W}}_i^{[kc]} \tilde{\mathbf{x}}_i^{[c]}(t) \right) < 2^{\delta},$$

from which we obtain that:

$$\delta > \max\big(\gamma + 1, l + \lambda\big) + \lceil \log_2 n \rceil + \lceil \log_2 M \rceil + \gamma + 2.$$

For the correct retrieval of the desired result, we require that:

$$\sum_{i=1}^{M} \sum_{c=1}^{n_i} \overline{\mathbf{W}}_i^{[kc]} \overline{\mathbf{x}}_i^{[c]}(t) < 2^{\gamma}. \tag{C.3.4}$$

From (C.3.4), we choose $\gamma = 2l + 1 + \lceil \log_2 n \rceil + \lceil \log_2 M \rceil$ and:

$$\delta = \max\big(l + 2 + \lceil \log_2 n \rceil + \lceil \log_2 M \rceil, \lambda\big) + 3l + 4 + 2(\lceil \log_2 n \rceil + \lceil \log_2 M \rceil). \tag{C.3.5}$$

Proof of Theorem 4.4.1. The correctness follows from the appropriate padding and packing to avoid overflow, as stated in (C.3.5). The aggregator obliviousness proof follows from Theorem 4.3.2, along with the intermediate values masking from (C.3.3). $\square$

C.4 Proof of Theorem 4.5.3

Correctness: The aggregator obtains the following:

$$\mathbf{x}_a(t) = \mathcal{D}_{\mathbf{sk}}\Big(\mathbf{G}\mathbf{e}(t) \bmod q\Big) = \mathcal{D}_{\mathbf{sk}}\Big(\mathbf{G}\,(\mathbf{c}(t) + \mathbf{s}_a^{\intercal}\mathbf{A}_t) \bmod q\Big) \tag{C.4.1}$$

$$= \mathcal{D}_{\mathbf{sk}}\Big(\mathbf{G}\Big(\sum_{i \in [M]} \mathbf{c}_i(t) + \mathbf{s}_a^{\intercal}\mathbf{A}_t\Big) \bmod q\Big) \tag{C.4.2}$$

$$= \mathcal{D}_{\mathbf{sk}}\Big(\mathbf{G}\Big(\sum_{i \in [M]} \mathbf{s}_i^{\intercal}\mathbf{A}_t + \mathbf{e}_i(t)^{\intercal} + \mathbf{s}_a^{\intercal}\mathbf{A}_t\Big) \bmod q\Big) \tag{C.4.3}$$

$$= \mathcal{D}_{\mathbf{sk}}\Big(\mathbf{G}\Big(\sum_{i\in[M]} \mathbf{e}_i(t)^\intercal\Big) \bmod q\Big) \tag{C.4.4}$$

$$= \mathcal{D}_{\mathbf{sk}}\Big(\sum_{i\in[M]} \mathbf{G}\mathbf{e}_i(t)^\intercal \bmod q\Big) = \mathcal{D}_{\mathbf{sk}}\Big(\sum_{i\in[M]} \mathbf{y}_i\Big) \tag{C.4.5}$$

$$= \mathcal{D}_{\mathbf{sk}}\Big(\sum_{i\in[M]} \mathcal{E}_{\mathbf{pk}}\left(w_i x_i(t)\right)\Big) = \sum_{i\in[M]} w_i x_i. \tag{C.4.6}$$

The correctness of the result follows from construction: (C.4.4) follows from (C.4.3) because the keys were selected such that $\mathbf{s}_a = -\sum_{i\in[M]} \mathbf{s}_i$, (C.4.5) follows from (C.4.4) due to the linearity of $\mathbf{G}$, and the correct transition from the transition from (C.4.1) to (C.4.6) is allowed by the fact that the PAHE scheme satisfies $\mathcal{D}_{\mathbf{sk}}(\mathbf{c}_1 + \mathbf{c}_2) = \mathcal{D}_{\mathbf{sk}}(\mathbf{c}_1) + \mathcal{D}_{\mathbf{sk}}(\mathbf{c}_2)$.

From the setup phase, the adversary learns the public parameters $\mathbf{A}_{t\in[T]}, q, \lambda, \kappa, \sigma, M$, $\mathbf{g}^\intercal, \mathbf{pk}$ and constructs $\mathbf{G}$.

We treat two cases: I, the adversary does not corrupt the aggregator and II, the adversary corrupts the aggregator:

$$\Pr[b' = b] = \frac{1}{2}\Pr[b' = b | a \notin \mathcal{C}] + \frac{1}{2}\Pr[b' = b | a \in \mathcal{C}].$$

From the compromise queries, the adversary holds the following information:
I. $a \notin \mathcal{C}$. $\{\mathbf{s}_i\}_{i\in\mathcal{C}}, \{\mathcal{E}_{\mathbf{pk}}(w_i)\}_{i\in\mathcal{C}}$ and $\sum_{i\in\mathcal{U}} \mathbf{s}_i = -\sum_{i\in\mathcal{C}} \mathbf{s}_i$.
II. $a \in \mathcal{C}$. $\mathbf{sk}, \{\mathbf{s}_i\}_{i\in\mathcal{C}}, \{w_i\}_{i\in\mathcal{C}}$ and $\sum_{i\in\mathcal{U}} \mathbf{s}_i = -\sum_{i\in\mathcal{C}} \mathbf{s}_i$.

Because $\mathbf{A}_t$ is different at every time step, the adversary cannot obtain meaningful information from $\mathbf{c}_i(t_1) - \mathbf{c}_i(t_2)$.

From the encryption queries at time t, for which the adversary submits $(i \in \mathcal{E}(t), t, w_i^{\mathcal{A}}$, $x_i(t))$, where $w_i^{\mathcal{A}} = \{w_i^{\mathcal{A},0}, w_i^{\mathcal{A},1}\}$, the adversary knows $\{\mathbf{c}_i(t) = \mathbf{s}_i^\intercal \mathbf{A}_t + \mathbf{e}_i(t)^\intercal\}_{i\in\mathcal{E}(t)}$, such that $\mathbf{G}\mathbf{e}_i(t) \bmod q \equiv \mathcal{E}_{\mathbf{pk}}(w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t))$ is satisfied for $i \in \mathcal{E}(t)$. Because PAHE is not a deterministic encryption, the adversary cannot recover information about $\mathbf{s}_i$ from computing $\mathbf{c}_i(t) - \mathbf{G}^\dagger \mathcal{E}_{\mathbf{pk}}(w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t)) \neq \mathbf{s}_i^\intercal \mathbf{A}_t$.

In the challenge phase, the adversary chooses $t^* \in T$ and a series of $\{x_i^0(t^*)\}_{i\in\mathcal{U}^*}$ and $\{x_i^1(t^*)\}_{i\in\mathcal{U}^*}$, $w_i^{\mathcal{A},0}$ and $w_i^{\mathcal{A},1}$.

II. $a \in \mathcal{C}$. $\sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},0} x_i^0(t^*) = \sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},1} x_i^1(t^*)$.

The challenger picks a random bit b and sends $\{\mathbf{c}_i(t^*) = \mathbf{s}_i^\intercal \mathbf{A}_{t^*} + \mathbf{e}_i^b(t^*)^\intercal\}_{i\in\mathcal{U}^*}$ to the adversary, such that $\mathbf{G}\mathbf{e}_i^b(t^*) \bmod q \equiv \mathcal{E}_{\mathbf{pk}}(w_i^{\mathcal{A},b} x_i^b(t^*))$ holds for $i \in \mathcal{U}^*$. The adversary does not have the individual secrets of the uncorrupted agents so it cannot recover the individual error terms or keys from $\{\mathbf{c}_i(t^*)\}_{i\in\mathcal{U}^*}$ due to the hardness of A-LWE.

The adversary can sum all the ciphertexts from the encrypted queries and uncompromised set at time t^*, along with the keys from the compromised queries:

$$\sum_{i\in\mathcal{E}(t^*)} \mathbf{s}_i^\intercal \mathbf{A}_t + \mathbf{e}_i(t^*)^\intercal + \sum_{i\in\mathcal{U}(t^*)} \mathbf{s}_i^\intercal \mathbf{A}_t + \mathbf{e}_i^b(t)^\intercal + \Big(\sum_{i\in\mathcal{C}} \mathbf{s}_i\Big)^\intercal \mathbf{A}_{t^*} = \sum_{i\in\mathcal{E}(t^*)} \mathbf{e}_i(t^*)^\intercal + \sum_{i\in\mathcal{U}(t^*)} \mathbf{e}_i^b(t)^\intercal. \tag{C.4.7}$$

Multiplying by $\mathbf{G}$, the adversary obtains:

$$\mathbf{y} = \mathcal{E}_{\mathbf{pk}}\bigg(\sum_{i\in\mathcal{E}(t^*)} w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t^*) + \sum_{i\in\mathcal{U}(t^*)} w_i^{\mathcal{A},b} x_i^b(t^*)\bigg). \tag{C.4.8}$$

II. $a \in \mathcal{C}$. The adversary uses the aggregator's key $\mathbf{sk}$ to decrypt (C.4.8) and obtains $p(t^*) = \sum_{i\in\mathcal{E}(t^*)} w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t^*) + \sum_{i\in\mathcal{U}(t^*)} w_i^{\mathcal{A},b} x_i^b(t^*)$. Because $\sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},0} x_i^0(t^*) = \sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},1} x_i^1(t^*)$, $p(t^*)$ does not reveal any information.

Then, for both I and II, the probability that the adversary wins is the probability that it solves the A-LWE problem:

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ solves A-LWE problem }] \leq \eta(\lambda),\\
&\Pr[b' = b | i \notin \mathcal{C}] \leq \frac{1}{2} + \eta(\lambda), \quad \Pr[b' = b | i \in \mathcal{C}] \leq \frac{1}{2} + \eta(\lambda).
\end{aligned}$$

where $\eta(\lambda)$ is a negligible function, according to Theorem 2 in [88], Theorem 1 in [28] and the semantic security of the PAHE scheme.

This results in $\mathbf{Adv}^{\text{pWSA}}(\mathcal{A}) \leq \eta(\lambda)$. □

Appendix D

Technical details for Chapter 5

D.1 Further details on LabHE

Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the identity function and $\tau \in \{0,1\}^*$ be a label. Denote the identity program for input label τ by $\mathcal{I}_\tau = (f_{id}, \tau)$. Any labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ (as in Definition 2.5.5) can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

The definitions of correctness, semantic security, circuit privacy and context-hiding are taken from [27, 55].

Definition D.1.1. (Correctness) A multi-user Labeled Homomorphic Encryption scheme correctly evaluates a family of functions $\mathcal{F}$ if for all honestly generated keys $(\mathrm{mpk}, \mathrm{msk}) \overset{\$}{\leftarrow} \hat{\mathrm{Init}}(1^\lambda)$, all user keys $(\mathrm{upk_k}, \mathrm{usk_k}) \overset{\$}{\leftarrow} \hat{\mathrm{KeyGen}}(\mathrm{mpk})$, $k \in [l]$, for all $f \in \mathcal{F}$, all labels $\tau_1, \ldots, \tau_t \in \mathcal{L}$, messages $m_1, \ldots, m_t \in \mathcal{M}$, any $C_i \leftarrow \hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{usk}_{j_i}, \tau_i, m_i)$, $\forall i \in [t], j_i \in [l]$ and $\tilde{g} \leftarrow (f, \hat{\mathrm{Eval}}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, (f, \tau_1, \ldots, \tau_t)))$:

$$\Pr\left[\hat{\mathrm{D}}(\text{msk,upk}, \mathcal{P}, \hat{\mathrm{Eval}}_2(\mathrm{mpk}, \tilde{g}, C_1, \ldots, C_t)) = f(m_1, \ldots, m_t)\right] > 1 - \mathrm{negl}(\lambda),$$

where the probability is taken over the random choices.

Proof. (Theorem 5.1.2) The proof can be easily extended from the proof of correctness of LabHE. The correctness of the eLabHE scheme holds as long as $f(m_1, \ldots, m_t) \in \mathcal{M}$

and $\prod_{i\in[d]} b_i \in \mathcal{M}$, where d is the degree of f. Consider ciphertexts produced by the $\hat{\mathrm{E}}$ primitive. For every $(\tau, m) \in \mathcal{L} \times \mathcal{M}$, the encryption primitive yields $\hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{upk}, \tau, m) \to C = (a, \beta)$, where $a = m - f_{id}(F(\mathrm{usk}, \tau))$ and it follows by the correctness of AHE that $m \leftarrow \hat{\mathrm{D}}(\mathrm{msk}, \mathcal{I}_\tau, C)$.

Consider ciphertexts produced by $\hat{\mathrm{Eval}}_2$. For $i \in [t]$, consider any t labeled programs $\mathcal{P}_i = \big(f_i, \tau_1^{(i)}, \ldots, \tau_{t_i}^{(i)}\big)$ and t ciphertexts C_i such that $m_i \leftarrow \hat{\mathrm{D}}(msk, usk, \mathcal{P}_i, C_i)$. Then, for any $f \in \mathcal{F}$, we want to evaluate $f(\mathcal{P}_1, \ldots, \mathcal{P}_t) =: \mathcal{P}^*$ on ciphertexts $C_1, \ldots, C_t$. For that we run $\hat{\mathrm{Eval}}_1$ and obtain $\tilde{g} \leftarrow \big(f, \hat{\mathrm{Eval}}_1\big(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \{\tau_1^{(i_j)}, \ldots, \tau_{t_{i_j}}^{(i_j)}\}_{j\in[t]}\big)\big)$. Denote the resulting ciphertext as $C \leftarrow \hat{\mathrm{Eval}}_2(\mathrm{mpk}, \tilde{g}, C_1, \ldots, C_t)$. By construction, a ciphertext C_i is either $(m_i - b_i, [[b_i]])$ or $[[m_i - b_i]]$, with $b_i \leftarrow f_i\big(F\big(\mathrm{usk}, \tau_1^{(i)}\big), \ldots, F\big(\mathrm{usk}, \tau_{t_i}^{(i)}\big)\big)$, obtained from $\hat{\mathrm{Eval}}_2$ applied to the inputs of $\mathcal{P}_i$ and the corresponding $\tilde{f}_i \leftarrow \big(f_i, \hat{\mathrm{Eval}}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \tau_1^{(i)}, \ldots, \tau_{t_i}^{(i)})\big)$, $i \in [t]$. It is clear that $\tilde{g}$ will contain $\tilde{f}_i$ for $i \in [t]$. After the evaluation of f, we obtain the ciphertext C that is either $(f(m_1, \ldots, m_t) - f(b_1, \ldots, b_t), [[f(b_1, \ldots, b_t)]])$ or $[[f(m_1, \ldots, m_t) - f(b_1, \ldots, b_t)]]$. Then, by construction of $\hat{\mathrm{D}}$, correctness of AHE and using $\mathcal{P}^* = \big(f_1\big(\mathcal{I}_{\tau_1^{(1)}}, \ldots, \mathcal{I}_{\tau_{t_1}^{(1)}}\big), \ldots, f_t\big(\mathcal{I}_{\tau_1^{(t)}}, \ldots, \mathcal{I}_{\tau_{t_t}^{(t)}}\big)\big)$, we obtain the correctness of the eLabHE scheme, i.e., $\hat{\mathrm{D}}(\mathrm{msk}, \mathrm{usk}, \mathcal{P}^*, C) \leftarrow f\big(m_1, \ldots, m_t\big)$. $\square$

The definition of semantic security [107], [105, Ch. 5] is adapted for labeled homomorphic encryption schemes in [27]:

Definition D.1.2. (Semantic security) Let eLabHE = $(\hat{\mathrm{Init}}, \hat{\mathrm{KeyGen}}, \hat{\mathrm{E}}, \hat{\mathrm{Eval}}_1, \hat{\mathrm{Eval}}_2, \hat{\mathrm{D}})$ be a multi-user labeled homomorphic encryption scheme and $\mathcal{A}$ be a PPT adversary. Consider the following experiment where $\mathcal{A}$ is given access to an oracle $\hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{usk}, \cdot, \cdot)$, for $\mathrm{usk} = (\mathrm{usk}_1, \ldots, \mathrm{usk}_l)$ that on input a pair (τ, m) outputs $\hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{usk}, \tau, m)$:

$$\begin{gathered}
\mathbf{Exp}_{\mathrm{eLabHE},\mathcal{A}}(\lambda) : b \xleftarrow{\$} \{0,1\}; (\mathrm{mpk}, \mathrm{msk}) \xleftarrow{\$} \hat{\mathrm{Init}}(1^\lambda) \\
(\mathrm{upk}, \mathrm{usk}) \xleftarrow{\$} \hat{\mathrm{KeyGen}}(\mathrm{mpk}) \\
(m_0, \tau_0^*, m_1, \tau_1^*) \xleftarrow{\$} \mathcal{A}^{\hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{usk}, \cdot, \cdot)}(\mathrm{mpk}, \mathrm{upk}) \\
C \xleftarrow{\$} \hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{usk}, \tau_b^*, m_b)
\end{gathered}$$

$$b' \leftarrow \mathcal{A}^{\hat{\mathrm{E}}(\mathrm{mpk},\mathrm{usk},\cdot,\cdot)}(C)$$

If $b' = b$ return 1. Else, return 0.

We say $\mathcal{A}$ is a legitimate adversary if it queries the encryption oracle on distinct labels (each label τ is never queried more than once) and never on the two challenge labels τ_0^*, τ_1^*. We define the adversary's advantage as $\mathrm{Adv}_{\mathrm{eLabHE},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}_{\mathrm{eLabHE},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}$. Then, we say that eLabHE has semantic security if for any PPT legitimate algorithm $\mathcal{A}$, the following holds $\mathrm{Adv}_{\mathrm{eLabHE},\mathcal{A}}(\lambda) = \mathrm{negl}(\lambda)$.

Proof. (Theorem 5.1.3) The scheme eLabHE has the same encryption and decryption primitives as LabHE and identically looking ciphertexts. Hence, the proof follows the semantic security of the LabHE scheme, which depends on the semantic security of the underlying AHE scheme and on the pseudorandomness of F. □

Semantic security is equivalent to ciphertext indistinguishability [107], [105, Ch. 4], so we can write it as:

$$\mathrm{SD}[\hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{usk}, \tau, m), \hat{\mathrm{Sim}}(1^\lambda, \mathrm{mpk}, \mathrm{usk})] \leq \mathrm{negl}(\lambda),$$

where $\hat{\mathrm{Sim}}$ is a PPT simulator that simply outputs a LabHE (or AHE) encryption of zeros. The same can be written for the secret sharing scheme which has perfect secrecy, with $\hat{\mathrm{Sim}}$ outputting a random value sampled from $\mathcal{M}$.

