Chapter 9
Secure Multi-party Computation for Cloud-Based Control

Andreea B. Alexandru and George J. Pappas

Abstract In this chapter, we will explore the cloud-outsourced privacy-preserving computation of a controller on encrypted measurements from a (possibly distributed) system, taking into account the challenges introduced by the dynamical nature of the data. The privacy notion used in this work is that of cryptographic multi-party privacy, i.e., the computation of a functionality should not reveal *anything* more than what can be inferred only from the inputs and outputs of the functionality. The main theoretical concept used towards this goal is Homomorphic Encryption, which allows the evaluation of sums and products on encrypted data, and, when combined with other cryptographic techniques, such as Secret Sharing, results in a powerful tool for solving a wide range of secure multi-party problems. We will rigorously define these concepts and discuss how multi-party privacy can be enforced in the implementation of a Model Predictive Controller, which encompasses computing stabilizing control actions by solving an optimization problem on encrypted data.

9.1 Introduction

Cloud computing has become a ubiquitous tool in the age of big data and geographically-spread systems, due to the capabilities of resource pooling, broad network access, rapid elasticity, measured service and on-demand self-service, as defined by NIST [44]. The computational power and storage space of a cloud service can be distributed over multiple servers. Cloud computing has been employed for machine learning applications in e.g., healthcare monitoring and social networks, smart grid control and other control engineering applications, and integration with the Internet of Things paradigm [13, 58]. However, these capabilities do not come without risks. The security and privacy concerns of cloud computing range from communication security to leaking and tampering with the stored data or interfering

A. B. Alexandru (✉) · G. J. Pappas
Department of Electrical Engineering, University of Pennsylvania, Philadelphia, PA, USA
e-mail: aandreea@seas.upenn.edu

G. J. Pappas
e-mail: pappasg@seas.upenn.edu

F. Farokhi (ed.), *Privacy in Dynamical Systems*,
https://doi.org/10.1007/978-981-15-0493-8_9

with the computation, that can be maliciously or unintentionally exploited by the cloud provider or the other tenants of the service [4, 36, 63]. In this chapter, we will focus on concerns related to the privacy of the data and computation. These issues can be addressed by the cryptographic tools described below.

Secure Multi-party Computation (SMPC) encompasses a range of cryptographic techniques that facilitate joint computation over secret data distributed between multiple parties, that 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 [68], where a secure solution to the millionaire's problem was proposed. Surveys on SMPC can be found in [20]. SMPC involves communication between parties and can include individual or hybrid approaches between techniques such as secret sharing [8, 53, 61], oblivious transfer [49, 55], garbled circuits [9, 33, 68], (threshold) homomorphic encryption [30, 48, 59], etc.

Homomorphic Encryption (HE), introduced in [59] as *privacy homomorphisms*, 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 one operation, 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 [29], 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. For a thorough history and description of HE, see the survey [42]. Privacy solutions based on HE were proposed for genome matching, national security and critical infrastructure, healthcare databases, machine learning applications and control systems, etc. [5, 6, 57]. Of particular interest to us are the the works in control applications with HE, see [2, 3, 28, 35, 40, 47, 60], to name a few.

Over the past decades, efforts in optimizing the computation and implementation of SMPC techniques, along with the improvement of network communication speed and more powerful hardware, have opened the way of SMPC deployment in real-world applications. Nevertheless, in the context of big data and the internet of things, which bring enormous amounts of data and large number of devices that want to participate in the computation, SMPC techniques lag behind plaintext computations due to the computational and communication bottleneck [17, 45, 46].

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 some cases [69], a different privacy definition is preferred.

Differential Privacy (DP) refers to methods of concealing the information leakage from the result of the computation, even when having access to auxiliary information [25, 26]. Intuitively, the contribution of the input of each individual on the output should be hidden from those who access the computation results. To achieve this, a carefully chosen noise is added to each entry such that the statistical properties of the database are preserved [27], which introduces a trade-off between utility and privacy. Several works combine SMPC with DP in order to achieve both computation privacy and output privacy, for instance [16, 54, 56, 62, 64].