Definition D.1.3. (Context-hiding) A multi-user labeled homomorphic encryption scheme satisfies context-hiding for a family of functions $\mathcal{F}$ if there exists a PPT simulator $\hat{\mathrm{Sim}}$ and a negligible function $\mathrm{negl}(\lambda)$ such that the following holds: for any $\lambda \in \mathbb{N}$, any pair of master keys $\mathrm{mpk}, \mathrm{msk} \xleftarrow{\$} \hat{\mathrm{Init}}(1^\lambda)$, any l user keys $(\mathrm{upk_k}, \mathrm{usk_k}) \xleftarrow{\$} \hat{\mathrm{KeyGen}}(\mathrm{mpk})$, $k \in [l]$, any function $f \in \mathcal{F}$ with t inputs, any tuple of messages $m_1, \ldots, m_t \in \mathcal{M}$, labels $\tau_1, \ldots, \tau_t \in \mathcal{L}$, and ciphertexts $C_i \xleftarrow{\$} \hat{\mathrm{E}}(\mathrm{mpk}, \mathrm{usk}_{j_i}, \tau_i, m_i)$, $i \in [t]$ and $j_i \in [l]$:

$$\mathrm{SD}[\hat{\mathrm{Eval}}_2(\mathrm{mpk}, \tilde{f}, C_1, \ldots, C_t), \hat{\mathrm{Sim}}(1^\lambda, \mathrm{msk}, \mathrm{upk}, \mathcal{P}, m)] \leq \mathrm{negl}(\lambda),$$

where we defined $\mathcal{P} = (f, \tau_1, \ldots, \tau_t)$, $m = f(m_1, \ldots, m_t)$, $\tilde{f} = (f, \hat{\mathrm{Eval}}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \mathcal{P}))$.

Context-hiding describes the property that a party that decrypts a ciphertext C as $m \leftarrow \hat{\mathrm{D}}(\mathrm{msk}, \mathrm{upk}, \mathcal{P}, C)$, with $\mathcal{P} = (f, \tau_1, \ldots, \tau_t)$, does not find out anything about the inputs $m'_1, \ldots, m'_t$, except for the fact that $m = f(m'_1, \ldots, m'_t)$. In order to prove that eLabHE is context-hiding, we need to make use of the concept of circuit privacy.

For any admissible linear function f, we can abstract the evaluation primitives in AHE as $\mathrm{Eval}(\mathrm{pk}, f, C_1, \ldots, C_t)$. We will use this notation when defining circuit privacy:

Definition D.1.4. (Circuit privacy) A homomorphic encryption scheme is circuit private for a family of circuits $\mathcal{F}$ if there exists a PPT simulator $\hat{\mathrm{Sim}}$ and a negligible function $\mathrm{negl}(\lambda)$ such that the following holds. For any λ, any pair of keys $(\mathrm{pk}, \mathrm{sk}) \overset{\$}{\leftarrow} \mathrm{KeyGen}(1^\lambda)$, any circuit $f \in \mathcal{F}$, any tuple of messages $m_1, \ldots, m_t \in \mathcal{M}$ and $m = f(m_1, \ldots, m_t)$ and ciphertexts $C_1, \ldots, C_t$ satisfying $\forall i \in [t] : C_i \overset{\$}{\leftarrow} \mathrm{E}(\mathrm{pk}, m_i)$, then:

$$\mathrm{SD}\big[\mathrm{Eval}(\mathrm{pk}, f, C_1, \ldots, C_t), \mathrm{Sim}(1^\lambda, \mathrm{pk}, m)\big] \leq \mathrm{negl}(\lambda).$$

The difference between circuit privacy and context-hiding is that in context-hiding, the decryptor has access to the function, whereas in circuit privacy, it does not.

Proof. (Theorem 5.1.4) The proof relies on the circuit privacy of the underlying AHE scheme. Let Sim be the simulator for the circuit privacy of AHE. A simulator $\hat{\mathrm{Sim}}(1^\lambda, \mathrm{msk}, \mathrm{upk}, (f, \tau_1, \ldots, \tau_t), m)$ computes for $\mathrm{upk} = (\mathrm{upk}_1, \ldots, \mathrm{upk}_l)$, $\mathrm{usk}_j \leftarrow \mathrm{D}(\mathrm{msk}, \mathrm{upk}_j)$, $j \in [l]$, and $b_i \leftarrow F(\mathrm{usk}_{j_i}, \tau_i)$, $i \in [t]$. Then, it computes $b \leftarrow f(b_1, \ldots, b_t)$. Notice that $\hat{\mathrm{Sim}}$ has the inputs necessary to compute $\tilde{f}$. If f is a degree-1 polynomial, then $\hat{\mathrm{Sim}}$ outputs $C = (m - b, \mathrm{Sim}(1^\lambda, \mathrm{mpk}, b))$. Else, if f is degree-d, with $d \geq 2$, $\hat{\mathrm{Sim}}$ outputs $C = \mathrm{Sim}(1^\lambda, \mathrm{mpk}, m - b)$. Using the circuit privacy of AHE, the outputs of $\hat{\mathrm{Sim}}$ are distributed identically to the corresponding outputs produced by $\hat{\mathrm{Eval}}_2$. $\square$

D.2 Privacy of Protocol 5.1.3

Proof of Theorem 5.1.5. We can build simulators for Protocol 5.1.3 from the simulators for semantic security, context hiding and perfect privacy and contrast them to the views of the parties, as in Definition 2.2.5.

The view of the actuator is:

$$V_A(f_{LQG}) = \big(f_{LQG}, \mathrm{upk}, \{\mathbf{u}_i\}_{i\in 0\cup[T-1]}, \{\hat{\mathbf{x}}_i - \mathbf{r}_i\}_{i\in[T-1]}, \mathrm{coins}\big).$$

We build a simulator $\hat{\mathrm{Sim}}_A$, that on inputs $\big(1^\lambda, f_{LQR}, \{\mathbf{u}_i\}_{i\in 0\cup[T-1]}\big)$ runs $(\mathrm{mpk}, \mathrm{msk}) \leftarrow \hat{\mathrm{Init}}(1^\lambda), (\widetilde{\mathrm{usk}}, \widetilde{\mathrm{upk}}) \leftarrow \hat{\mathrm{KeyGen}}(\mathrm{mpk})$, samples random values $\{\widetilde{\mathbf{r}}_i\}_{i\in[T-1]} \in \mathcal{M}^n$ and then outputs:

$$\big(f_{LQG}, \widetilde{\mathrm{upk}}, \{\mathbf{u}_i\}_{i\in 0\cup[T-1]}, \{\widetilde{\mathbf{r}}_i\}_{i\in[T-1]}, \widetilde{\mathrm{coins}}\big).$$

The indistinguishability between $V_A(f_{LQG})$ and $\hat{\mathrm{Sim}}_A\big(1^\lambda, f_{LQR}, \{\mathbf{u}_i\}_{i\in 0\cup[T-1]}\big)$ follows from the context-hiding property of LabHE and perfect secrecy of the one-time pad.

The cloud's view is:

$$V_C(\emptyset) = \big(\hat{\mathrm{E}}(\mathbf{x}_0), \hat{\mathrm{E}}(\mathbf{x}_r), \hat{\mathrm{E}}(\mathbf{u}_r), \hat{\mathrm{E}}(\mathbf{\Omega}), \{\hat{\mathrm{E}}(\mathbf{z}_i)\}_{i\in[T]}, \{\hat{\mathrm{E}}(\hat{\mathbf{x}}_i)\}_{i\in[T-1]}, \mathrm{coins}\big),$$

where $\mathbf{\Omega}$ is the collection of $\mathbf{K}, \mathbf{L}, \mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3$. We build a simulator $\hat{\mathrm{Sim}}_C$ that on input (1^λ) runs $\hat{\mathrm{KeyGen}}(1^\lambda)$, generates randomness for the shares, and creates encryptions of random values for all model parameters and signals received from the subsystems and actuators, and outputs:

$$\big(\widetilde{\hat{\mathrm{E}}(\mathbf{x}_0)}, \widetilde{\hat{\mathrm{E}}(\mathbf{x}_r)}, \widetilde{\hat{\mathrm{E}}(\mathbf{u}_r)}, \widetilde{\hat{\mathrm{E}}(\mathbf{\Omega})}, \{\widetilde{\hat{\mathrm{E}}(\mathbf{z}_i)}\}_{i\in[T]}, \{\widetilde{\hat{\mathrm{E}}(\hat{\mathbf{x}}_i)}\}_{i\in[T-1]}, \widetilde{\mathrm{coins}}\big).$$

The indistinguishability between $V_C(\emptyset)$ and $\hat{\mathrm{Sim}}_A(1^\lambda)$ follows from the semantic security of LabHE.

The setup's view is $V_S(\mathbf{K}, \mathbf{L}, \mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3) = (\mathbf{K}, \mathbf{L}, \mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3)$. We build a simulator $\hat{\mathrm{Sim}}_S$ that simply outputs its inputs $(\mathbf{K}, \mathbf{L}, \mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3)$, which is trivially indistinguishable from

$V_S(\mathbf{K}, \mathbf{L}, \mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3)$. Similarly, the view of each subsystems is $V_{S^i}(\mathbf{x}_0^i, \mathbf{x}_r^i, \mathbf{u}_r^i) = (\mathbf{x}_0^i, \mathbf{x}_r^i, \mathbf{u}_r^i)$. We then build simulators $\hat{\mathrm{Sim}}_{S^i}$ that simply output their inputs $(\mathbf{x}_0^i, \mathbf{x}_r^i, \mathbf{u}_r^i)$, being trivially indistinguishable from $V_{S^i}(\mathbf{x}_0^i, \mathbf{x}_r^i, \mathbf{u}_r^i)$.

The correctness of Protocol 5.1.3 follows from the correctness of the LQG algorithm and of the LabHE scheme. The proof is now complete. □

Proof of Theorem 5.1.6. The proof of multi-party privacy uses the fact that neither the cloud nor the actuator can be in a coalition that has all the private data in the system and they cannot extract any information from the communication between themselves: the cloud receives encrypted data from the actuator, and does not have access to the key, and the actuator receives only random shares from the cloud.

A coalition between the cloud and the setup reduces to the case presented in Section 5.1.4.1. Consider now a coalition between the cloud, the setup and $\bar{N}$ subsystems, where $0 \leq \bar{N} < N$. Because the data of the non-colluding subsystems is encrypted with a semantically secure encryption scheme, and the coalition has access neither to the master key nor to the secret key of the non-colluding subsystems, the coalitions cannot infer anything new about the private data of the non-colluding parties.

Similarly, a coalition between the actuator and the setup reduces to the case presented in Section 5.1.4.1. Consider now a coalition between the actuator, the setup and $\bar{N}$ subsystems, where $0 \leq \bar{N} < N$. The outputs of the actuator consist in $\mathbf{u}_k = -\mathbf{K}\hat{\mathbf{x}}_k + \mathbf{K}\mathbf{x}_r + \mathbf{u}_r$, where $\mathbf{K}$ is known because of the collusion with the setup and some entries in $\mathbf{x}_r$ and $\mathbf{u}_r$ are known from the collusion with some subsystems. However, because the communication with the cloud is additively blinded by large random numbers (the secrets from the shares), the actuator cannot infer anything more about $\hat{\mathbf{x}}_k$ than what it can infer solely from the colluding parties' data.

With this observations, one can construct simulators on the inputs of coalitions like in the proof of Theorem 5.1.5. □

Appendix E

Technical details for Chapter 6

E.1 Proof of Theorem 6.2.2

The view of S_1 is composed by its inputs and exchanged messages, and no output. All the messages the first server receives are encrypted. Furthermore, in the oblivious transfer procedure in line 13, an encryption of zero is added to the quantity S_1 receives such that the encryption is re-randomized and S_1 cannot recognize it. Due to the semantic security of the cryptosystems, the view of S_1 is computationally indistinguishable from the view of a simulator which follows the same steps as S_1, but replaces the incoming messages by random encryptions of corresponding length.

The view of S_2 is composed by its inputs and exchanged messages, and no output. Apart from the comparison bits, the latter are always blinded by noise that has at least λ_σ bits more than the private data being sent. For λ_σ chosen appropriately large (e.g. 100 bits [45]), the following is true: $a + r \overset{s}{\equiv} r'$, where a is a value of p bits, r is the noise chosen uniformly at random from $(0, 2^{p+\lambda_\sigma}) \cap \mathbb{Z}_{N_\sigma}$ and r' is a value chosen uniformly at random from $(0, 2^{p+\lambda_\sigma+1}) \cap \mathbb{Z}_{N_\sigma}$. In Protocol 2.7.2, a similar blinding is performed.

Crucially, the noise selected by S_1 is different at each iteration. Hence, S_2 cannot extract any information by combining messages from multiple iterations, as they are always blinded by a different large enough noise. Moreover, the randomization step in line 11 ensures that S_2 cannot infer anything from the values of β_k, as the order of the inputs is unknown. Thus, we construct a simulator that follows the same steps as S_2, but instead of the received messages, it randomly generates values of appropriate length, corresponding to the blinded private values, and random bits corresponding to the comparison bits. The view of such a simulator will be computationally indistinguishable from the view of S_2.

A more detailed proof that explicitly constructs the simulators can be adapted from Appendix B.1. □

E.2 Analysis of errors

Quantization errors

We will use the following observation to investigate the quantization error bounds. Define $\epsilon_a = \bar{a} - a$ and $\epsilon_b = \bar{b} - b$. Then: $\bar{a}\bar{b} - ab = \bar{a}\bar{b} - \bar{a}b + \bar{a}b - ab = \epsilon_a b + \bar{a}\epsilon_b$.

Consider problem (6.2.3) where the coefficients are replaced by the fixed-point representations of the matrices $\mathbf{H}/(cL), \mathbf{F}/(cL)$, of the vector $\mathbf{x}(t)$ and of the set $\mathcal{U}$, but otherwise the iterates $\tilde{\mathbf{U}}_k, \tilde{\mathbf{t}}_k, \tilde{\mathbf{z}}_k$ are real values. Now, consider iteration k of the projected FGM. The errors induced by quantization of the coefficients between the original iterates and the approximation iterates will be:

$$\begin{aligned} \tilde{\mathbf{t}}_k - \mathbf{t}_k &= -\boldsymbol{\epsilon}_{H_f}\mathbf{z}_k + (\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f)\epsilon_{z,k} - \boldsymbol{\epsilon}_{Fx} \\ \boldsymbol{\xi}^q_{k+1} &:= \tilde{\mathbf{U}}_{k+1} - \mathbf{U}_{k+1} = \mathbf{D}^q_k(\tilde{\mathbf{t}}_k - \mathbf{t}_k) \\ \tilde{\mathbf{z}}_{k+1} - \mathbf{z}_{k+1} &= \epsilon_\eta \Delta\mathbf{U}_k + (1+\bar{\eta})\boldsymbol{\xi}^q_{k+1} - \bar{\eta}\boldsymbol{\xi}^q_k, \end{aligned} \tag{E.2.1}$$

where we used the notation: $\Delta\mathbf{U}_k = \mathbf{U}_{k+1} - \mathbf{U}_k$; $\epsilon_\eta = \bar{\eta} - \eta$; $\boldsymbol{\epsilon}_{H_f} = \bar{\mathbf{H}}_f - \mathbf{H}/(cL)$; $\boldsymbol{\epsilon}_{Fx} = \bar{\mathbf{F}}_f^\intercal \bar{\mathbf{x}}(t) - \mathbf{F}^\intercal \mathbf{x}(t)/(cL) = \boldsymbol{\epsilon}_{F_f}^\intercal \mathbf{x}(t) + \bar{\mathbf{F}}_f \boldsymbol{\epsilon}_x$; $\boldsymbol{\epsilon}_x = \bar{\mathbf{x}}(t) - \mathbf{x}(t)$; $\boldsymbol{\epsilon}_{F_f} = \bar{\mathbf{F}}_f - \mathbf{F}/(cL)$.

The error between $\tilde{\mathbf{U}}_{k+1}$ and $\mathbf{U}_{k+1}$ is reduced from $\tilde{\mathbf{t}}_k - \mathbf{t}_k$ due to the projection on the hyperbox. Hence, to represent $\boldsymbol{\xi}^q_{k+1}$ in (E.2.1), we multiply $\tilde{\mathbf{t}}_k - \mathbf{t}_k$ by the diagonal matrix $\mathbf{D}^q_k$ that has positive elements at most one.

We set $\boldsymbol{\xi}^q_{-1} = \boldsymbol{\xi}^q_0$. From (E.2.1), we derive a recursive iteration that characterizes the error of the primal iterate, for $k = 0, \dots, K$, which we can write as a linear system:

$$\begin{aligned}
\tilde{\mathbf{A}}(\mathbf{D}_k^q) &:= \begin{bmatrix} (1+\bar{\eta})\mathbf{D}_k^q(\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f) & -\bar{\eta}\mathbf{D}_k^q(\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f) \\ \mathbf{I}_{Nm} & \mathbf{0}_{Nm} \end{bmatrix} \\
\tilde{\mathbf{B}}(\mathbf{D}_k^q) &:= \begin{bmatrix} -\boldsymbol{\epsilon}_{H_f}\mathbf{D}_k^q & \epsilon_\eta \mathbf{D}_k^q(\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f) \\ \mathbf{0}_{Nm} & \mathbf{0}_{Nm} \end{bmatrix} \\
\begin{bmatrix} \boldsymbol{\xi}_{k+1}^q \\ \boldsymbol{\xi}_k^q \end{bmatrix} &= \tilde{\mathbf{A}}(\mathbf{D}_k^q) \begin{bmatrix} \boldsymbol{\xi}_k^q \\ \boldsymbol{\xi}_{k-1}^q \end{bmatrix} + \tilde{\mathbf{B}}(\mathbf{D}_k^q) \begin{bmatrix} \mathbf{z}_k \\ \Delta\mathbf{U}_{k-1} \end{bmatrix} - \boldsymbol{\epsilon}_{Fx}.
\end{aligned} \tag{E.2.2}$$

We choose this representation in order to have a relevant error bound in Theorem E.2.1 that shrinks to zero as the number of fractional bits grows. In the following, we find an upper bound of the error using $\tilde{\mathbf{A}} := \tilde{\mathbf{A}}(\mathbf{I}_{Nm})$ and $\tilde{\mathbf{B}} := \tilde{\mathbf{B}}(\mathbf{I}_{Nm})$.

Theorem E.2.1. *Under Assumption 6.2.4, the system defined by* (E.2.2) *is bounded. Furthermore, the norm of the error between the primal iterates of the original problem and of the problem with quantized coefficients is bounded by:*

$$\begin{aligned}
\|\boldsymbol{\xi}_k^q\|_2 &\le \left\|\mathbf{E}\tilde{\mathbf{A}}^k\right\|_2 \left\| \begin{bmatrix} \boldsymbol{\xi}_0^q \\ \boldsymbol{\xi}_{-1}^q \end{bmatrix} \right\|_2 + \gamma \sum_{l=0}^{k-1} \left\|\mathbf{E}\tilde{\mathbf{A}}^{k-1-l}\tilde{\mathbf{B}}\right\|_2 + \zeta \sum_{l=0}^{k-1} \left\|\mathbf{E}\tilde{\mathbf{A}}^{k-1-l}\right\|_2 =: \epsilon_1 \\
\gamma &= (3 + 2\bar{\eta})\sqrt{Nm}\mathcal{R}_{\bar{\mathcal{U}}}, \\
\zeta &= \|\boldsymbol{\epsilon}_{F_f}\|_2 \mathcal{R}_{\mathcal{X}_0}^2 + 2^{-l_f}\sqrt{n}\|\bar{\mathbf{F}}_f\|_2,
\end{aligned}$$

where $\mathbf{E} = [\mathbf{I}_{Nm} \;\; \mathbf{0}_{Nm}]$, $\mathcal{R}_{\mathcal{X}_0}^2$ *is the radius of the compact set* $\mathcal{X}_0$ *w.r.t. the 2-norm and* $\mathcal{R}_{\bar{\mathcal{U}}} = \max\{\|\mathbf{l}_u\|_\infty, \|\mathbf{h}_u\|_\infty\}$.

Proof. The stability of the system is given by the fact that $\tilde{\mathbf{A}}$ has spectral radius $\rho(\tilde{\mathbf{A}}) < 1$ which is proven in Lemma 1 in [129]. The same holds for $\tilde{\mathbf{A}}(\mathbf{D}_k^q)$. Since we want to give a bound of the error in terms of computable values, we use the fact that $\|\tilde{\mathbf{A}}(\mathbf{D}_k^q)\|_2 \le \|\tilde{\mathbf{A}}\|_2$ (respectively, $\|\tilde{\mathbf{B}}(\mathbf{D}_k^q)\|_2 \le \|\tilde{\mathbf{B}}\|_2$) and express the bounds in terms of the latter.

From (E.2.2), one can obtain the following expression for the errors at time k and $k-1$, for $k = 0, \ldots, K-1$:

$$\begin{bmatrix} \boldsymbol{\xi}_k^q \\ \boldsymbol{\xi}_{k-1}^q \end{bmatrix} \le \tilde{\mathbf{A}}^k \begin{bmatrix} \boldsymbol{\xi}_0^q \\ \xi_{-1}^r \end{bmatrix} + \sum_{l=0}^{k-1} \tilde{\mathbf{A}}^{k-1-l} \left(\tilde{\mathbf{B}} \begin{bmatrix} \mathbf{z}_l \\ \Delta\mathbf{U}_{l-1} \end{bmatrix} - \boldsymbol{\epsilon}_{Fx} \right),$$

and the first term goes to zero as $k \to \infty$. We multiply this by $E = [\mathbf{I}_{Nm}\ \mathbf{0}_{Nm}]$ to obtain the expression of $\|\boldsymbol{\xi}_k^q\|$.

Subsequently, for any $0 \le k \le K-1$:

$$\begin{aligned} &\left|\left|\left[\mathbf{z}_k^\intercal \;\; \Delta\mathbf{U}_{k-1}^\intercal\right]^\intercal\right|\right|_2 = \|\mathbf{z}_k\|_2 + \|\Delta\mathbf{U}_{k-1}\|_2 \le \|\mathbf{U}_k + \bar{\eta}\Delta\mathbf{U}_{k-1}\|_2 + \|\Delta\mathbf{U}_k\|_2 \\ &\le \mathcal{R}_{\bar{\mathcal{U}}} + 2(1+\bar{\eta})\mathcal{R}_{\bar{\mathcal{U}}} \le (3+2\bar{\eta})\sqrt{Nm(\max_i\{\mathbf{l}_u^i, \mathbf{h}_u^i\})^2} := \gamma \\ &\|\boldsymbol{\epsilon}_{Fx}\|_2 \le \|\boldsymbol{\epsilon}_{F_f}\|_2\|\mathbf{x}(t)\|_2 + \|\bar{\mathbf{F}}_f\|_2\|\boldsymbol{\epsilon}_x\|_2 \le \|\boldsymbol{\epsilon}_{F_f}\|_2\mathcal{R}_{\mathcal{X}_0}^2 + 2^{-l_f}\sqrt{n}\|\bar{\mathbf{F}}_f\|_2 := \zeta. \qquad \square \end{aligned}$$

One can eliminate the initial error $\boldsymbol{\xi}_0^q$ and its effect by choosing in both exact and fixed-point coefficient-FGM algorithms the initial iterate to be represented on l_f fractional bits. Therefore, only the persistent noise counts.

Remark E.2.2. In primal-dual algorithms, the maximum values of the dual variables corresponding to the complicating constraints cannot be bounded a priori, i.e., we cannot give overflow or coefficient quantization error bounds. This justifies our focus on a problem with only simple input constraints. The work in [183] considers the bound on the dual variables as a parameter that can be tuned by the user.

Arithmetic round-off errors

Let us now investigate the error between the solution of the previous problem and the solution of the fixed-point FGM corresponding to Protocols 6.2.1 and 6.2.2. The encrypted values do not necessarily maintain the same number of bits after operations, so we will consider round-off errors where we perform truncations. This happens in line 10 in both protocols. In this case, we obtain similar results to [129], where the quantization errors were not analyzed, i.e., as if the nominal coefficients of the problem were represented with l_f

fractional bits from the problem formulation. Consider iteration k of the projected FGM. The errors due to round-off between the primal iterates of the two solutions will be:

$$\begin{aligned}\bar{\mathbf{t}}_k - \tilde{\mathbf{t}}_k =& (\mathbf{I}_{Nm} - \bar{\mathbf{H}}_f)(\bar{\mathbf{z}}_k - \tilde{\mathbf{z}}_k) + \boldsymbol{\epsilon}'_{t,k} \\ \boldsymbol{\xi}^r_{k+1} :=& \bar{\mathbf{U}}_{k+1} - \tilde{\mathbf{U}}_{k+1} = \mathbf{D}^r_k(\bar{\mathbf{t}}_k - \mathbf{t}_k) \\ \bar{\mathbf{z}}_{k+1} - \tilde{\mathbf{z}}_{k+1} =& (1+\bar{\eta})\boldsymbol{\xi}^r_{k+1} - \bar{\eta}\boldsymbol{\xi}^r_k.\end{aligned} \tag{E.2.3}$$

Again, the projection on the hyperbox reduces the error, so $\mathbf{D}^r_k$ is a diagonal matrix with positive elements less than one. For Protocol 6.2.1, the round-off error due to truncation is $(\epsilon'_{t,k})^i \in [-Nm2^{-l_f},\ 0]$, $i = 1, \dots, Nm$. The encrypted truncation step in Protocol 6.2.2 introduces an extra term due to blinding, making $(\boldsymbol{\epsilon}'_{t,k})^i \in [-(1+Nm)2^{-l_f},\ 2^{-l_f}]$.

We set set $\boldsymbol{\xi}^r_{-1} = \boldsymbol{\xi}^r_0$. From (E.2.3), we can derive a recursive iteration that characterizes the error of the primal iterate, which we can write as a linear system, with $\tilde{\mathbf{A}}(\cdot)$ as before:

$$\begin{bmatrix} \xi^r_{k+1} \\ \xi^r_k \end{bmatrix} = \tilde{\mathbf{A}}(\mathbf{D}^r_k) \begin{bmatrix} \xi^r_k \\ \xi^r_{k-1} \end{bmatrix} + \mathbf{D}^r_k \epsilon'_{t,k}. \tag{E.2.4}$$

Theorem E.2.3. *Under Assumption 6.2.4, the system defined by* (E.2.4) *is bounded. Furthermore, the norm of the error of the primal iterate is bounded by:*

$$\begin{aligned}\|\boldsymbol{\xi}^r_k\|_2 \leq& \left|\left|\mathbf{E}\tilde{\mathbf{A}}^k\right|\right|_2 \left|\left|\begin{bmatrix} \boldsymbol{\xi}^r_0 \\ \boldsymbol{\xi}^r_{-1} \end{bmatrix}\right|\right|_2 + \gamma' \sum_{l=0}^{k-1} \left|\left|\mathbf{E}\tilde{\mathbf{A}}^{k-1-l}\right|\right|_2 =: \epsilon_2, \\ \gamma'_{CS} =& 2^{-l_f}(Nm)^{\frac{3}{2}}; \gamma'_{SS} = 2^{-l_f}\sqrt{Nm}(1+Nm).\end{aligned}$$

The proof is straightforward.

As before, one can eliminate the initial error $\boldsymbol{\xi}^r_0$ and its effect by choosing the same initial iterate represented on l_f fractional bits for both problems.

E.3 Proof of Theorem 6.3.2

The components of the protocol: AHE, secret sharing, pseudorandom generator, LabHE, oblivious transfer and the comparison protocol are individually secure, meaning either their output is computationally indistinguishable from a random output or they already satisfy Definition 2.2.6. We next build the views of the allowed coalitions and their corresponding simulators and use the previous results to prove they are computationally indistinguishable.

The cloud has no inputs and no outputs, hence its view is composed solely from received messages and coins:

$$\begin{aligned} V_C(\emptyset) = \Bigg(&\mathrm{mpk}, \hat{\mathrm{E}}\left(-\frac{1}{L}\mathbf{H}\right), \hat{\mathrm{E}}\left(-\frac{\eta}{L}\mathbf{H}\right), \hat{\mathrm{E}}\left(\frac{1}{L}\mathbf{F}^\intercal\right), \hat{\mathrm{E}}(\eta), [[\mathbf{h}_u]], [[\mathbf{l}_u]], \mathrm{R}_C, \hat{\mathrm{E}}(\mathbf{U}'_0), \\ &\left\{\hat{\mathrm{E}}(\mathbf{U}_k), [[\mathbf{a}_k]], [[\mathbf{b}_k]], [[U_{k+1}]], \mathrm{msg}_{\mathrm{Pr.2.7.2}}, \mathrm{msg}_{\mathrm{OT}}\right\}_{k\in\{0,\dots,K-1\}}\Bigg). \end{aligned} \tag{E.3.1}$$

The actuator's input is the function f_{MPC} and the output is $\mathbf{u}(t)$. Then, its view is:

$$V_A(f_{\mathrm{MPC}}) = \left(f_{\mathrm{MPC}}, \mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \mathrm{R}_A, \left\{\mathbf{U}'_{k+1}, \mathrm{msg}_{\mathrm{Pr.2.7.2}}, \mathrm{msg}_{\mathrm{OT}}\right\}_{k\in\{0,\dots,K-1\}}, \mathbf{u}(t)\right), \tag{E.3.2}$$

which includes the keys mpk, msk because their generation involve randomness.

The setup's inputs are the model and costs of the system and no output after the execution of Protocol 6.3.2, since it is just a helper entity. Its view is:

$$V_{Set}(\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}) = \left(\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathrm{mpk}, \mathrm{usk}_{Set}, \mathrm{R}_{Set}\right). \tag{E.3.3}$$

Finally, for a subsystem i, $i \in [M]$, the inputs are the local control action constraints and the measured states and there is no output obtained through computation after the execution of Protocol 6.3.2. Its view is:

$$V_{S^i}(\mathcal{U}^i, \mathbf{x}^i(t)) = \left(\mathcal{U}^i, \mathbf{x}^i(t), \mathrm{mpk}, \mathrm{usk}_{S^i}, \mathrm{R}_{S^i}\right). \tag{E.3.4}$$

In general, the indistinguishability between the view of the adversary corrupting the real-world parties and the simulator is proved through sequential games in which some real components of the view are replaced by components that could be generated by the simulator, which are indistinguishable from each other. In our case, we can directly make the leap between the real view and the simulator by showing that the cloud only receives encrypted messages, and the actuator receives only messages blinded by one-time pads. In [13], the proof for the privacy of a quadratic optimization problem solved in the same architecture is given with sequential games.

For the cloud, consider a simulator S_C that generates $\widetilde{\text{mpk}}, \widetilde{\text{msk}} \leftarrow \text{Init}(1^\sigma)$, generates $\widetilde{\text{usk}}_j \leftarrow \text{KeyGen}(\widetilde{\text{mpk}})$, for $j \in \{S^i, Set, A\}$, $i \in [M]$ and then $(\widetilde{\boldsymbol{\tau}}, \widetilde{\mathbf{b}}, \widetilde{[[\mathbf{b}]]})_j$. We use $\widetilde{(\cdot)}$ also over the encryptions to show that the keys are different from the ones in the view. Subsequently, the simulator encrypts random values of appropriate sizes to obtain $\widetilde{\hat{\text{E}}\left(-\frac{1}{L}\mathbf{H}\right)}, \widetilde{\hat{\text{E}}\left(-\frac{\eta}{L}\mathbf{H}\right)}, \widetilde{\hat{\text{E}}\left(\frac{1}{L}\mathbf{F}^\intercal\right)}, \widetilde{\hat{\text{E}}(\eta)}, \widetilde{[[\mathbf{h}_u]]}, \widetilde{[[\mathbf{l}_u]]}, \widetilde{\hat{\text{E}}(\mathbf{U}_0')}$. S_C generates the coins $\widetilde{\text{R}}_C$ as in line 4 in Protocol 6.3.1 and obtains $\hat{\text{E}}(\mathbf{U}_0)$ as in line 4 in Protocol 6.3.2. Then, for each $k = \{0, \dots, K-1\}$, it computes $\widetilde{[[\mathbf{t}_k]]}$ as in line 7 in Protocol 6.3.2 and shuffles $\widetilde{[[\mathbf{t}_k]]}$ and $\widetilde{[[\mathbf{h}_u]]}$ into $\widetilde{[[\mathbf{a}_k]]}, \widetilde{[[\mathbf{b}_k]]}$. S_C then performs the same steps as the simulator for party A in Protocol 2.7.2 and gets $\widetilde{\text{msg}}_{\text{Pr.2.7.2}}$. Furthermore, S_C generates an encryption of random bits $\widetilde{\boldsymbol{\delta}}_k$ and of $\widetilde{\hat{\text{E}}(\mathbf{U}_k)}$ and performs the same steps as the simulator for party A as in the proof of Proposition 2.6.1 (or the simulator for the standard OT) and gets $\widetilde{\text{msg}}_{\text{OT}}$. It then outputs:

$$S_C(\emptyset) = \Bigg(\widetilde{\text{mpk}}, \widetilde{\hat{\text{E}}\left(-\frac{1}{L}\mathbf{H}\right)}, \widetilde{\hat{\text{E}}\left(-\frac{\eta}{L}\mathbf{H}\right)}, \widetilde{\hat{\text{E}}\left(\frac{1}{L}\mathbf{F}^\intercal\right)}, \widetilde{\hat{\text{E}}(\eta)}, \widetilde{[[\mathbf{h}_u]]}, \widetilde{[[\mathbf{l}_u]]}, \widetilde{\text{R}}_C, \widetilde{\hat{\text{E}}(\mathbf{U}_0')},$$
$$\left\{\hat{\text{E}}(\widetilde{\mathbf{U}_k}), \left(\widetilde{[[\mathbf{a}_k]]}, \widetilde{[[\mathbf{b}_k]]}\right), \widetilde{[[U_{k+1}]]}, \widetilde{\text{msg}}_{\text{Pr.2.7.2}}, \widetilde{\text{msg}}_{\text{OT}}\right\}_{k\in\{0,\dots,K-1\}}\Bigg). \tag{E.3.5}$$

All the values in the view of the cloud in (E.3.1)–with the exception of the random values R_C and the key mpk, which are statistically indistinguishable from $\widetilde{\text{R}}_C$ and $\widetilde{\text{mpk}}$ because they are drawn from the same distributions–are encrypted with semantically secure encryptions schemes (AHE and LabHE). This means they are computationally indistinguishable from the encryptions of random values in (E.3.5). This happens even when the values from different iterations are encryptions of correlated quantities. Thus, $V_C(\emptyset) \overset{c}{\equiv} S_C(\emptyset)$.

We now build a simulator S_A for the actuator that takes as input $f_{\text{MPC}}, \mathbf{u}(t)$. S_A will take the same steps as in lines 1, 3–7 in Protocol 6.3.1, obtaining $\widetilde{\text{mpk}}, \widetilde{\text{msk}}, \widetilde{\text{upk}}, \widetilde{\text{usk}}, \widetilde{\boldsymbol{\tau}}_A$, $\widetilde{\mathbf{b}}_A, \widetilde{[[\mathbf{b}_A]]}, \widetilde{\text{R}}_A$, $\widetilde{\mathbf{U}'_0}$ and instead of line 2, it generates $\widetilde{\text{upk}}_j, \widetilde{\text{usk}}_j \leftarrow \text{KeyGen}(\widetilde{\text{mpk}})$ itself, for $j \in \{S^i, Set\}$, $i \in [M]$. For $k = 0, \ldots, K-1$, S_A performs the same steps as the simulator for party B in Protocol 2.7.2 and gets $\widetilde{\text{msg}}_{\text{Pr.2.7.2}}$. Furthermore, S_A performs the same steps as the simulator for party B as in the proof of Proposition 2.6.1 (or the simulator for the standard OT) and gets $\widetilde{\text{msg}}_{\text{OT}}$. It then outputs:

$$\begin{aligned} S_A(f_{\text{MPC}}, \mathbf{u}(t)) = \Bigg(& f_{\text{MPC}}, \widetilde{\text{mpk}}, \widetilde{\text{msk}}, \widetilde{\text{upk}}, \widetilde{\text{R}}_A, \\ & \left\{ \widetilde{\mathbf{U}'_{k+1}}, \widetilde{\text{msg}}_{\text{Pr.2.7.2}}, \widetilde{\text{msg}}_{\text{OT}} \right\}_{k \in \{0,\ldots,K-1\}}, \mathbf{u}(t) \Bigg). \end{aligned} \tag{E.3.6}$$

All the values in the view of the actuator in (E.3.2)–with the exception of the random values R_A and the keys upk, which are statistically indistinguishable from $\widetilde{\text{R}}_A$ and $\widetilde{\text{upk}}$ because they are drawn from the same distributions and $\mathbf{u}(t)$–are blinded by random numbers, different at every iteration, which means that they are statistically indistinguishable from the random values in (E.3.6). This again holds even when the values that are blinded at different iterations are correlated and the actuator knows the solution $\mathbf{u}(t)$, because the values of interest are drowned in large noise. Thus, $V_A(f_{\text{MPC}}) \overset{c}{\equiv} S_A(f_{\text{MPC}}, \mathbf{u}(t))$.