In this chapter, we focus on proposing SMPC schemes for optimization and control problems, where we require the convergence and accuracy of the results. DP techniques can be further investigated in order to ensure output privacy.

9.1.1 Dynamical Data Challenges

Cryptographic techniques were developed mainly for static data, such as databases, or independent messages. 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/times step 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 has to be repeated for a continuously running process. In this chapter, we design the solution with all these issues in mind.

9.1.2 Contribution and Roadmap

Examples of control applications that require privacy include smart metering, crucial infrastructure control, prevention of industrial espionage, swarms of robots deployed in adversarial environments, etc. In order to ensure the privacy of a control application, i.e., secure both the signals and the model, as well as the intermediate computations, most of the time, complex solutions have to be devised. 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. The purpose of this chapter is to describe these cryptographic techniques and show how to combine them in order to build private cloud-based protocols that produce the desired control actions.

The layout of the chapter is as follows: in Sect. 9.2, we describe the model predictive control problem that we address, we show how to compute the control action when there are no privacy requirements and we outline the privacy-preserving solution. In Sect. 9.3, we formally describe the adversarial model and privacy definition we seek for our multi-party computation problem. In Sect. 9.4, we introduce the cryptographic tools that we will use in our solution and their properties. Then, in Sect. 9.5, we design the multi-party protocol for the model predictive control problem with encrypted states and model and prove its privacy. Finally, we discuss some details about the requirements of the proposed protocol in Sect. 9.6.

9.1.3 Notation

We use bold-face lower case for vectors, e.g., $\mathbf{x}$, and bold-face upper case for matrices, e.g., $\mathbf{A}$. $\mathbb{N}$ denotes the set of non-negative integers, $\mathbb{Z}$ 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. For $n \in \mathbb{N}$, let $[n] := \{1, 2, \dots, n\}$. $\mathrm{E}(x)$ and $[[x]]$ represent encryptions of the scalar value x. We use the same notation for multidimensional objects: an encryption of a matrix $\mathbf{A}$ is denoted by $\mathrm{E}(\mathbf{A})$ (or $[[\mathbf{A}]]$) and is performed element-wise. $\{0, 1\}^*$ defines a sequence of bits of unspecified length.

9.2 Model Predictive Control

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

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{x}(t) = \left[\mathbf{x}^1(t)^\intercal \dots \mathbf{x}^M(t)^\intercal\right]&, \quad \mathbf{u}(t) = \left[\mathbf{u}^1(t)^\intercal \dots \mathbf{u}^M(t)^\intercal\right], \end{aligned} \tag{9.1}$$

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 system can be either centralized or partitioned in subsytems for $i \in [M]$, $\mathbf{x}^i(t) \in \mathbb{R}^{n_i}$, $\sum_{i=1}^M n_i = n$ and $\mathbf{u}^i(t) \in \mathbb{R}^{m_i}$, $\sum_{i=1}^M m_i = m$.

The Model Predictive Control (MPC) is the optimal control receding horizon problem with constraints written as:

$$\begin{aligned} J_N^*(\mathbf{x}(t)) = \min_{\mathbf{u}_0,\dots,\mathbf{u}_{N-1}} &\frac{1}{2}\left(\mathbf{x}_N^\intercal \mathbf{P}\mathbf{x}_N + \sum_{k=0}^{N-1} \mathbf{x}_k^\intercal \mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\intercal \mathbf{R}\mathbf{u}_k\right) \\ s.t.\ &\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k,\ k = 0, \dots, N-1;\quad \mathbf{x}_0 = \mathbf{x}(t); \\ &\mathbf{u}_k \in \mathcal{U},\ k = 0, \dots, N-1, \end{aligned} \tag{9.2}$$

where N is the length of the horizon and $\mathbf{P}, \mathbf{Q}, \mathbf{R} \succ 0$ are cost matrices. We consider input constrained systems with box constraints $\mathbf{0} \in \mathcal{U} = \{\mathbf{l}_u \preceq \mathbf{u} \preceq \mathbf{h}_u\}$, and impose stability without a terminal state constraint, but with appropriately chosen costs and horizon, 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 [43].