The setup and subsystems do not receive any other messages apart from the master public key (E.3.3), (E.3.4). Hence, a simulator S_{Set} for the setup and a simulator S_{S^i} for a subsystem i can simply generate $\widetilde{\text{mpk}} \leftarrow \text{Init}(1^\sigma)$ and then proceed with the execution of lines 2–5 in Protocol 6.3.1 and output their inputs, messages and coins. The outputs of the simulators are trivially indistinguishable from the views.

When an adversary corrupts a coalition, the view of the coalition contains the inputs of all parties, and a simulator takes the coalition's inputs and outputs. The view of the coalition between the cloud, the setup, and a number l of subsystems is:

$$V_{CSl}\big(\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}, \{\mathcal{U}^i, \mathbf{x}^i(t)\}_{i \in i_1,\dots,i_l}\big) = V_C(\emptyset) \cup V_{Set}(\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}) \cup \\ \cup V_{S^{i_1}}(\mathcal{U}^{i_1}, \mathbf{x}^{i_1}(t)) \cup \ldots \cup V_{S^{i_l}}(\mathcal{U}^{i_l}, \mathbf{x}^{i_l}(t)).$$

A simulator S_{CSl} for this coalition takes in the inputs of the coalition and no output and performs almost the same steps as S_C, S_{Set}, S_{S^i}, without randomly generating the quantities that are known by the coalition. The same argument of having the messages drawn from the same distributions and encrypted with semantically secure encryption schemes proves the indistinguishability between $V_{CSl}(\cdot)$ and $S_{CSl}(\cdot)$.

The view of the coalition between the actuator, the setup, and l subsystems is:

$$V_{ASl}\big(f_{\text{MPC}}, \mathbf{u}(t),\ \mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}, \{\mathcal{U}^i, \mathbf{x}^i(t)\}_{i \in i_1,\dots,i_l}\big) = V_A(f_{\text{MPC}}, \mathbf{u}(t)) \cup \\ \cup V_{Set}(\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}) \cup V_{S^{i_1}}(\mathcal{U}^{i_1}, \mathbf{x}^{i_1}(t)) \cup \ldots \cup V_{S^{i_l}}(\mathcal{U}^{i_l}, \mathbf{x}^{i_l}(t)).$$

A simulator S_{ASl} for this coalition takes in the inputs of the coalition and $\mathbf{u}(t)$ and performs almost the same steps as S_A, S_{Set}, S_{S^i}, without randomly generating the quantities that are now known. The same argument of having the messages drawn from the same distributions and blinded with one-time pads proves the indistinguishability between $V_{ASl}(\cdot)$ and $S_{ASl}(\cdot)$.

The proof is now complete. $\square$

Appendix F

Technical details for Chapter 7

F.1 Proof of closeness

In this section, we present the proof of Theorem 7.2.2. First, we characterize the solution set of the original behavioral problem (7.2.5). Then, we characterize the solution of the approximate problem (7.2.6). To prove closeness, we make suitable use of the inversion lemma, the pseudoinverse (denoted by †) limit definition, and Singular Value Decomposition (SVD).

Before we proceed with the proof, we consider the following definitions:

$$\hat{\mathbf{M}} := \mathbf{Y}^{f\top}\mathbf{Q}\mathbf{Y}^f + \mathbf{U}^{f\top}\mathbf{R}\mathbf{U}^f + \lambda_g \mathbf{I} \tag{F.1.1}$$

$$\hat{\mathbf{M}}_0 := \mathbf{Y}^{f\top}\mathbf{Q}\mathbf{Y}^f + \mathbf{U}^{f\top}\mathbf{R}\mathbf{U}^f, \tag{F.1.2}$$

where $\hat{\mathbf{M}}$ is matrix $\mathbf{M}$ defined in (7.2.8), without the penalty terms. Similarly, $\hat{\mathbf{M}}_0$ is matrix $\mathbf{M}$ without any regularization or penalty terms.

For compactness, we denote the matrix of past precollected data as $\mathbf{D}_p$ and the vector of the last M outputs and inputs as $\mathbf{d}_t$:

$$\mathbf{D}_p := \begin{bmatrix} \mathbf{U}^p \\ \mathbf{Y}^p \end{bmatrix}, \qquad \mathbf{d}_t := \begin{bmatrix} \bar{\mathbf{u}}_t \\ \bar{\mathbf{y}}_t \end{bmatrix}. \tag{F.1.3}$$

Both matrices $\hat{\mathbf{M}}_0$, $\mathbf{D}_p$ are singular. Let the SVD of the past precollected data $\mathbf{D}_p$ be:

$$\mathbf{D}_p = \begin{bmatrix} \mathbf{E} & \mathbf{E}_\perp \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{F}^\intercal \\ \mathbf{F}_\perp{}^\intercal \end{bmatrix}, \tag{F.1.4}$$

where $\mathbf{\Sigma} \in \mathbb{R}^{q\times q}$ is an invertible diagonal matrix, containing the nonzero singular values of $\mathbf{D}_p$ with $q = \operatorname{rank}(\mathbf{D}_p)$, while matrices $\begin{bmatrix} \mathbf{E} & \mathbf{E}_\perp \end{bmatrix}$, $\begin{bmatrix} \mathbf{F} & \mathbf{F}_\perp \end{bmatrix}$ are orthonormal. Recall that the set of optimal solutions of the original behavioral problem (7.2.5) is denoted by $\mathcal{G}_{\text{opt}}$–see (7.2.11). The minimum norm element of $\mathcal{G}_{\text{opt}}$ is denoted by $\mathbf{g}_{\text{min}}$–see (7.2.12).

For completeness we present some inversion formulae.

Lemma F.1.1. *Let* $\mathbf{K}, \mathbf{L}, \mathbf{V}$ *be matrices of appropriate dimensions. Then:*

a) Provided the respective inverses exist:

$$(\mathbf{K} + \mathbf{V}^\intercal\mathbf{L}\mathbf{V})^{-1} = \mathbf{K}^{-1} - \mathbf{K}^{-1}\mathbf{V}^\intercal(\mathbf{L}^{-1} + \mathbf{V}\mathbf{K}^{-1}\mathbf{V}^\intercal)^{-1}\mathbf{V}\mathbf{K}^{-1}$$
$$(\mathbf{K} + \mathbf{V}^\intercal\mathbf{L}\mathbf{V})^{-1}\mathbf{V}^\intercal\mathbf{L} = \mathbf{K}^{-1}\mathbf{V}^\intercal(\mathbf{L}^{-1} + \mathbf{V}\mathbf{K}^{-1}\mathbf{V}^\intercal)^{-1}.$$

b) Let $\begin{bmatrix} \mathbf{V} & \mathbf{V}_\perp \end{bmatrix}$ *be an orthonormal matrix. Then, if* $\mathbf{L}$ *is invertible:*

$$(\mathbf{V}^\intercal\mathbf{L}^{-1}\mathbf{V})^{-1} = \mathbf{V}^\intercal\mathbf{L}\mathbf{V} - \mathbf{V}^\intercal\mathbf{L}\mathbf{V}_\perp(\mathbf{V}_\perp^\intercal\mathbf{L}\mathbf{V}_\perp)^{-1}\mathbf{V}_\perp^\intercal\mathbf{L}\mathbf{V} \tag{F.1.5}$$
$$\mathbf{I} - \mathbf{V}_\perp(\mathbf{V}_\perp^\intercal\mathbf{L}\mathbf{V}_\perp)^{-1}\mathbf{V}_\perp^\intercal\mathbf{L} = \mathbf{L}^{-1}\mathbf{V}(\mathbf{V}^\intercal\mathbf{L}^{-1}\mathbf{V})^{-1}\mathbf{V}^\intercal. \tag{F.1.6}$$

Proof. Part a) is standard, also known as the Woodbury matrix identity. To prove (F.1.5), we use the identity $\mathbf{V}_\perp\mathbf{V}_\perp^\intercal + \mathbf{V}\mathbf{V}^\intercal = \mathbf{I}$ and verify that the properties of the inverse hold. To prove (F.1.6), we use (F.1.5). □

Original behavioral problem

In the following lemma, we characterize the solution set $\mathcal{G}_{\text{opt}}$ of the original constrained behavioral problem (7.2.5). In general, there are infinite optimal solutions since $\hat{\mathbf{M}}_0$ and $\mathbf{D}_p$ have low rank.

Lemma F.1.2. *Consider optimization problem* (7.2.5)*. Recall the SVD decomposition of* $\mathbf{D}_p$ *in* (F.1.4)*. Then:*

a) $\mathbf{g} \in \mathcal{G}_{\text{opt}}$ *is an optimal solution of* (7.2.5) *if and only if* $\mathbf{g}$ *lies in the following affine subspace for some* $\boldsymbol{\mu} \in \mathbb{R}^{q\times 1}$*, where* q *is the rank of the past precollected data* $\mathbf{D}_p$*:*

$$\begin{bmatrix} \hat{\mathbf{M}}_0 & \mathbf{F} \\ \mathbf{F}^\intercal & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{g} \\ \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t \\ \boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal\mathbf{d}_t \end{bmatrix}. \tag{F.1.7}$$

b) Vector $\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$ *lies in the column space of* $\hat{\mathbf{M}}_0$*.*

c) The above affine subspace can be equivalently described by the closed form expression:

$$\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3, \tag{F.1.8}$$

where $\mathbf{g}_i$*,* $i = 1, 2, 3$ *are orthogonal with each other and, for arbitrary* $\mathbf{s}$*:*

$$\mathbf{g}_1 = \mathbf{F}\boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal\mathbf{d}_t \tag{F.1.9}$$

$$\mathbf{g}_2 = \mathbf{F}_\perp(\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp)^\dagger\mathbf{F}_\perp{}^\intercal\left(\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t - \hat{\mathbf{M}}_0\mathbf{g}_1\right) \tag{F.1.10}$$

$$\mathbf{g}_3 = \mathbf{F}_\perp(\mathbf{I} - (\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp)^\dagger\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp)\mathbf{s}. \tag{F.1.11}$$

d) As a result, the minimum norm element of $\mathcal{G}_{\text{opt}}$ *is*

$$\mathbf{g}_{\min} = \mathbf{g}_1 + \mathbf{g}_2. \tag{F.1.12}$$

Proof. **Proof of a)** Since $\mathbf{d}_t$ lies in the range space of $\mathbf{D}_p$, we can replace the singular equality constraint $\mathbf{D}_p\mathbf{g} = \mathbf{d}_t$ by $\mathbf{F}^\intercal\mathbf{g} = \boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal\mathbf{d}_t$, where now $\mathbf{F}$ has full rank. If we also replace $\mathbf{u}$, $\mathbf{y}$ with $\mathbf{U}^f\mathbf{g}$ and $\mathbf{Y}^f$ respectively, and we ignore the constant terms, then we obtain the optimization problem:

$$\min_{\mathbf{g}} \ \tfrac{1}{2}\mathbf{g}^\intercal\hat{\mathbf{M}}_0\mathbf{g} - \mathbf{g}^\intercal\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t \qquad \text{s.t. } \mathbf{F}^\intercal\mathbf{g} = \boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal\mathbf{d}_t, \tag{F.1.13}$$

which is equivalent to (7.2.5). The Lagrangian of (F.1.13) is $\mathcal{L} = \frac{1}{2}\mathbf{g}^\intercal\hat{\mathbf{M}}_0\mathbf{g} - \mathbf{g}^\intercal\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t + \boldsymbol{\mu}^\intercal(\mathbf{F}^\intercal\mathbf{g} - \boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal\mathbf{d}_t)$, where $\boldsymbol{\mu} \in \mathbb{R}^q$ are the Lagrange multipliers. The result follows by applying standard first order optimality conditions.

Proof of b) Observe that $\hat{\mathbf{M}}_0 = \begin{bmatrix}\mathbf{Y}^{f\intercal}\mathbf{Q}^{1/2} & \mathbf{U}^{f\intercal}\mathbf{R}^{1/2}\end{bmatrix}\begin{bmatrix}\mathbf{Y}^{f\intercal}\mathbf{Q}^{1/2} & \mathbf{U}^{f\intercal}\mathbf{R}^{1/2}\end{bmatrix}^\intercal$. It is clear that $\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$ lies in the column space of $\begin{bmatrix}\mathbf{Y}^{f\intercal}\mathbf{Q}^{1/2} & \mathbf{U}^{f\intercal}\mathbf{R}^{1/2}\end{bmatrix}$. Then, using the fact that for a matrix $\mathbf{A}$, Range($\mathbf{A}$) = Range($\mathbf{A}\mathbf{A}^\intercal$) (the proof easily follows from using SVD on $\mathbf{A}$), we also show that $\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$ lies in the column space of $\hat{\mathbf{M}}_0$.

Proof of c) First, we solve subequation $\mathbf{F}^\intercal\mathbf{g} = \boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal\mathbf{d}_t$. Since matrix $\begin{bmatrix}\mathbf{F} & \mathbf{F}_\perp\end{bmatrix}$ is an orthonormal basis, we can express all solutions $\mathbf{g}$ as:

$$\mathbf{g} = \mathbf{F}\boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal\mathbf{d}_t + \mathbf{F}_\perp\boldsymbol{\xi} = \mathbf{g}_1 + \mathbf{F}_\perp\boldsymbol{\xi}, \tag{F.1.14}$$

where $\mathbf{F}_\perp\boldsymbol{\xi}$ captures the components of $\mathbf{g}$ in the kernel of $\mathbf{F}^\intercal$, and $\boldsymbol{\xi}$ is to be determined.

Second, we solve subequation $\hat{\mathbf{M}}_0\mathbf{g} + \mathbf{F}\boldsymbol{\mu} = \mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$. Multiplying from the left by the orthonormal matrix $\begin{bmatrix}\mathbf{F} & \mathbf{F}_\perp\end{bmatrix}^\intercal$ results in the equivalent system of subequations:

$$\mathbf{F}^\intercal\hat{\mathbf{M}}_0\mathbf{g} + \boldsymbol{\mu} = \mathbf{F}^\intercal\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t \tag{F.1.15}$$

$$\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{g} = \mathbf{F}_\perp{}^\intercal\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t. \tag{F.1.16}$$

Equation (F.1.15) determines the Lagrange multiplier $\boldsymbol{\mu}$. Replacing (F.1.14) into (F.1.16) gives: $\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp\boldsymbol{\xi} = \mathbf{F}_\perp{}^\intercal\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t - \mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{g}_1$. By the standard property of the pseudoinverse [32, Ch. 2] and for an arbitrary $\mathbf{s}$, we can write all possible solutions $\boldsymbol{\xi}$ as:

$$\boldsymbol{\xi} = \underbrace{(\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp)^\dagger\mathbf{F}_\perp{}^\intercal\left(\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t - \hat{\mathbf{M}}_0\mathbf{g}_1\right)}_{\boldsymbol{\xi}_1} + \underbrace{\left(\mathbf{I} - (\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp)^\dagger\mathbf{F}_\perp{}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp\right)\mathbf{s}}_{\boldsymbol{\xi}_2}, \tag{F.1.17}$$

The two components $\boldsymbol{\xi}_1$, $\boldsymbol{\xi}_2$ are orthogonal. Combining (F.1.14) and (F.1.17) gives: $\mathbf{g} = \mathbf{g}_1 + \mathbf{F}_\perp\boldsymbol{\xi}_1 + \mathbf{F}_\perp\boldsymbol{\xi}_2$, where all components are orthogonal to each other.

Proof of d) Follows from b), orthogonality of the three summands and setting the only free parameter $\mathbf{s}$ to zero. □

Approximate problem

Due to the regularization term, the solution of the approximate problem (7.2.6) is unique– see (7.2.9). We can show that if we chose large enough λ, then we can show that the solution of (7.2.6) converges to an intermediate one. Let $\hat{\mathbf{F}} := \mathbf{F}^\intercal \hat{\mathbf{M}}^{-1}\mathbf{F}$.

Lemma F.1.3. *Consider optimization problem* (7.2.6) *with* $\lambda_u = \lambda_y = \lambda > 0$ *and* $\lambda_g > 0$ *and let* $\mathbf{g}^*$ *be the optimal solution. Define the intermediate solution:*

$$\bar{\mathbf{g}} := \hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}\boldsymbol{\Sigma}^{-1}\mathbf{E}^\intercal \mathbf{d}_t + \left[\hat{\mathbf{M}}^{-1} - \hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}\mathbf{F}^\intercal\hat{\mathbf{M}}^{-1}\right]\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t. \tag{F.1.18}$$

Then, as $\lambda \to \infty$ *and* $\lambda_g \to 0^+$*:*

$$\|\mathbf{g}^* - \bar{\mathbf{g}}\|_2 \leq O(\lambda^{-1}), \tag{F.1.19}$$

where we used big-O notation to hide quantities which do not depend on λ, λ_g.

Proof. **Step a).** Recall that $\mathbf{g}^*$ is given by (7.2.9). We will use the inversion lemma to bring $\mathbf{g}^*$ in a form closer to $\bar{\mathbf{g}}$. Using the notation of this section, (7.2.9) can be written as:

$$\begin{aligned}\mathbf{g}^* &= \mathbf{M}^{-1}\left(\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t + \lambda\mathbf{D}_p\right) = (\hat{\mathbf{M}} + \lambda\mathbf{F}\boldsymbol{\Sigma}^2\mathbf{F}^\intercal)^{-1}\left(\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t + \lambda\mathbf{F}\boldsymbol{\Sigma}\mathbf{E}^\intercal\mathbf{d}_t\right)\\ &= (\hat{\mathbf{M}} + \lambda\mathbf{F}\boldsymbol{\Sigma}^2\mathbf{F}^\intercal)^{-1}\left(\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t + \lambda\mathbf{F}\boldsymbol{\Sigma}^2\mathbf{F}^\intercal\mathbf{g}_1\right),\end{aligned}$$

where the second equality follows from $\mathbf{D}_p^\intercal\mathbf{D}_p = \mathbf{F}\boldsymbol{\Sigma}^2\mathbf{F}^\intercal$ and the last equality follows from $\mathbf{F}^\intercal\mathbf{F} = \mathbf{I}$. By Lemma F.1.1 a):

$$\begin{aligned}(\hat{\mathbf{M}} + \lambda\mathbf{F}\boldsymbol{\Sigma}^2\mathbf{F}^\intercal)^{-1} &= \hat{\mathbf{M}}^{-1} - \hat{\mathbf{M}}^{-1}\mathbf{F}\left(\lambda^{-1}\boldsymbol{\Sigma}^{-2} + \hat{\mathbf{F}}\right)^{-1}\mathbf{F}^\intercal\hat{\mathbf{M}}^{-1},\\ (\hat{\mathbf{M}} + \lambda\mathbf{F}\boldsymbol{\Sigma}^2\mathbf{F}^\intercal)^{-1}\mathbf{F}\lambda\boldsymbol{\Sigma}^2 &= \hat{\mathbf{M}}^{-1}\mathbf{F}\left(\lambda^{-1}\boldsymbol{\Sigma}^{-2} + \hat{\mathbf{F}}\right)^{-1}.\end{aligned}$$

Step b). Next, define the difference:

$$\boldsymbol{\Delta} := \left(\lambda^{-1}\boldsymbol{\Sigma}^2 + \hat{\mathbf{F}}\right)^{-1} - \hat{\mathbf{F}}^{-1} = \lambda^{-1}\hat{\mathbf{F}}^{-1}\boldsymbol{\Sigma}^2\left(\lambda^{-1}\boldsymbol{\Sigma}^2 + \hat{\mathbf{F}}\right)^{-1},$$

where the equality follows from $\mathbf{A}^{-1}+\mathbf{B}^{-1}=\mathbf{B}^{-1}(\mathbf{A}+\mathbf{B})\mathbf{A}^{-1}$. The error $\mathbf{g}^*-\bar{\mathbf{g}}$ can be written as:

$$\mathbf{g}^*-\bar{\mathbf{g}}=-\hat{\mathbf{M}}^{-1}\mathbf{F}\boldsymbol{\Delta}\mathbf{F}^{\intercal}\hat{\mathbf{M}}^{-1}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t+\hat{\mathbf{M}}^{-1}\mathbf{F}\boldsymbol{\Delta}\boldsymbol{\Sigma}^{-1}\mathbf{E}^{\intercal}\mathbf{d}_t.$$

To complete the proof, it is sufficient to show that the matrices $\hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}$, $\left(\lambda^{-1}\boldsymbol{\Sigma}^2+\hat{\mathbf{F}}\right)^{-1}$, and $\hat{\mathbf{M}}^{-1}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$ are bounded as $\lambda_g\to 0^+$, $\lambda\to\infty$.

Step c) Boundedness of $\hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}$ follows by Lemma F.1.4.

Step d) The following matrix is always bounded: $\left\|\left(\lambda^{-1}\boldsymbol{\Sigma}^2+\hat{\mathbf{F}}\right)^{-1}\right\|_2\le O(1)$. It follows from $\hat{\mathbf{M}}\preceq(\lambda_g+\|\hat{\mathbf{M}}_0\|_2)\mathbf{I}$, which in turn implies: $\lambda^{-1}\boldsymbol{\Sigma}^2+\hat{\mathbf{F}}\succeq\hat{\mathbf{F}}\succeq(\lambda_g+\|\hat{\mathbf{M}}_0\|_2)^{-1}\mathbf{I}$.

Step e). The norm of $\hat{\mathbf{M}}^{-1}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$ is $O(1)$. From Lemma F.1.2 b), $\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$ lies in the range space of $\hat{\mathbf{M}}_0$, which means $\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t=\hat{\mathbf{M}}_0\hat{\mathbf{M}}_0^{\dagger}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$. Hence, $\|\hat{\mathbf{M}}^{-1}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t\|_2\le O(\|\hat{\mathbf{M}}^{-1}\hat{\mathbf{M}}_0\|_2)=O(1)$, since $\hat{\mathbf{M}}=\hat{\mathbf{M}}_0+\lambda_g\mathbf{I}\succeq\hat{\mathbf{M}}_0$, and $\|\hat{\mathbf{M}}^{-1}\hat{\mathbf{M}}_0\|_2\le 1$. □

Proof of Theorem 7.2.2

Consider the intermediate solution $\bar{\mathbf{g}}$ defined in (F.1.18). By the triangle inequality we have:

$$\|\mathbf{g}_{\min}-\mathbf{g}^*\|_2\le\|\mathbf{g}_{\min}-\bar{\mathbf{g}}\|_2+\|\bar{\mathbf{g}}-\mathbf{g}^*\|_2.$$

In Lemma F.1.3, we show that the second error term decays to zero as fast as λ^{-1}. To complete the proof, we invoke the following Lemma, which states that the first error term decays to zero as well. □

Lemma F.1.4. *Consider the minimum norm solution* $\mathbf{g}_{\min}$ *defined in* (7.2.12) *and the intermediate solution* $\bar{\mathbf{g}}$ *defined in* (F.1.18). *Let* $\lambda_g>0$. *Then:*
a) The following limit holds

$$\lim_{\lambda_g\to 0^+}(\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}\mathbf{F}_{\perp})^{-1}\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}_0=(\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}_0\mathbf{F}_{\perp})^{\dagger}\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}_0. \tag{F.1.20}$$

b) The following matrix is bounded

$$\left\|\hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}\right\|_2 = O(1). \tag{F.1.21}$$

c) The intermediate solution converges to the minimum norm solution as λ_g goes to zero

$$\lim_{\lambda_g \to 0^+} \|\mathbf{g}_{\min} - \bar{\mathbf{g}}\|_2 = 0. \tag{F.1.22}$$

Proof. **Proof of a).** Let $\mathbf{T} := \hat{\mathbf{M}}_0^{1/2}\mathbf{F}_\perp$. Since ${\mathbf{F}_\perp}^\intercal\mathbf{F}_\perp = \mathbf{I}$:

$$({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}_0 = (\lambda_g\mathbf{I} + \mathbf{T}^\intercal\mathbf{T})^{-1}\mathbf{T}^\intercal\hat{\mathbf{M}}_0^{1/2}.$$

The result follows from the pseudoinverse limit definition and $\mathbf{T}^\dagger = (\mathbf{T}^\intercal\mathbf{T})^\dagger\mathbf{T}^\intercal$ [32, Ch 1].

Proof of b). Note that the norm remains unchanged if we multiply from the right by $\mathbf{F}^\intercal$: $\left\|\hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}\right\|_2 = \left\|\hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}\mathbf{F}^\intercal\right\|_2$. By (F.1.6) and the triangle inequality:

$$\begin{aligned}
&\left\|\hat{\mathbf{M}}^{-1}\mathbf{F}\hat{\mathbf{F}}^{-1}\right\|_2 = \left\|\mathbf{I} - \mathbf{F}_\perp(\mathbf{F}_\perp^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}\mathbf{F}_\perp^\intercal\hat{\mathbf{M}}\right\|_2 \\
&\le 1 + \left\|\mathbf{F}_\perp({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\right\|_2 = 1 + \left\|\mathbf{F}_\perp({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\begin{bmatrix}\mathbf{F} & \mathbf{F}_\perp\end{bmatrix}\right\|_2,
\end{aligned}$$

where the last equality follows from the fact that multiplying from the right by $\begin{bmatrix}\mathbf{F} & \mathbf{F}_\perp\end{bmatrix}$ leaves the norm unchanged. Submatrix $\mathbf{F}_\perp({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp = \mathbf{F}_\perp$ has bounded norm. For the other submatrix, since by orthogonality ${\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F} = {\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}_0\mathbf{F}$, we get:

$$\mathbf{F}_\perp({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F} = \mathbf{F}_\perp({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}_0\mathbf{F}.$$

By (F.1.20), we immediately obtain boundedness since:

$$\lim_{\lambda_g \to 0^+}({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F}_\perp)^{-1}{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}\mathbf{F} = ({\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}_0\mathbf{F}_\perp)^\dagger{\mathbf{F}_\perp}^\intercal\hat{\mathbf{M}}_0\mathbf{F}. \tag{F.1.23}$$

Hence, both submatrices have bounded norm, implying boundedness for the original matrix.

Proof of c). Using (F.1.6), we can rewrite $\bar{\mathbf{g}}$ in a form that resembles $\mathbf{g}_{\min}$:

$$\bar{\mathbf{g}} := \mathbf{F}\boldsymbol{\Sigma}^{-1}\mathbf{E}^{\intercal}\mathbf{d}_t - \mathbf{F}_{\perp}{}^{\intercal}(\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}\mathbf{F}_{\perp})^{-1}\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}\mathbf{F}\boldsymbol{\Sigma}^{-1}\mathbf{E}^{\intercal}\mathbf{d}_t + \mathbf{F}_{\perp}{}^{\intercal}(\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}\mathbf{F}_{\perp})^{-1}\mathbf{F}_{\perp}{}^{\intercal}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t. \tag{F.1.24}$$

By (F.1.23), the second term converges to $-\mathbf{F}_{\perp}{}^{\intercal}(\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}_0\mathbf{F}_{\perp})^{\dagger}\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}_0\mathbf{F}\boldsymbol{\Sigma}^{-1}\mathbf{E}^{\intercal}\mathbf{d}_t$. For the third term, recall from Lemma F.1.2 b), that $\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$ lies in the column space of $\hat{\mathbf{M}}_0$, which implies that $\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t = \hat{\mathbf{M}}_0\hat{\mathbf{M}}_0^{\dagger}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$. Using this identity and (F.1.20), we also get that the third term converges to $\mathbf{F}_{\perp}{}^{\intercal}(\mathbf{F}_{\perp}{}^{\intercal}\hat{\mathbf{M}}_0\mathbf{F}_{\perp})^{\dagger}\mathbf{F}_{\perp}{}^{\intercal}\mathbf{Y}^{f\intercal}\mathbf{Q}\mathbf{r}_t$. The result follows by (F.1.12), (F.1.24) and the convergence of the aforementioned terms. ☐

F.2 Implementation details for online solution

Ciphertext packing

Assume without loss of generality that we shift the time axis to the left by $M + N$, such that the trajectory concatenation mentioned in Section 7.2.2.3 is performed before $t = 0$. This will simplify the circuit depth expression.

In the following, we will address some of the encoding methods we investigated, with the goals to minimize the depth of the resulting circuit and to reduce the memory consumption and time complexity. When encoding vectors in plaintexts, we use similar notations as the vector encryptions: $\mathrm{e}_{\mathrm{v}0}(\mathbf{x})$, $\mathrm{e}_{\mathrm{v}*}(\mathbf{x})$, $\mathrm{e}_{\mathrm{vv}}(\mathbf{x})$, $\mathrm{e}_{\mathrm{vr}0}(\mathbf{x})$ and $\mathrm{e}_{\mathrm{vr}*}(\mathbf{x})$.

Scalar encoding: each ciphertext only encodes and encrypts a scalar, i.e., element of a vector or matrix. This version achieves the smallest multiplicative depth compared to the other packing methods, no rotation key storage, but the largest ciphertext storage and the largest number of operations. It returns m ciphertexts instead of one to the client, unless we add more processing to mask and rotate the elements of $\mathbf{u}_t$ to obtain a single ciphertext.

"Hybrid" encoding: some ciphertexts encode and encrypt individual elements while other ciphertexts encode and encrypt vectors. Specifically, we pack the input and output vectors and columns of input and output matrices (without repetitions or other redundancy), but hold the entries of the matrix $\mathbf{M}_t^{-1}$ and intermediate vectors $\mathbf{m}_t$ as individual ciphertexts. This version returns only one vector for the client to decrypt.

The “hybrid” encoding was implemented in our prior work [15]. For the best compromise of depth versus client computation and communication, we asked the client to send back an encryption of $1/s_t$, along with an encryption of the input $\mathbf{u}_t$ and output $\mathbf{y}_t$. This gives the depth expression of the denominator and numerator of $\mathbf{u}_t$, for $t > 1$:

$$d(D\mathbf{u}_t) = d(s_t) = 2t + 1, \quad d(N\mathbf{u}_t) = 2t + 4. \tag{F.2.1}$$

However, this “hybrid” solution still incurred a large storage: namely $O((S+t)^2)$ ciphertexts and $O((S+t)^2)$ rotation keys (only $O(S+t)$ rotation keys when there is no refreshing and packing of the matrix $\mathbf{M}_t^{-1}$ required), and $O((S+t)^2)$ computations at the cloud, with large hidden constants. This suggests the need for a more tractable solution, involving a more efficient encoding.

Vector encoding: each ciphertext encodes and encrypts multiple values, i.e. a vector or a column matrix. Via this type of encoding, we aim to keep the minimum possible depth as in the previous two versions, but minimize the memory requirements for ciphertexts, from $O((S+t)^2)$ to $O(S+t)$. At the same time, we want to decrease the rotation key storage from $O((S+t)^2)$ to $O(S+t)$, and keep the number of computations at $O((S+t)^2)$.

With the previous “hybrid” encoding on the inputs we found it preferable to encode the entries of $\mathbf{M}_t^{-1}$ in individual ciphertexts in order to avoid masking when computing the desired result, which would have increased the depth. However, we found that using redundancy in the stored variables, i.e., repeating the elements in vector encoding of the input columns, along with completely rewriting the way the operations are performed, can alleviate this issue and keep the same depth as before.

We will use the encryption of the repeated elements encoding, $\mathcal{E}_{\mathrm{vr0}}(\cdot)$, for the columns of the Hankel matrices $H\mathbf{U}_t$ and $H\mathbf{Y}_t$, the vectors $\bar{\mathbf{u}}_t, \bar{\mathbf{y}}_t$, the inputs and outputs $\mathbf{u}_t, \mathbf{y}_t$, the reference $\mathbf{r}_t$, and to encode the costs. In the encoding, each element is repeated for $S + \bar{T}$ times, where $\bar{T}$ is the maximum number of online collected samples.

For $t = -M-N+1:0$, follow the steps in the offline feedback solution, cf. Section 7.2.3.

Let $S' := S + t - 1$, for $t \geq 1$. The following plaintexts and ciphertexts are stored at the cloud: λ_y and $\mathbf{Q}$ encoded as $\mathrm{e}_{\mathrm{vr0}}(\lambda\mathbf{Q}) := \mathrm{e}_{\mathrm{vr0}}\left(\begin{bmatrix}\lambda_y & \dots & \lambda_y & q_{[0]} & \dots & q_{[pN-1]}\end{bmatrix}\right)$, λ_u and $\mathbf{R}$ encoded as $\mathrm{e}_{\mathrm{vr0}}(\lambda\mathbf{R}) := \mathrm{e}_{\mathrm{vr0}}\left(\begin{bmatrix}\lambda_u & \dots & \lambda_u & r_{[0]} & \dots & r_{[mN-1]}\end{bmatrix}\right)$, λ_g encoded as $\mathrm{e}_{\mathrm{vr0}}(\lambda_g)$, $\mathcal{E}_{\mathrm{vr0}}(\mathbf{y}_t)$, $\mathcal{E}_{\mathrm{vr0}}(\mathbf{u}_t)$, $\mathcal{E}_{\mathrm{vr0}}(\mathbf{r}_t)$, $\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{y}}_{t-1})$, $\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{u}}_{t-1})$ and for $i \in \{0, \dots, S'+1\}$: $\mathcal{E}_{\mathrm{v0}}(\mathrm{col}_i(\mathbf{M}_{t-1}^{-1}))$, $\mathcal{E}_{\mathrm{vr0}}(\mathrm{col}_i(H\mathbf{Y}_t))$, $\mathcal{E}_{\mathrm{vr0}}(\mathrm{col}_i(H\mathbf{U}_t))$. To avoid some masking operations, the cloud also stores the rows of $\mathbf{U}_t^f$: $\mathcal{E}_{\mathrm{v0}}(\mathrm{row}_i(\mathbf{U}_t^f))$ for $i \in \{0, \dots, mN]\}$.

The cloud service then locally computes the following, corresponding to lines 9 and 13 of Algorithm 7.2.1.

$$\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{y}}_t) = \rho(\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{y}}_{t-1}), p(S'+1)) + \rho(\mathcal{E}_{\mathrm{vr0}}(\mathbf{y}_t), (1-p)M(S'+1))$$
$$\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{u}}_t) = \rho(\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{u}}_{t-1}), m(S'+1)) + \rho(\mathcal{E}_{\mathrm{vr0}}(\mathbf{u}_t), (1-m)M(S'+1))$$
$$\mathcal{E}_{\mathrm{vr0}}(\mathrm{col}_{S'+1}H\mathbf{Y}_t) = \rho(\mathcal{E}_{\mathrm{vr0}}(\mathrm{col}_{S'}H\mathbf{Y}_t), p(S'+1)) + \rho(\mathcal{E}_{\mathrm{vr0}}(\mathbf{y}_t), -p(M+N-1)(S'+1))$$
$$\mathcal{E}_{\mathrm{vr0}}(\mathrm{col}_{S'+1}H\mathbf{U}_t) = \rho(\mathcal{E}_{\mathrm{vr0}}(\mathrm{col}_{S'}H\mathbf{U}_t), m(S'+1)) + \rho(\mathcal{E}_{\mathrm{vr0}}(\mathbf{u}_t), -m(M+N-1)(S'+1))$$
$$\mathcal{E}_{\mathrm{v0}}(\mathrm{row}_i(\mathbf{U}_t^f)) = \mathcal{E}_{\mathrm{v0}}(\mathrm{row}_i(\mathbf{U}_t^f)) + \rho(\mathcal{E}_{\mathrm{vr0}}(\mathrm{col}_{S'+1}(H\mathbf{U}_t)) \odot \mathbf{e}_{i(S+\bar{T})}, S'+1), i \in \{0, \dots, m\}$$
$$\mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{y}}_t, \mathbf{r}_t) = \mathcal{E}_{\mathrm{vr0}}(\bar{\mathbf{y}}_t) + \rho(\mathcal{E}_{\mathrm{vr0}}(\mathbf{r}_t), -pM(S'+1)).$$

The precollected input and output measurements have a multiplicative depth of 0. To reduce the total circuit depth, at time $t+1$, although the cloud service has computed the encryption of $\mathbf{u}_t$, the client sends a fresh encryption of it. This allows us to have, $\forall t \geq 0$: $d(\mathrm{cols}H\mathbf{U}_t) = d(\mathrm{cols}H\mathbf{Y}_t) = d(\bar{\mathbf{u}}_t) = d(\bar{\mathbf{y}}_t) = d(\mathbf{r}_t) = 0$. The update of the row representation of $\mathbf{U}^f$ needs a masking, so $d(\mathrm{rows}\mathbf{U}_f) = 1$. From the offline computations, we assume that we have fresh encryptions for the columns of $\mathbf{M}_0^{-1}$, hence $d(\mathrm{cols}\mathbf{M}_0^{-1}) = 0$.