Through straightforward manipulations, (9.2) can be written as the quadratic problem (9.3)—see details on obtaining the matrices $\mathbf{H}$ and $\mathbf{F}$ in [11, Chaps. 8, 11]—in the variable $\mathbf{U} := \begin{bmatrix}\mathbf{u}_0^\intercal & \mathbf{u}_1^\intercal & \dots & \mathbf{u}_{N-1}^\intercal\end{bmatrix}^\intercal$. For simplicity, we keep the same notation $\mathcal{U}$ for the augmented constraint set. After obtaining the optimal solution, the first m components of $\mathbf{U}^*(\mathbf{x}(t))$ are applied as input to the system (9.1): $\mathbf{u}^*(\mathbf{x}(t)) = \{\mathbf{U}^*(\mathbf{x}(t))\}_{1:m}$.

$$\mathbf{U}^*(\mathbf{x}(t)) = \arg\min_{\mathbf{U}\in\mathcal{U}} \frac{1}{2}\mathbf{U}^\intercal\mathbf{H}\mathbf{U} + \mathbf{U}^\intercal\mathbf{F}^\intercal\mathbf{x}(t). \tag{9.3}$$

9.2.1 Solution Without Privacy Requirements

The privacy-absent cloud-MPC problem is depicted in Fig. 9.1. The system (9.1) 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 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.

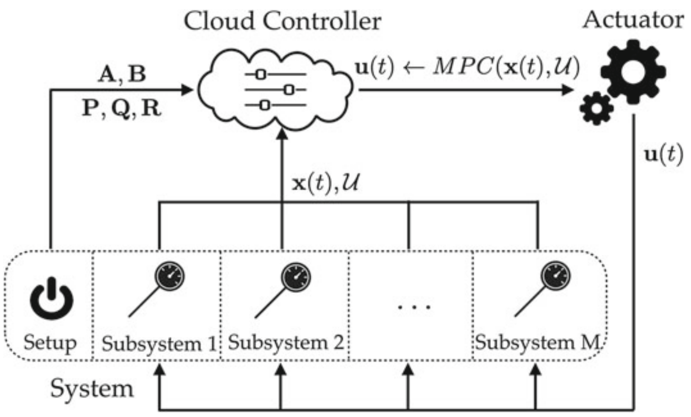

Fig. 9.1 Cloud-based MPC problem for a system composed of a number of subsystems and a setup entity. The control action is computed by a cloud controller and sent to a virtual actuator

The constraint set $\mathcal{U}$ is a hyperbox, so the projection step required for solving (9.3) 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) [50], given in (9.4):

$$\text{For } k = 0\ldots, K-1$$

$$\mathbf{t}_k \leftarrow \left(\mathbf{I}_{Mm} - \frac{1}{L}\mathbf{H}\right)\mathbf{z}_k - \frac{1}{L}\mathbf{F}^\intercal\mathbf{x}(t) \tag{9.4a}$$

$$\mathbf{U}_{k+1} \leftarrow \Pi_{\mathcal{U}}(\mathbf{t}_k) \tag{9.4b}$$

$$\mathbf{z}_{k+1} \leftarrow (1+\eta)\mathbf{U}_{k+1} - \eta\mathbf{U}_k \tag{9.4c}$$

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 $\mathbf{H}$.

9.2.2 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 (9.3) 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 (9.4) for the model predictive control problem, such that the actuator obtains the control action $\mathbf{u}^*(t)$ for system (9.1), without learning anything else in the process. 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 formally introduce the adversarial model and multi-party privacy definition in Sect. 9.3.

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. The labeled homomorphic encryption builds on top of an *additively homomorphic encryption* scheme, which allows only the evaluation of additions over encrypted data, 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 will be described in detail in Sect. 9.4, and our private cloud-based MPC solution that incorporates them will be presented in Sect. 9.5.

9.3 Adversarial Model and Privacy Definition

In cloud applications, the service provider has to deliver the contracted service that was agreed upon, otherwise the clients switch to another service provider. The clients' interest is to obtain the correct result for the service they pay for, so we may assume they do not alter the data sent to the cloud. However, the parties can locally process copies of the data they receive in any fashion they want. This adversarial model is known as semi-honest, which is defined formally as follows:

Definition 9.1 (*Semi-honest model* [32, Chap. 7]) 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 its internal coin tosses, as well as 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. Malicious and active adversaries—that diverge from the protocols or tamper with the messages—are not considered in this chapter. Privacy against malicious adversaries can be obtained by introducing zero-knowledge proofs and commitment schemes, at the cost of a computational overhead [23, 32].

We introduce some concepts necessary for the multi-party privacy definitions. An ensemble $X = \{X_\sigma\}_{\sigma\in\mathbb{N}}$ is a sequence of random variables ranging over strings of bits of length polynomial in σ, arising from distributions defined over a finite set Ω.

Definition 9.2 (*Statistical indistinguishability* [31, Chap. 3]) The ensembles $X = \{X_\sigma\}_{\sigma\in\mathbb{N}}$ and $Y = \{Y_\sigma\}_{\sigma\in\mathbb{N}}$ are **statistically indistinguishable**, denoted $\overset{s}{\equiv}$, if for every positive polynomial p, and all sufficiently large σ:

$$\mathrm{SD}[X_\sigma, Y_\sigma] := \frac{1}{2}\sum_{\alpha\in\Omega}\left|\Pr[X_\sigma = \alpha] - \Pr[Y_\sigma = \alpha]\right| < \frac{1}{p(\sigma)}.$$

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

It can be proved that two ensembles are statistically indistinguishable if no algorithm can distinguish between them. Statistical indistinguishability holds against computationally unbounded adversaries. Computational indistinguishability is a weaker notion of the statistical version, as follows:

Definition 9.3 (*Computational Indistinguishability* [31, Chap. 3]) The ensembles $X = \{X_\sigma\}_{\sigma\in\mathbb{N}}$ and $Y = \{Y_\sigma\}_{\sigma\in\mathbb{N}}$ are **computationally indistinguishable**, denoted $\overset{c}{\equiv}$, if for every probabilistic polynomial-time algorithm $D : \{0,1\}^* \to \{0,1\}$, called the distinguisher, every positive polynomial p, and all sufficiently large σ:

$$\left| \Pr_{x\leftarrow X_\sigma}[D(x)=1] - \Pr_{y\leftarrow Y_\sigma}[D(y)=1]\right| < \frac{1}{p(\sigma)}.$$

Let us now look at the privacy definition that is considered in secure multi-party computation, that makes use of the real and ideal world paradigms. Consider a multi-party protocol Π that executes a functionality $f = (f_1,\ldots,f_p)$ on inputs $I = (I_1,\ldots,I_p)$ and produces an output $f(I) = (f_1(I),\ldots,f_p(I))$ in the following way: the parties have their inputs, then exchange messages between themselves in order to obtain an output. If an adversary corrupts a party (or a set of parties) in this real world, after the execution of the protocol, it will have access to its (their) input, the messages received and its (their) output. In an ideal world, there is a trusted incorruptible party that takes the inputs from all the parties, computes the functionality on them, and then sends to each party its output. In this ideal world, an adversary that corrupts a party (or a set of parties) will only have access to its (their) input and output. This concept is illustrated for three parties in Fig. 9.2. We then say that a multi-party protocol achieves computational privacy if, for each party, what a computationally bounded adversary finds out from the real world execution is equivalent to what it finds out from the ideal-world execution.