The encrypted operations, their flow and motivations are described in Section 7.2.4.2. Compared to the previous approach in [15] (described in "Hybrid encoding"), we observed that we can avoid extra computations and a possible extra level in the computations if the server asks the client to compute $1/s_t$ immediately after it computes s_t. Hence, the server can compute directly the encryptions of $\mathbf{M}_t^{-1}$ and $\mathbf{u}_t$ instead of the numerators $N\mathbf{M}_t^{-1}$ and

$N\mathbf{u}_t$. The encrypted operations are described in (F.2.2)–(F.2.4), which correspond to lines 6 and 7 from Algorithm 7.2.1. $\text{EvalSum}_{S+\bar{T}}$ refers to rotating and summing batches of $S+\bar{T}$ slots in the ciphertexts and $\mathbf{v}_t := (\mathbf{m}_t\mathbf{M}_{t-1}^{-1})^\intercal$.

$$\begin{aligned}
&\mathcal{E}_{\text{vr}*}(\mu_t) = \text{EvalSum}_{S+\bar{T}}\big(\mathcal{E}_{\text{vr0}}(\text{col}_{S'+1}H\mathbf{Y}_{t-1}) \odot \text{e}_{\text{vr0}}(\lambda\mathbf{Q}) \odot \mathcal{E}_{\text{vr0}}(\text{col}_{S'+1}H\mathbf{Y}_{t-1}) + \\
&\quad + \mathcal{E}_{\text{vr0}}(\text{col}_{S'+1}H\mathbf{U}_{t-1}) \odot \text{e}_{\text{vr0}}(\lambda\mathbf{R}) \odot \mathcal{E}_{\text{vr0}}(\text{col}_{S'+1}H\mathbf{U}_{t-1}) + \text{e}_{\text{vr0}}(\lambda_g)\\
&\mathcal{E}_{\text{vr}*}(\mathbf{m}_{t[i]}) = \text{EvalSum}_{S+\bar{T}}\big(\mathcal{E}_{\text{vr0}}(\text{col}_{S'+1}H\mathbf{Y}_{t-1}) \odot \text{e}_{\text{vr0}}(\lambda\mathbf{Q}) \odot \mathcal{E}_{\text{vr0}}(\text{col}_{i}H\mathbf{Y}_{t-1}) + \\
&\quad + \mathcal{E}_{\text{vr0}}(\text{col}_{S'+1}H\mathbf{U}_{t-1}) \odot \text{e}_{\text{vr0}}(\lambda\mathbf{R}) \odot \mathcal{E}_{\text{vr0}}(\text{col}_{i}H\mathbf{U}_{t-1}),\ i \in \{0,\dots,S'+1\}\\
&\mathcal{E}_{\text{v0}}(\mathbf{v}_t) = \sum_{i=0}^{S'} \mathcal{E}_{\text{v0}}(\text{col}_i(\mathbf{M}_{t-1}^{-1})) \odot \mathcal{E}_{\text{vr}*}(\mathbf{m}_{t[i]}) \qquad\qquad \text{(F.2.2)}\\
&\mathcal{E}_{\text{vr}*}(s_t) = \mathcal{E}_{\text{vr}*}(\mu_t) - \sum_{i=0}^{S'} \mathcal{E}_{\text{vr}*}(\mathbf{m}_{t[i]}) \odot \rho\left(\mathcal{E}_{\text{v0}}(\mathbf{v}_t), i\right).
\end{aligned}$$

After receiving a fresh encryption of $1/s_t$ from the client:

$$\begin{aligned}
&\mathcal{E}_{\text{v0}}(\text{col}_{S'+1}(\mathbf{M}_t^{-1})) = (-\mathcal{E}_{\text{v0}}(\mathbf{v}_t) + \mathbf{e}_{S'+1}) \odot \mathcal{E}_{\text{vr0}}(1/s_t)\\
&\mathcal{E}_{\text{v0}}(\mathbf{w}_i) = \sum_{j=0}^{S'} \mathcal{E}_{\text{v0}}(\text{col}_j\mathbf{M}_{t-1}^{-1}) \odot \big(\mathcal{E}_{\text{vr}*}(\mathbf{m}_{t[i]}) \odot \mathcal{E}_{\text{v0}}(1/s_t\,\mathbf{e}_i)\big)\\
&\mathcal{E}_{\text{v0}}(\text{col}_i(1/s_t\mathbf{v}_t\mathbf{v}_t^\intercal)) = \mathcal{E}_{\text{v0}}(\mathbf{v}_t) \odot \sum_{j=0}^{S'} \rho\left(\mathcal{E}_{\text{v0}}(\mathbf{w}_i), i-j\right),\ i \in \{0,\dots,S'+1\}\\
&\mathcal{E}_{\text{v0}}(\text{col}_i(\mathbf{M}_t^{-1})) = \mathcal{E}_{\text{v0}}(\text{col}_i\mathbf{M}_t^{-1}) - \mathcal{E}_{\text{v0}}(\text{col}_i(1/s_t\,\mathbf{v}_t\mathbf{v}_t^\intercal)) + \qquad\qquad \text{(F.2.3)}\\
&\quad + \rho\left(\mathcal{E}_{\text{v0}}(\mathbf{v}_t) \odot (-\mathbf{e}_i \odot \mathcal{E}_{\text{vr0}}(1/s_t)), S'+1\right),\ i \in \{0,\dots,S'+2\}\\
&\mathcal{E}_{\text{vr}*}(\mathbf{Z}_{t[j]}) = \text{EvalSum}_{S+\bar{T}}\big(\mathcal{E}_{\text{vr0}}(\text{col}_jH\mathbf{Y}_t) \odot \text{e}_{\text{vr0}}(\lambda\mathbf{Q}) \odot \mathcal{E}_{\text{vr0}}(\bar{\mathbf{y}}_t, \mathbf{r}_t) + \\
&+ \mathcal{E}_{\text{vr0}}(\text{col}_jH\mathbf{U}_t) \odot \text{e}_{\text{vr0}}(\lambda\mathbf{R}) \odot \mathcal{E}_{\text{vr0}}(\bar{\mathbf{u}}_t)\big),\ i \in \{0,\dots,S'+2\}\\
&\mathcal{E}_{\text{v0}}(\boldsymbol{v}_{t,i}) = \sum_{k=0}^{S'+1}\sum_{j=0}^{S'+1} \mathcal{E}_{\text{v0}}(\text{col}_k(\mathbf{M}_t^{-1})) \odot \Big(\mathcal{E}_{\text{v0}}(\text{row}_i\mathbf{U}_t^f) \odot \mathcal{E}_{\text{vr}*}(\mathbf{Z}_{t[j]})\Big),\ i \in \{0,\dots,m\}\\
&\mathcal{E}_{\text{v}*}(\boldsymbol{v}_t) = \sum_{i=0}^{m-1} \rho\left(\mathcal{E}_{\text{v0}}(\boldsymbol{v}_{t,i}, i(S+\bar{T})\right).
\end{aligned}$$

After decryption: $$\mathbf{u}_{t[i]} = \sum_{j=i(S+\bar{T})}^{i(S+\bar{T})+S'+1} \boldsymbol{v}_{t,i},\ i \in \{0,\dots,m\}. \tag{F.2.4}$$

Following (F.2.2)–(F.2.4), we get the depth expressions for $t \geq 1$:

$$d(\mathbf{M}_t^{-1}) = 2t+3, \quad d(\mathbf{u}_t) = d(\boldsymbol{v}_t) = 2t+4. \tag{F.2.5}$$

Remark F.2.1. We make an abuse of notation when writing that we obtain $\mathcal{E}_{\mathrm{v0}}(\cdot)$ and $\mathcal{E}_{\mathrm{vr0}}(\cdot)$ after the rotation of a ciphertext with trailing zeros. Actually, the initial nonzero values will be shifted to the end of the encoded vector and we need to maintain a counter on how many times we can perform these rotations. In the vector packing case, we need the parameters to satisfy the following rules, before a masking is necessary:

$$\begin{aligned} &\mathrm{ringDim}/2 > \max(m,p)(N+M+\bar{T})(S+\bar{T}) \\ &\mathrm{ringDim}/2 > (S+\bar{T})^2 \\ &\mathrm{ringDim}/2 > \max(mM, p(N+M))t(S+\bar{T}), \end{aligned} \tag{F.2.6}$$

where the first line is for the online collection of samples ($\bar{T}$ is the total number of samples collected online), the second line is for the correct packing of the matrix into one ciphertext and the last line is the rule for after finalizing the collection of new samples $t \geq \bar{T}$ (instead of masking, here we can ask the client to prune the junk elements in $\bar{\mathbf{u}}, \bar{\mathbf{y}}$).

It is more efficient (and not problematic for the client when M is small) that after the adaptation ends, the client sends to the server encryptions of $\bar{\mathbf{u}}_t$ and $\bar{\mathbf{y}}_t$ instead of $\mathbf{u}_t$ and $\mathbf{y}_t$. Then, the last line of (F.2.6) is not needed and the computation can go on without the need of refreshing. If the rules in Remark F.2.1 are not satisfied for the desired parameters of an application, the values can be split into the necessary number of ciphertexts (each packing up to ringDim/2 values) and the computations can be readily extended to deal with this.

Remark F.2.2. Note that one can use the inverse and forward Fast Fourier Transform (IFFT and FFT) to multiply a Hankel matrix by a vector using $\mathrm{IFFT}(\mathrm{FFT}(\mathrm{vec}(\mathbf{H})) \circ \mathrm{FFT}(\mathbf{v}))$, and reassembling the resulting vector afterwards, where vec refers to vectorizing the Hankel matrix. Since the CKKS decoding and encoding use IFFT and FFT, we can avoid some plaintext encoding and decoding operations. However, we need to encode the ciphertext of the result to be able to further operate on it, which ends up canceling the advantage of the above observation. For other specific applications, where the resulting ciphertext is the final output, using the above observation to perform the multiplication would be preferable.

F.3 Proof of Lemma 7.2.6

The following identities can be found in [32, Ch. 1,3]:

$$\lim_{\delta\to 0^+} \mathbf{A}^*(\delta\mathbf{I}+\mathbf{A}\mathbf{A}^*)^{-1} = \mathbf{A}^\dagger \tag{F.3.1}$$

$$\mathbf{A}^\dagger = \mathbf{A}^*\mathbf{A}^{\dagger *}\mathbf{A}^\dagger. \tag{F.3.2}$$

(a) Using (F.3.1), we obtain:

$$\begin{aligned}\lim_{\lambda_g\to 0^+} s &= \mathbf{h}^\intercal\mathbf{P}\mathbf{h} - \mathbf{h}^\intercal\mathbf{P}^{1/2}\big(\lim_{\lambda_g\to 0^+}\mathbf{P}^{1/2}\mathbf{H}(\mathbf{H}^\intercal\mathbf{P}\mathbf{H}+\lambda_g\mathbf{I})^{-1}\mathbf{H}^\intercal\mathbf{P}^{1/2}\big)\mathbf{P}^{1/2}\mathbf{h}\\ &= \mathbf{h}^\intercal\mathbf{P}^{1/2}\left(\mathbf{I}-(\mathbf{P}^{1/2}\mathbf{H})(\mathbf{P}^{1/2}\mathbf{H})^\dagger\right)\mathbf{P}^{1/2}\mathbf{h},\end{aligned}$$

Furthermore, $\big(\mathbf{I}-(\mathbf{P}^{1/2}\mathbf{H})(\mathbf{P}^{1/2}\mathbf{H})^\dagger\big)$ is orthogonal onto the range of $\mathbf{P}^{1/2}\mathbf{H}$. From the behavioral system definition (under the condition of persistancy of excitation) $\mathbf{P}^{1/2}\mathbf{h} \in \mathrm{Range}(\mathbf{P}^{1/2}\mathbf{H})$, so $\lim_{\lambda_g\to 0^+} s = 0$.

(b) We use the L'Hôspital rule (because $\lim_{\lambda_g\to 0^+} s = 0$):

$$\begin{aligned}\lim_{\lambda_g\to 0^+}\frac{s}{\lambda_g} &= \lim_{\lambda_g\to 0^+}\frac{\partial s/\partial\lambda_g}{\partial\lambda_g/\partial\lambda_g} = 1 + \lim_{\lambda_g\to 0^+}\mathbf{h}^\intercal\mathbf{P}\mathbf{H}(\mathbf{H}^\intercal\mathbf{P}\mathbf{H}+\lambda_g\mathbf{I})^{-2}\mathbf{H}^\intercal\mathbf{P}\mathbf{h}\\ &= 1 + \mathbf{h}^\intercal\mathbf{P}^{1/2}(\mathbf{H}^\intercal\mathbf{P}^{1/2})^\dagger(\mathbf{P}^{1/2}\mathbf{H})^\dagger\mathbf{P}^{1/2}\mathbf{h},\end{aligned}$$

where we used the pseudoinverse limit definition (F.3.1).

(c) The second part goes to 0 as $\lambda_g \to \infty$:

$$\lim_{\lambda_g \to \infty} \frac{s}{\lambda_g} = 1 + \lim_{\lambda_g \to \infty} \frac{\mathbf{h}^\intercal \mathbf{P} \mathbf{h}}{\lambda_g} - \frac{\mathbf{h}^\intercal \mathbf{P} \mathbf{H} (\mathbf{H}^\intercal \mathbf{P} \mathbf{H} + \lambda_g \mathbf{I})^{-1} \mathbf{H}^\intercal \mathbf{P} \mathbf{h}}{\lambda_g}.$$

(d) The proof is laborious so, for conciseness, we do not present the complete derivations. We show that $f(\lambda_g)$ is decreasing by showing that $\frac{\partial f(\lambda_g)}{\partial \lambda_g} := -\frac{g(\lambda_g)}{\lambda_g^2}$ takes only negative values, which we show by proving $\lim\limits_{\lambda_g \to 0^+} g(\lambda_g) \geq 0$ and $\frac{\partial g(\lambda_g)}{\partial \lambda_g} \geq 0$. To this end, we used again the pseudoinverse limit definitions and that $(\mathbf{P}^{1/2}\mathbf{H})(\mathbf{P}^{1/2}\mathbf{H})^\dagger$ is an orthogonal projection onto the range of $(\mathbf{P}^{1/2}\mathbf{H})$ and hence, its eigenvalues are in $\{0, 1\}$. □

Let us investigate (b) further. Let $\mathbf{h} = \mathbf{H}\boldsymbol{\eta}$ for some vector $\boldsymbol{\eta}$, that in slowly-varying systems has a small magnitude. Then, by equation (F.3.2), we get that:

$$\lim_{\lambda_g \to 0^+} \frac{\partial s / \partial \lambda_g}{\partial \lambda_g / \partial \lambda_g} = 1 + \boldsymbol{\eta}^\intercal (\mathbf{P}^{1/2}\mathbf{H})^\dagger (\mathbf{P}^{1/2}\mathbf{H}) \boldsymbol{\eta}.$$

$(\mathbf{P}^{1/2}\mathbf{H})^\dagger(\mathbf{P}^{1/2}\mathbf{H})$ is an orthogonal projection operator, i.e., its eigenvalues are in $\{0, 1\}$, meaning that the quantity $\boldsymbol{\eta}^\intercal (\mathbf{P}^{1/2}\mathbf{H})^\dagger (\mathbf{P}^{1/2}\mathbf{H}) \boldsymbol{\eta}$ has a small magnitude. Then, the second term in (b) is small.

F.4 Privacy of the encrypted algorithms

Our goal is to satisfy client privacy with respect to the semi-honest behavior of the server, captured in Definition 2.2.8. Recall that x_C denotes the input of the client, x_S denotes the input of the server. The functionality f in Definition 2.2.8 represents in this case the data-driven algorithm, and $f_S(x_C, x_S)$ represents the output of the server after evaluating the functionality of the given inputs.

Fix a number of time steps $\mathcal{T}$. We want to prove a secure evaluation of the data-driven control functionality for $\mathcal{T}$ steps, after which we assume the protocol ends. In reality, because the security of the encryption scheme used is based on computational problems, after a long time (years), it is recommended that the secret key is changed; so that represents a natural stopping point $\mathcal{T}$ and is not restrictive.

Under the aforementioned condition, the output of the protocol at the server, $f_S(x_C, x_S)$, is actually the empty set. Furthermore, all the intermediate messages in the view of the server will be ciphertext encrypted with the client's public key using the CKKS scheme. Finally, the ring dimension of the CKKS scheme is selected such that, for the multiplicative budget of the desired functionality, an adversary cannot brute force the encryption. These are the main observations that will be used in proving the subsequent theorems.

There is an preprocessing protocol, where the server gets and computes the encrypted information it will have to use for the online control of the system, i.e., the encrypted data trajectories, costs and parameters. This preprocessing protocol is proven secure under the same Definition 2.2.8, hence, there is an ideal functionality $\mathcal{F}^{pre}$, which is essentially indistinguishable from the real-world functionality of the preprocessing protocol. We work in a $\mathcal{F}^{pre}$-hybrid model, which means that we can securely compose this functionality with the functionality of the online protocols (see [152] for more details about simulation proofs in the hybrid model and composition theorems).