The formal definition is the following:

Definition 9.4 (*Multi-party privacy w.r.t. semi-honest behavior* [32, Chap. 7]) Let $f : (\{0,1\}^*)^p \to (\{0,1\}^*)^p$ be a p-ary functionality, where $f_i(x_1,\ldots,x_p)$ denotes the i-th element of $f(x_1,\ldots,x_p)$. Denote the inputs by $\bar{x} = (x_1,\ldots,x_p)$. For $I = \{i_1,\ldots,i_t\} \subset [p]$, 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 p-party protocol that computes f. The **view** of the i-th party during an execution of Π on the inputs $\bar{x}$, denoted $V_i^\Pi(\bar{x})$, is $(x_i, \text{coins}, m_1,\ldots,m_t)$, where coins represents the outcome of the

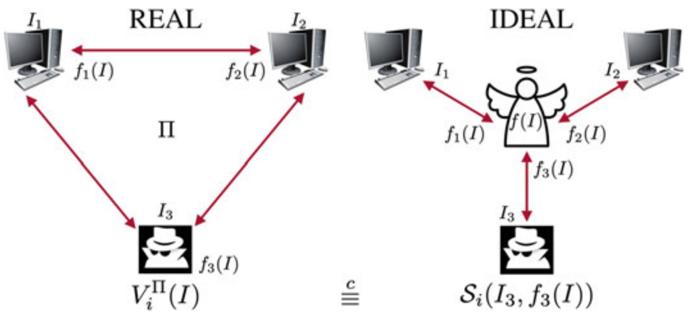

Fig. 9.2 Real and ideal paradigms for secure multi-party computation

i'th party's internal coin tosses, and m_j represents the j-th message it has received. We let the view of a coalition be denoted 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 $\boldsymbol{\Pi}$ **privately computes** $\boldsymbol{f}$ if there exist simulators S, such that, for every $I \subset [p]$, it holds that, for $\bar{x}_t = (x_{i_1}, \ldots, x_{i_t})$:

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

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 [32, 41]. Auxiliary inputs, which are inputs that capture additional information available to each of the parties, (e.g., local configurations, side-information), are implicit in this definition [32, Chap. 7], [31, Chap. 4].

Definition 9.4 states that the view of any of the parties participating in the protocol, on each possible set of inputs, can be simulated based only on its own input and output. For parties that have no assigned input and output, like the cloud server in our problem, Definition 9.4 captures the strongest desired privacy model.

9.4 Cryptographic Tools

9.4.1 Secret Sharing

Secret sharing [53, 61] is a tool that distributes a private message to a number of parties, by splitting it into random shares. Then, the private message can be reconstructed only by an authorized subset of parties, which combine their shares.

One common and simple scheme is the additive 2-out-of-2 secret sharing scheme, which involves a party splitting its secret message $m \in G$, where G is a finite abelian group, into two shares, in the following way: generate uniformly at random an element $\mathfrak{b} \in G$, subtract it from the message and then distribute the shares $\mathfrak{b}$ and $m - \mathfrak{b}$. This can be also thought of as a one-time pad [10, 65] variant on G. Both shares are needed in order to recover the secret. The 2-out-of-2 secret sharing scheme achieves perfect secrecy, which means that the shares of two distinct messages are uniformly distributed on G [19].

We will also use an additive blinding scheme weaker than secret sharing (necessary for the private comparison protocol in Sect. 9.4.5): for messages of l bits, $\mathfrak{b}$ will be generated from a message space $\mathcal{M}$ wit h length of $\lambda + l$ bits, where λ is the security parameter, with the requirement that $\lambda + l$-bit messages can still be represented in $\mathcal{M}$, i.e. there is no wrap-around and overflow. The distribution of $m + \mathfrak{b}$ is statistically indistinguishable from a random number sampled of $l + \lambda + 1$ bits. Such a scheme is commonly employed for blinding messages, for instance in [12, 38, 66].

9.4.2 Pseudorandom Generators

Pseudorandom generators are efficient deterministic functions that expand short seeds into longer pseudorandom bit sequences, that are computationally indistinguishable from truly random sequences. More details can be found in [31, Chap. 3].

9.4.3 Homomorphic Encryption

Let $\mathrm{E}(\cdot)$ denote a generic encryption primitive, with domain the space of private data, called **plaintexts**, and codomain the space of encrypted data, called **ciphertexts**. $\mathrm{E}(\cdot)$ also takes as input the public key, and probabilistic encryption primitives also take a random number. The decryption primitive $\mathrm{D}(\cdot)$ is defined on the space of ciphertexts and takes values on the space of plaintexts. $\mathrm{D}(\cdot)$ also takes as input the private key. **Additively homomorphic** schemes satisfy the property that there exists an operator $\oplus$ defined on the space of ciphertexts such that:

$$\mathrm{E}(a) \oplus \mathrm{E}(b) \subset \mathrm{E}(a+b), \tag{9.5}$$

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, Eq. (9.5) 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 $\mathrm{D}(\cdot)$ 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.

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

$$\mathrm{E}(a) \otimes \mathrm{E}(b) \subset \mathrm{E}(a \cdot b), \tag{9.6}$$

for any plaintexts a, b supported by the scheme. If the same scheme satisfies both (9.5) and (9.6) for an unlimited amount of operations, it is called **fully homomorphic**. If a scheme satisfies both (9.5) and (9.6) but only for a limited amount of operations, it is called **somewhat homomorphic**.

Remark 9.1 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.

9.4.3.1 Additively Homomorphic Cryptosystem

Additively Homomorphic Encryptions schemes, abbreviated as AHE, can be instantiated by various public key additively homomorphic encryption schemes such as [24, 34, 39, 52]. Let AHE = ($\widehat{\text{KeyGen}}$, $\hat{\text{E}}$, $\hat{\text{D}}$, $\widehat{\text{Add}}$, $\widehat{\text{cMlt}}$) be an instance of an asymmetric additively homomorphic encryption scheme, with $\mathcal{M}$ the message space and $\hat{\mathcal{C}}$ the ciphertext space, where we will use the following abstract notation: $\hat{\oplus}$ denotes the addition on $\hat{\mathcal{C}}$ and $\hat{\otimes}$ denotes the multiplication between a plaintext and a ciphertext. We save the notation without $\hat{(\cdot)}$ for the scheme in Sect. 9.4.3.2. 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. We will denote the encryption of a message $m \in \mathcal{M}$ by $[[m]]$ as a shorthand notation for $\hat{\text{E}}(\text{public key}, m)$.

(i) $\widehat{\text{KeyGen}}(1^\sigma)$: Takes the security parameter σ and outputs a public key $\hat{\text{pk}}$ and a private key $\hat{\text{sk}}$.
(ii) $\hat{\text{E}}(\hat{\text{pk}}, m)$: Takes the public key and a message $m \in \mathcal{M}$ and outputs a ciphertext $[[m]] \in \hat{\mathcal{C}}$.
(iii) $\hat{\text{D}}(\hat{\text{sk}}, c)$: Takes the private key and a ciphertext $c \in \hat{\mathcal{C}}$ and outputs the message that was encrypted $m' \in \mathcal{M}$.
(iv) $\widehat{\text{Add}}(c_1, c_2)$: Takes ciphertexts $c_1, c_2 \in \hat{\mathcal{C}}$ and outputs ciphertext $c = c_1 \hat{\oplus} c_2 \in \hat{\mathcal{C}}$ such that: $\hat{\text{D}}(\hat{\text{sk}}, c) = \hat{\text{D}}(\hat{\text{sk}}, c_1) + \hat{\text{D}}(\hat{\text{sk}}, c_2)$.
(v) $\widehat{\text{cMlt}}(m_1, c_2)$: Takes plaintext $m_1 \in \mathcal{M}$ and ciphertext $c_2 \in \hat{\mathcal{C}}$ and outputs ciphertext $c = m_1 \hat{\otimes} c_2 \in \hat{\mathcal{C}}$ such that: $\hat{\text{D}}(\hat{\text{sk}}, c) = m_1 \cdot \hat{\text{D}}(\hat{\text{sk}}, c_2)$.