Proof of Theorem 7.2.3. At time $t \leq \mathcal{T}$, the client's input is $H\mathbf{U}$, $H\mathbf{Y}$, $\mathbf{u}_{0:t-1}$, $\mathbf{y}_{0:t-1}$, $\mathbf{r}_{0:t-1}$ and $\mathbf{pk}$, where $\mathbf{pk}$ is the public key of the leveled homomorphic scheme used. We assume that, after the preprocessing protocol, the server has input $\mathbf{Q}$, $\mathbf{R}$, λ_g, λ_u, λ_y, $\mathbf{pk}$ and receives as messages $\mathcal{E}_{\text{v0}}(\text{diag}_i\mathbf{A}_r)$, $\mathcal{E}_{\text{v0}}(\text{diag}_i\mathbf{A}_y)$, $\mathcal{E}_{\text{v0}}(\text{diag}_i\mathbf{A}_u)$. At time t, the server receives from the client either $\mathcal{E}_{\text{vv}}(\bar{\mathbf{u}}_t)$, $\mathcal{E}_{\text{vv}}(\bar{\mathbf{y}}_t)$, $\mathcal{E}_{\text{vv}}(\mathbf{r}_t)$ or $\mathcal{E}_{\text{vv}}(\bar{\mathbf{y}}_t)$, $\mathcal{E}_{\text{vv}}(\mathbf{r}_t)$. After applying the functionality as described in Section 7.2.3, the server obtains $\mathcal{E}_{\text{v0}}(\mathbf{u}_t)$ or $\mathcal{E}_{\text{v}*}(\mathbf{u}_t)$ and sends it to the client.

The proof immediately follows from the fact that the server only receives fresh ciphertexts from the client. Since the homomorphic encryption scheme is semantically secure, these ciphertexts are computationally indistinguishable from random encryptions, so we can construct a simulator for the server that replaces the true messages from the client by random encryptions of the same size. The input-output distribution of such a simulator will be indistinguishable from the input-output distribution of the true protocol. □

Proof of Theorem 7.2.7. The input of the client is the same as in the proof of Theorem 7.2.3.

The input of the server is $\mathbf{Q}, \mathbf{R}, \lambda_g, \lambda_u, \lambda_y, \mathbf{pk}$ and after the preprocessing protocol, it has the corresponding ciphertexts for $H\mathbf{U}_0, H\mathbf{Y}_0, \mathbf{M}_0^{-1}, \mathbf{U}_0^f$. At time $t < \bar{T}$, while collecting new samples, the server has ciphertexts corresponding to $H\mathbf{U}_{t-1}, H\mathbf{Y}_{t-1}, \mathbf{M}_{t-1}^{-1}, \bar{\mathbf{u}}_{t-1}, \bar{\mathbf{y}}_{t-1}$, receives from the client $\mathcal{E}_{\mathrm{vr0}}(\mathbf{u}_t), \mathcal{E}_{\mathrm{vr0}}(\mathbf{y}_t), \mathcal{E}_{\mathrm{vr0}}(\mathbf{r}_t)$, then computes $\mathcal{E}_{\mathrm{vr}*}(s_t)$ and receives from the client $\mathcal{E}_{\mathrm{vr0}}(1/s_t)$. The analysis in Section 7.2.4.3 does not reveal private information: although the cloud server knows λ_g, it never gets s_t in cleartext. Finally, after applying the functionality described in Section 7.2.4 and Section F.2, the server obtains the corresponding ciphertexts of $H\mathbf{U}_t, H\mathbf{Y}_t, \mathbf{M}_t^{-1}, \mathbf{U}_t^f, \bar{\mathbf{u}}_t, \bar{\mathbf{y}}_t$ and $\mathcal{E}_{\mathrm{v}*}(\boldsymbol{\upsilon}_t)$, and sends the latter to the client, which decrypts it and computes $\mathbf{u}_t$. During the times designated for refreshing $\mathbf{M}_t^{-1}$, the server packs it into a single ciphertext, sends it to the client and receives $\mathcal{E}_{\mathrm{v0}}(\mathbf{M}_t^{-1})$, which uses new uncorrelated randomness. After the collection of the new samples ends, i.e., $\bar{T} \leq t \leq \mathcal{T}$, the server only updates the ciphertexts corresponding to the quantities $\bar{\mathbf{u}}_t$ and $\bar{\mathbf{y}}_t$, but otherwise computes the same output.

Since the server only receives fresh encryptions of the private data, we can construct a simulator for the server that replaces the true messages from the client by random encryption of corresponding size and apply the server's functionality. Privacy follows because of the semantic security of the underlying CKKS scheme. $\square$

Bibliography

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] G. Ács and C. Castelluccia. I have a DREAM!(DiffeRentially privatE smArt Metering). In *International Workshop on Information Hiding*, pages 118–132. Springer, 2011.

[3] S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *Annual International Cryptology Conference*, pages 333–362. Springer, 2016.

[4] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9:169–203, 2015. `https://lwe-estimator.readthedocs.io/en/latest`.

[5] M. R. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, et al. Homomorphic encryption standard. *IACR Cryptol. ePrint Arch.*, 2019:939, 2019.

[6] A. B. Alexandru and G. J. Pappas. Encrypted LQG using Labeled Homomorphic Encryption. In *Proceedings of 10th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 129–140, 2019.

[7] A. B. Alexandru and G. J. Pappas. Private weighted sum aggregation for distributed control systems. *IFAC-PapersOnLine*, 53(2):11081–11088, 2020. 21th IFAC World Congress.

[8] A. B. Alexandru and G. J. Pappas. Private weighted sum aggregation. *arXiv preprint arXiv:2010.10640*, 2020.

[9] A. B. Alexandru and G. J. Pappas. Secure multi-party computation for cloud-based control. In *Privacy in Dynamical Systems*, pages 179–207. Springer, 2020.

[10] A. B. Alexandru, K. Gatsis, and G. J. Pappas. Privacy preserving cloud-based quadratic optimization. In *55th Annual Allerton Conference on Communication, Control, and Computing*, pages 1168–1175. IEEE, 2017.

[11] A. B. Alexandru, M. Morari, and G. J. Pappas. Cloud-based MPC with encrypted data. In *Proceedings of the 57th IEEE Conference on Decision and Control (CDC)*, pages 5014–5019, 2018.

[12] A. B. Alexandru, M. Schulze Darup, and G. J. Pappas. Encrypted cooperative control revisited. In *Proceedings of the 58th Conference on Decision and Control (CDC)*, pages 7196–7202. IEEE, 2019.

[13] A. B. Alexandru, K. Gatsis, Y. Shoukry, S. A. Seshia, P. Tabuada, and G. J. Pappas. Cloud-based quadratic optimization with partially homomorphic encryption. *IEEE Transactions on Automatic Control*, 2020.

[14] A. B. Alexandru, A. Tsiamis, and G. J. Pappas. Data-driven control on encrypted data. *arXiv preprint arXiv:2008.12671*, 2020.

[15] A. B. Alexandru, A. Tsiamis, and G. J. Pappas. Towards private data-driven control. In *Proceedings of the 59th Conference on Decision and Control (CDC)*, pages 5449–5456. IEEE, 2020.

[16] A. B. Alexandru, A. Tsiamis, and G. J. Pappas. Encrypted distributed Lasso for sparse data predictive control. *arXiv preprint arXiv:2104.11632*, 2021.

[17] M. Ali, S. U. Khan, and A. V. Vasilakos. Security in cloud computing: Opportunities and challenges. *Information sciences*, 305:357–383, 2015,.

[18] M. P. Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H.-S. Kim, D. E. Culler, and R. A. Popa. WAVE: A decentralized authorization framework with transitive delegation. In *28th USENIX Security Symposium*, pages 1375–1392, 2019.

[19] D. Archer, L. Chen, J. H. Cheon, R. Gilad-Bachrach, R. A. Hallman, Z. Huang, X. Jiang, R. Kumaresan, B. A. Malin, H. Sofia, Y. Song, and S. Wang. Applications of homomorphic encryption. Technical report, Microsoft Research, 2017.

[20] M. Aristov, B. Noack, U. D. Hanebeck, and J. Müller-Quade. Encrypted multisensor information filtering. In *21st International Conference on Information Fusion*, pages 1631–1637, Cambridge, UK, 2018. IEEE.

[21] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, 2017.

[22] L. J. Aslett, P. M. Esperança, and C. C. Holmes. A review of homomorphic encryption and software tools for encrypted statistical machine learning. *arXiv preprint arXiv:1508.06574*, 2015.

[23] G. Aspin, G. Collins, P. Krumkachev, M. Metzger, S. Radeztsky, and S. Srinivasan. Everything-as-a-service: Modernizing the core through a services lens. In *Tech trends: The kinetic enterprise*, pages 78–91. Deloitte University Trends, 2017.

[24] M. Athans. The discrete time Linear-Quadratic-Gaussian stochastic control problem. In *Annals of Economic and Social Measurement*, volume 1, no. 4, pages 449–491. NBER, New York, NY, USA, 1972.

[25] G. Baggio, V. Katewa, and F. Pasqualetti. Data-driven minimum-energy controls for linear systems. *IEEE Control Systems Letters*, 3(3):589–594, 2019.

[26] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens. Pretp: privacy-preserving electronic toll pricing. In *Proceedings of the 19th USENIX Conf. on Security*, 2010.

[27] M. Barbosa, D. Catalano, and D. Fiore. Labeled homomorphic encryption. In *European Symposium on Research in Computer Security*, pages 146–166. Springer, 2017.

[28] D. Becker, J. Guajardo, and K.-H. Zimmermann. Revisiting private stream aggregation: Lattice-based PSA. In *Network & Distributed System Security Symposium (NDSS)*, 2018.

[29] A. Beimel. Secret-sharing schemes: a survey. In *International Conference on Coding and Cryptology*, pages 11–46. Springer, 2011.

[30] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *Proceedings of the Conference on Computer and Communications Security*, pages 784–796. ACM, 2012.

[31] S. M. Bellovin. Frank Miller: Inventor of the one-time pad. *Cryptologia*, 35(3):203–222, 2011.

[32] A. Ben-Israel and T. N. Greville. *Generalized inverses: theory and applications*, volume 15. Springer Science & Business Media, 2003.

[33] F. Benhamouda, M. Joye, and B. Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Transactions on Information and System Security*, 18(3):10, 2016.

[34] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin. Can a public blockchain keep a secret? In *Theory of Cryptography Conference*, pages 260–290. Springer, 2020.

[35] J. Berberich, J. Köhler, M. A. Muller, and F. Allgower. Data-driven model predictive control with stability and robustness guarantees. *IEEE Transactions on Automatic Control*, 2020.

[36] J. Berberich, J. Köhler, M. A. Müller, and F. Allgöwer. Robust constraint satisfaction in data-driven MPC. In *Proceedings of the 59th Conference on Decision and Control (CDC)*, pages 1260–1267. IEEE, 2020.

[37] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 313–314. Springer, 2013.

[38] D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific, Belmont, MA, USA, 2012.

[39] D. Bickson, T. Reinman, D. Dolev, and B. Pinkas. Peer-to-peer secure multi-party numerical computation facing malicious adversaries. *Peer-to-peer networking and applications*, 3(2):129–144, 2010.

[40] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff, and V. Vaikuntanathan. Optimized homomorphic encryption solution for secure genome-wide association studies. *BMC Medical Genomics*, 13(7):1–13, 2020.

[41] 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 *ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[42] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography*, volume 3378, pages 325–341. Springer, 2005.

[43] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.

[44] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[45] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[46] A. Botta, W. De Donato, V. Persico, and A. Pescapé. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, 56:684–700, 2016.

[47] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[48] S. Boyd, N. Parikh, and E. Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[49] Z. Brakerski. Fundamentals of fully homomorphic encryption. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 543–563. ACM, 2019.

[50] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[51] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6 (3):13, 2014.

[52] E. J. Candes, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(8):1207–1223, 2006.

[53] A. A. Cardenas, S. Amin, and S. Sastry. Secure control: Towards survivable cyber-physical systems. In *28th International Conference on Distributed Computing Systems Workshops*, pages 495–500, 2008.

[54] C. Castelluccia, A. C. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 5(3):20, 2009.

[55] D. Catalano and D. Fiore. Boosting linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. Cryptology ePrint Archive, Report 2014/813, 2015. `https://eprint.iacr.org/2014/813`.

[56] D. Catalano and D. Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1518–1529. ACM, 2015.

[57] H. T. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *International Conference on Financial Cryptography and Data Security*, pages 200–214. Springer, 2012.

[58] M. Chase, R. Gilad-Bachrach, K. Laine, K. E. Lauter, and P. Rindal. Private collaborative neural network learning. *IACR Cryptology ePrint Archive*, 2017:762, 2017.

[59] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[60] H. Chen, I. Chillotti, and Y. Song. Improved bootstrapping for approximate homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 34–54. Springer, 2019.

[61] T. M. Chen and S. Abu-Nimeh. Lessons from Stuxnet. *Computer*, 44(4):91–93, 2011.

[62] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[63] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

[64] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 347–368. Springer, 2018.

[65] J. H. Cheon, K. Han, H. Kim, J. Kim, and H. Shim. Need for controllers having integer coefficients in homomorphically encrypted dynamic system. In *Proceedings of the 57th Conference on Decision and Control (CDC)*, pages 5020–5025. IEEE, 2018.

[66] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[67] M. S. Chong, H. Sandberg, and A. M. H. Teixeira. A tutorial introduction to security and privacy for cyber-physical systems. In *Proc. of the 2019 European Control Conference*, pages 968–978, 2019.

[68] C. J. Cleveland, R. K. Kaufmann, and D. I. Stern. Aggregation and the role of energy in the economy. *Ecological Economics*, 32(2):301–317, 2000.

[69] J.-P. Corfmat and A. S. Morse. Decentralized control of linear multivariable systems. *Automatica*, 12(5):479–495, 1976.

[70] J. Coulson, J. Lygeros, and F. Dörfler. Data-enabled predictive control: In the shallows of the DeePC. In *Proceedings of the 18th European Control Conference (ECC)*, pages 307–312. IEEE, 2019.

[71] J. Coulson, J. Lygeros, and F. Dörfler. Regularized and distributionally robust data-enabled predictive control. In *Proceedings of the 58th Conference on Decision and Control (CDC)*, 2019.

[72] J. Coulson, J. Lygeros, and F. Dörfler. Distributionally robust chance constrained data-enabled predictive control. *arXiv preprint arXiv:2006.01702*, 2020.

[73] G. Couteau. New protocols for secure equality test and comparison. In *International Conference on Applied Cryptography and Network Security*, pages 303–320. Springer, 2018.

[74] R. Cramer and I. B. Damgård. *Secure Multiparty Computation.* Cambridge University Press, 2015.

[75] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–300. Springer, 2001.

[76] J. Daemen and V. Rijmen. The rijndael block cipher: AES proposal. In *First candidate conference (AeS1)*, pages 343–348, 1999.

[77] I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian Conference on Information Security and Privacy*, pages 416–430. Springer, 2007.

[78] I. Damgård, M. Geisler, and M. Krøigaard. A correction to "Efficient and secure comparison for on-line auctions". *International Journal of Applied Cryptography*, 1(4): 323–324, 2009.

[79] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[80] I. B. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*, pages 119–136. Springer, 2001.

[81] F. K. Dankar and K. El Emam. The application of differential privacy to health data. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 158–166. ACM, 2012.

[82] C. De Persis and P. Tesi. On persistency of excitation and formulas for data-driven control. In *Proceedings of the 58th Conference on Decision and Control (CDC)*, pages 873–878. IEEE, 2019.

[83] F. Dörfler, J. Coulson, and I. Markovsky. Bridging direct & indirect data-driven control formulations via regularizations and relaxations. *arXiv preprint arXiv:2101.01273*, 2021.

[84] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[85] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Annual International Cryptology Conference*, pages 528–544. Springer, 2004.

[86] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[87] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[88] R. El Bansarkhani, Ö. Dagdelen, and J. Buchmann. Augmented learning with errors: The untapped potential of the error term. In *International Conference on Financial Cryptography and Data Security*, pages 333–352. Springer, 2015.

[89] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Pérez-González. Privacy-preserving data aggregation in smart metering systems: An overview. *IEEE Signal Processing Magazine*, 30(2):75–86, 2013.

[90] F. Fagnani and S. Zampieri. Average consensus with packet drop communication. *SIAM Journal on Control and Optimization*, 48(1):102–133, 2009.

[91] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[92] F. Farokhi, I. Shames, and N. Batterham. Secure and private control using semi-homomorphic encryption. *Control Engineering Practice*, 67:13–20, 2017.

[93] W. Favoreel, B. De Moor, P. an Overschee, and M. Gevers. Model-free subspace-based LQG-design. In *Proceedings of the American Control Conference (ACC)*, volume 5, pages 3372–3376. IEEE, 1999.

[94] J. Ferreau, H. Peyrl, D. Kouzoupis, and A. Zanelli. MPC Benchmarking Collection. `https://github.com/ferreau/mpcBenchmarking`, 2010.

[95] N. M. Freris and P. Patrinos. Distributed computing over encrypted data. In *54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1116–1122. IEEE, 2016.

[96] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat, and J.-P. Hubaux. Scalable privacy-preserving distributed learning. *Proceedings on Privacy Enhancing Technologies*, 2021(2):323–347, 2021.

[97] S. Gade and N. H. Vaidya. Private learning on networks. *arXiv preprint arXiv:1612.05236*, 2016.

[98] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342. Springer, 2007.

[99] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017(4):345–364, 2017.

[100] T. Ge and S. Zdonik. Answering aggregation queries in a secure system model. In *Proceedings of the 33rd international conference on Very Large Data Bases*, pages 519–530. VLDB Endowment, 2007.

[101] N. Genise and D. Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 174–203. Springer, 2018.

[102] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Department of Computer Science, Stanford University, 2009.

[103] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*, pages 75–92. Springer, 2013.

[104] O. Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2003.

[105] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[106] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the 19th symposium on Theory of computing*, pages 218–229. ACM, 1987.

[107] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the 14th symposium on Theory of Computing*, pages 365–377. ACM, 1982.

[108] F. J. Gonzalez-Serrano, A. Amor-Martın, and J. Casamayon-Anton. State estimation using an extended Kalman filter with privacy-protected observed inputs. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 54–59. IEEE, 2014.

[109] U. Greveler, P. Glösekötterz, B. Justusy, and D. Loehr. Multimedia content identification through smart meter power usage profiles. In *Proceedings of the International Conference on Information and Knowledge Engineering*, page 1, 2012.

[110] A. Gupta, C. Langbort, and T. Basar. Optimal control in the presence of an intelligent jammer with limited actions. In *Proceedings of the 49th Conference on Decision and Control (CDC)*, pages 1096–1101, 2010.

[111] O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[112] C. N. Hadjicostis. Privacy preserving distributed average consensus via homomorphic encryption. In *Proceedings of the 57th Conference on Decision and Control (CDC)*, pages 1258–1263, 2018.

[113] C. N. Hadjicostis and A. D. Dominguez-Garcia. Privacy-preserving distributed averaging via homomorphically encrypted ratio consensus. *IEEE Transactions on Automatic Control*, 2020.

[114] M. Hale, A. Jones, and K. Leahy. Privacy in feedback: The differentially private LQG. In *American Control Conference (ACC)*, pages 3386–3391, Milwaukee, WI, USA, 2018. IEEE.

[115] S. Halevi and V. Shoup. Algorithms in HElib. In *Annual Cryptology Conference*, pages 554–571. Springer, 2014.

[116] S. Halevi and V. Shoup. Faster homomorphic linear transformations in HElib. In *Annual International Cryptology Conference*, pages 93–120. Springer, 2018.

[117] S. Halevi, Y. Polyakov, and V. Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *Cryptographers' Track at the RSA Conference*, pages 83–105. Springer, 2019.

[118] HAMLab (Heat, Air and Moisture Laboratory). `https://archbpswiki.bwk.tue.nl/bpswiki/index.php/Hamlab`, 2007.

[119] A. Hamlin, N. Schear, E. Shen, M. Varia, S. Yakoubov, and A. Yerukhimovich. Cryptography for big data security. In F. Hu, editor, *Big Data: Storage, Sharing, and Security*, chapter 10, pages 241–288. Taylor & Francis LLC, CRC Press, 2016.

[120] S. Han, W. K. Ng, L. Wan, and V. C. Lee. Privacy-preserving gradient-descent methods. *IEEE Transactions on Knowledge and Data Engineering*, 22(6):884–899, 2010.

[121] S. Han, U. Topcu, and G. J. Pappas. Differentially private distributed constrained optimization. *IEEE Transactions on Automatic Control*, 62(1):50–64, 2017.

[122] 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.

[123] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical learning with sparsity: the lasso and generalizations.* CRC press, 2015.

[124] Ø. Hegrenæs, J. T. Gravdahl, and P. Tøndel. Spacecraft attitude control using explicit model predictive control. *Automatica*, 41(12):2107–2114, 2005.

[125] L. Huang, J. Coulson, J. Lygeros, and F. Dörfler. Data-enabled predictive control for grid-connected power converters. In *Proceedings of the 58th Conference on Decision and Control (CDC)*, 2019.

[126] Z. Huang, S. Mitra, and N. Vaidya. Differentially private distributed optimization. In *Proceedings of the International Conference on Distributed Computing and Networking*, pages 1–10. ACM, 2015.

[127] IDASH Privacy & Security Workshop - secure genome analysis competition. http://www.humangenomeprivacy.org/, 2020.

[128] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer–efficiently. In *Annual International Cryptology Conference*, pages 572–591. Springer, 2008.

[129] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, 59(12):3238–3251, 2014.

[130] M. Joye and B. Libert. Efficient cryptosystems from 2^k-th power residue symbols. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 76–92. Springer, 2013.

[131] M. Joye and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *International Conference on Financial Cryptography and Data Security*, pages 111–125. Springer, 2013.

[132] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium*, pages 1651–1669, 2018.

[133] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* Chapman & Hall/CRC, 2007.

[134] J. Kim, C. Lee, H. Shim, J. H. Cheon, A. Kim, M. Kim, and Y. Song. Encrypting controller using fully homomorphic encryption for security of cyber-physical systems. *IFAC-PapersOnLine*, 49(22):175–180, 2016.

[135] J. Kim, H. Shim, and K. Han. Dynamic controller that operates over homomorphically encrypted data for infinite time horizon. *arXiv preprint arXiv:1912.07362*, 2019.

[136] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations*, 2017.

[137] W. Knowles, D. Prince, D. Hutchiso, J. F. P. Disso, and K. Jones. A survey of cyber security management in industrial control systems. *International Journal of Critical Infrastructure Protection*, 9:52–80, 2015.

[138] K. Kogiso. Upper-bound analysis of performance degradation in encrypted control system. In *American Control Conference (ACC)*, pages 1250–1255, Milwaukee, WI, USA, 2018. IEEE.

[139] K. Kogiso and T. Fujita. Cyber-security enhancement of networked control systems using homomorphic encryption. In *Proceedings of the 54th Conference on Decision and Control (CDC)*, pages 6836–6843, 2015.

[140] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *International Conference on Cryptology and Network Security*, pages 1–20. Springer, 2009.

[141] F. Koufogiannis and G. J. Pappas. Differential privacy for dynamical sensitive data. In *Proceedings of the 56th Conference on Decision and Control (CDC)*, pages 1118–1125. IEEE, 2017.

[142] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 175–191. Springer, 2011.

[143] B. Kuykendall, H. Krawczyk, and T. Rabin. Cryptography for# metoo. *Proceedings on Privacy Enhancing Technologies*, 3:409–429, 2019.

[144] J. Le Ny and G. J. Pappas. Differentially private filtering. *IEEE Transactions on Automatic Control*, 59(2):341–354, 2014.

[145] A. S. Leong, D. E. Quevedo, D. Dolz, and S. Dey. On remote state estimation in the presence of an eavesdropper. *IFAC-PapersOnLine*, 50(1):7339–7344, 2017.

[146] D. Levinson and E. Chang. A model for optimizing electronic toll collection systems. *Transportation Research Part A: Policy and Practice*, 37(4):293–314, 2003.

[147] F. Li, B. Luo, and P. Liu. Secure information aggregation for smart grids using homomorphic encryption. In *First International Conference on Smart Grid Communications*, pages 327–332. IEEE, 2010.

[148] J. Li and M. J. Atallah. Secure and private collaborative linear programming. In *Proceedings International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1–8. IEEE, 2006.

[149] Q. Li, G. Cao, and T. F. La Porta. Efficient and privacy-aware data aggregation in mobile sensing. *IEEE Transactions on Dependable and Secure Computing*, 11(2): 115–129, 2014.

[150] Y. Li, L. Shi, P. Cheng, J. Chen, and D. E. Quevedo. Jamming attacks on remote state estimation in cyber-physical systems: A game-theoretic approach. *IEEE Transactions on Automatic Control*, 60(10):2831–2836, 2015.

[151] F. Lin, M. Fardad, and M. R. Jovanović. Augmented Lagrangian approach to design of structured optimal state feedback gains. *IEEE Transactions on Automatic Control*, 56(12):2923–2929, 2011.

[152] Y. Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[153] H. Lipmaa and T. Toft. Secure equality and greater-than tests with sublinear online complexity. In *International Colloquium on Automata, Languages, and Programming*, pages 645–656. Springer, 2013.

[154] L. Ljung. *System Identification: Theory for the User.* Prentice Hall, 1999.

[155] Y. Lu and M. Zhu. Privacy preserving distributed optimization using homomorphic encryption. *Automatica*, 96:314–325, 2018.

[156] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[157] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[158] I. Markovsky and P. Rapisarda. Data-driven simulation and control. *International Journal of Control*, 81(12):1946–1959, 2008.

[159] P. Martins, L. Sousa, and A. Mariano. A survey on fully homomorphic encryption: An engineering perspective. *ACM Computing Surveys (CSUR)*, 50(6):83, 2018.

[160] J. C. Mason and D. C. Handscomb. *Chebyshev polynomials.* CRC press, 2002.

[161] O. Masters, H. Hunt, E. Steffinlongo, J. Crawford, F. Bergamaschi, M. E. D. Rosa, C. C. Quini, C. T. Alves, F. de Souza, and D. G. Ferreira. Towards a homomorphic machine learning big data pipeline for the financial services sector. *IACR Cryptol. ePrint Arch.*, 2019.

[162] D. Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

[163] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[164] L. K. McGovern and E. Feron. Closed-loop stability of systems driven by real-time, dynamic optimization algorithms. In *Proceedings of the 38th Conference on Decision and Control (CDC)*, volume 4, pages 3690–3696. IEEE, 1999.

[165] B. McMahan and D. Ramage. Federated learning: Collaborative machine learning without centralized training data. *Google Research Blog*, 3, 2017.

[166] Microsoft SEAL. `https://github.com/Microsoft/SEAL`, Oct. 2019. Microsoft Research, Redmond, WA.

[167] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[168] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*, pages 61–66, 2010.

[169] C. Mouchet, J. R. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Cryptology ePrint Archive, Report 2020/304*, 2020.

[170] C. Murguia, F. Farokhi, and I. Shames. Secure and private implementation of dynamic controllers using semihomomorphic encryption. *IEEE Transactions on Automatic Control*, 65(9):3950–3957, 2020.

[171] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd workshop on Cloud computing security workshop*, pages 113–124, 2011.

[172] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *12th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457. SIAM, 2001.

[173] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[174] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology–CRYPTO*, pages 681–700. Springer, 2012.

[175] E. Nozari, P. Tallapragada, and J. Cortés. Differentially private average consensus: Obstructions, trade-offs, and optimal algorithm design. *Automatica*, 81:221–231, 2017.

[176] E. Nozari, P. Tallapragada, and J. Cortés. Differentially private distributed convex optimization via functional perturbation. *IEEE Transactions on Control of Network Systems*, 5(1):395–408, 2018.

[177] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[178] PALISADE Lattice Cryptography Library (release 1.10.4). `https://palisade-crypto.org/`, Sept. 2020.

[179] PALISADE Lattice Cryptography Library (release 1.9.2). `https://palisade-crypto.org/`, Apr. 2020.

[180] G. Pannocchia, J. B. Rawlings, and S. J. Wright. Inherently robust suboptimal nonlinear MPC: theory and application. In *Proceedings of the 50th Conference on Decision and Control and European Control*, pages 3398–3403. IEEE, 2011.

[181] F. Pasqualetti, F. Dörfler, and F. Bullo. Attack detection and identification in cyber-physical systems. *IEEE Transactions on Automatic Control*, 58(11):2715–2729, 2013.

[182] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.

[183] P. Patrinos, A. Guiggiani, and A. Bemporad. Fixed-point dual gradient projection for embedded model predictive control. In *European Control Conference (ECC)*, pages 3602–3607. IEEE, 2013.

[184] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.

[185] M. Pettai and P. Laud. Combining differential privacy and secure multiparty computation. In *31st Annual Computer Security Applications Conference*, pages 421–430. ACM, 2015.

[186] M. O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. `https://eprint.iacr.org/2005/187`.

[187] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *ACM SIGMOD International Conference on Management of data*, pages 735–746, 2010.

[188] M. S. Riazi, B. D. Rouhani, and F. Koushanfar. Deep learning on private data. *IEEE Security and Privacy Magazine*, 2018.

[189] J. W. Rittinghouse and J. F. Ransome. *Cloud computing: implementation, management, and security.* CRC press, 2016.

[190] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[191] R. Roman, J. Lopez, and M. Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[192] M. Ruan, H. Gao, and Y. Wang. Secure and privacy-preserving consensus. *IEEE Transactions on Automatic Control*, 2019.

[193] M. Rubagotti, P. Patrinos, A. Guiggiani, and A. Bemporad. Real-time model predictive control based on dual gradient projection: Theory and fixed-point FPGA implementation. *International Journal of Robust and Nonlinear Control*, 26(15):3292–3310, 2016.

[194] A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao. Sharing graphs using differentially private graph models. In *Proceedings of the SIGCOMM conference on Internet measurement*, pages 81–98. ACM, 2011.

[195] S. Salinas, C. Luo, W. Liao, and P. Li. Efficient secure outsourcing of large-scale quadratic programs. In *Proceedings 11th ACM on Asia Conference on Computer and Communications Security*, pages 281–292. ACM, 2016.

[196] J. R. Salvador, D. R. Ramírez, T. Alamo, D. M. de la Peña, and G. García-Marín. Data driven control: an offset free approach. In *18th European Control Conference (ECC)*, pages 23–28. IEEE, 2019.

[197] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux. POSEIDON: Privacy-preserving federated neural network learning. *arXiv preprint arXiv:2009.00349*, 2020.

[198] M. Schulze Darup, A. Redder, I. Shames, F. Farokhi, and D. Quevedo. Towards encrypted mpc for linear constrained systems. *IEEE Control Systems Letters*, 2(2): 195–200, 2018.

[199] M. Schulze Darup, A. Redder, and D. E. Quevedo. Encrypted cooperative control based on structured feedback. *IEEE Control Systems Letters*, 3(1):37–42, 2019.

[200] M. Schulze Darup, A. B. Alexandru, D. E. Quevedo, and G. J. Pappas. Encrypted control for networked systems - an illustrative introduction and current challenges. *IEEE Control Systems*, 2021.

[201] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[202] E. Shi, H. T. H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *Network & Distributed System Security Symposium (NDSS)*. Internet Society., 2011.

[203] Y. Shoukry, K. Gatsis, A. Alanwar, G. J. Pappas, S. A. Seshia, M. Srivastava, and P. Tabuada. Privacy-aware quadratic optimization using partially homomorphic encryption. In *Proceedings of the 55th Conference on Decision and Control (CDC)*, pages 5053–5058. IEEE, 2016.

[204] J. Suh and T. Tanaka. SARSA (0) reinforcement learning over fully homomorphic encryption. *arXiv preprint arXiv:2002.00506*, 2020.

[205] S. Sun, J. Lin, L. Xie, and W. Xiao. Quantized Kalman filtering. In *IEEE 22nd International Symposium on Intelligent Control*, pages 7–12, Singapore, Singapore, 2007. IEEE.

[206] S. Teerapittayanon, B. McDanel, and H.-T. Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *37th International Conference on Distributed Computing Systems*, pages 328–339. IEEE, 2017.

[207] A. Teixeira, K. C. Sou, H. Sandberg, and K. H. Johansson. Secure control systems: A quantitative risk management approach. *IEEE Control Systems Magazine*, 35(1): 24–45, 2015.

[208] K. Tjell and R. Wisniewski. Privacy preserving distributed summation in a connected graph. In *IFAC-PapersOnLine*. Elsevier, 2020.

[209] J. R. Troncoso-Pastoriza and F. Pérez-González. Secure adaptive filtering. *IEEE Transactions on Information Forensics and Security*, 6(2):469–485, 2011.

[210] A. Tsiamis, K. Gatsis, and G. J. Pappas. State estimation codes for perfect secrecy. In *Proceedings of the 56th Conference on Decision and Control (CDC)*, pages 176–181, 2017.

[211] A. Tsiamis, A. B. Alexandru, and G. J. Pappas. Motion planning with secrecy. In *Proceedings of the American Control Conference*, pages 784–791. IEEE, 2019.

[212] S. Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017.

[213] S. Vadhan. Multiparty differential privacy. Differential Privacy Meets Multi-Party Computation (DPMPC) Workshop https://www.bu.edu/hic/dpmpc-2018/, 2018.

[214] J. Vaidya. Privacy-preserving linear programming. In *Proceedings ACM Symposium on Applied Computing*, pages 2002–2007. ACM, 2009.

[215] M. van Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. *HotSec*, 10:1–8, 2010.

[216] P. van Overschee and B. de Moor. *Subspace identification for linear systems: Theory–Implementation–Applications*. Springer Science & Business Media, 2012.

[217] A. W. van Schijndel. *Integrated heat air and moisture modeling and simulation*. PhD thesis, T. U. Eindhoven, 2007.

[218] H. J. van Waarde, C. De Persis, M. K. Camlibel, and P. Tesi. Willems' fundamental lemma for state-space systems and its extension to multiple datasets. *IEEE Control Systems Letters*, 4(3):602–607, 2020.

[219] H. J. van Waarde, J. Eising, H. L. Trentelman, and M. K. Camlibel. Data informativity: a new perspective on data-driven analysis and control. *IEEE Transactions on Automatic Control*, 2020.

[220] G. S. Vernam. Cipher printing telegraph systems: For secret wire and radio telegraphic communications. *Journal of the AIEE*, 45(2):109–115, 1926.

[221] T. Veugen. Improving the DGK comparison protocol. In *International Workshop on Information Forensics and Security*, pages 49–54. IEEE, 2012.

[222] J. Wang, H. Zhong, C. Wu, E. Du, Q. Xia, and C. Kang. Incentivizing distributed energy resource aggregation in energy and capacity markets: An energy sharing scheme and mechanism design. *Applied Energy*, 252:113471, 2019.

[223] Y. Wang, D. J. Hill, and G. Guo. Robust decentralized control for multimachine power systems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 45(3):271–279, 1998.

[224] Y. Wang, Z. Huang, S. Mitra, and G. E. Dullerud. Differential privacy in linear distributed control systems: Entropy minimizing mechanisms and performance tradeoffs. *IEEE Transactions on Control of Network Systems*, 4(1):118–130, 2017.

[225] P. C. Weeraddana, G. Athanasiou, C. Fischione, and J. S. Baras. Per-se privacy preserving solution methods based on optimization. In *Proceedings of the 52nd Conference on Decision and Control (CDC)*, pages 206–211, 2013.

[226] J. C. Willems, P. Rapisarda, I. Markovsky, and B. L. De Moor. A note on persistency of excitation. *Systems & Control Letters*, 54(4):325–329, 2005.

[227] Z. Xu and Q. Zhu. A secure data assimilation for large-scale sensor networks using an untrusted cloud. *IFAC-PapersOnLine*, 50(1):2609–2614, 2017.

[228] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10 (2):1–19, 2019.

[229] S. Yang, S. Tan, and J.-X. Xu. Consensus based approach for economic dispatch problem in a smart grid. *IEEE Transactions on Power Systems*, 28(4):4416–4426, 2013.

[230] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.

[231] Y. Ye, H. Chen, M. Xiao, M. Skoglund, and H. V. Poor. Privacy-preserving incremental ADMM for decentralized consensus optimization. *IEEE Transactions on Signal Processing*, 68:5842–5854, 2020.

[232] C. Yu, L. Ljung, and M. Verhaegen. Identification of structured state-space models. *Automatica*, 90:54–61, 2018.

[233] M. Zellner, T. T. De Rubira, G. Hug, and M. Zeilinger. Distributed differentially private model predictive control for energy storage. *IFAC-PapersOnLine*, 50(1):12464–12470, 2017.

[234] C. Zhang, M. Ahmad, and Y. Wang. ADMM based privacy-preserving decentralized optimization. *IEEE Transactions on Information Forensics and Security*, 14(3):565–580, 2018.

[235] X. Zhang, M. M. Khalili, and M. Liu. Improving the privacy and accuracy of ADMM-based distributed algorithms. In *International Conference on Machine Learning*, pages 5796–5805. PMLR, 2018.

[236] Z. Zhang, Y. Xu, J. Yang, X. Li, and D. Zhang. A survey of sparse representation: algorithms and applications. *IEEE access*, 3:490–530, 2015.

[237] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 724–738. IEEE, 2019.

[238] T. Zhu, G. Li, W. Zhou, and S. Y. Philip. *Differential privacy and applications*. Springer, 2017.