In this chapter, we use the popular Paillier encryption [52], that has plaintext space $\mathbb{Z}_N$ and ciphertext space $(\mathbb{Z}_{N^2})^*$. Any other AHE scheme that is semantically secure [34], [32, Chap. 4] and circuit-private [14] can be employed.

9.4.3.2 Labeled Homomorphic Encryption

The model predictive control problem from Fig. 9.1, and more general control decision problems as well, can be abstracted in the following general framework in Fig. 9.3. Consider a cloud server that collects encrypted data from several clients. The data represents time series and is labeled with the corresponding time. A requester makes queries that can be written as multivariate polynomials over the data stored at the cloud server and solicits the result.

Labeled Homomorphic Encryption (LabHE) can process data from multiple users with different private keys, as long as the requesting party has a master key. This

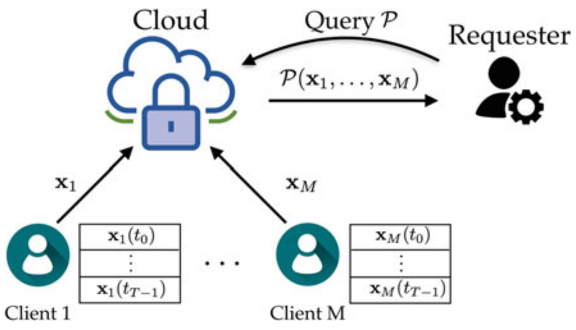

Fig. 9.3 The clients send their private data to a cloud server. A requester sends a query to the cloud, which evaluates it on the data and sends the result to the requester

scheme makes use of the fact that the decryptor (or requester in Fig. 9.3) 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 (or client in Fig. 9.3) 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. 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 [7].

Definition 9.5 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. Given t programs $\mathcal{P}_1, \dots, \mathcal{P}_t$ 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 $\mathcal{P}_1, \dots, \mathcal{P}_t$, and can be denoted compactly as $\mathcal{P}^g = g(\mathcal{P}_1, \dots, \mathcal{P}_t)$. The labeled inputs of $\mathcal{P}^g$ are all the distinct labeled inputs of $\mathcal{P}_1, \dots, \mathcal{P}_t$.

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 $\mathcal{P} = (f, \tau_1, \dots, \tau_n)$, as in Definition 9.5, can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \dots, \mathcal{I}_{\tau_n})$.

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. The idea is that an encryptor splits their private message as described in Sect. 9.4.1 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 generator applied to the corresponding label. The label acts like the seed of the pseudo-random generator. The encryptor then forms the LabHE ciphertext from the encryption of the first share along with the second share, yielding $\mathrm{E}(m) = (m - \mathfrak{b}, [[\mathfrak{b}]])$, as described in Step 1 in the following. This enables us to decrypt one multiplication of two encrypted values, using the observation (9.7). The AHE scheme allows computing $(m_1 - \mathfrak{b}_1)\hat{\otimes}[[\mathfrak{b}_2]]$, $(m_2 - \mathfrak{b}_2)\hat{\otimes}[[\mathfrak{b}_1]]$ and $[[(m_1 - \mathfrak{b}_1) \cdot (m_2 - \mathfrak{b}_2)]]$, for plaintexts $(m_i - \mathfrak{b}_i)$, $i = 1, 2$. Hence, we can obtain the AHE encryption of one multiplication $[[m_1 \cdot m_2 - \mathfrak{b}_1 \cdot \mathfrak{b}_2]]$ from $\mathrm{E}(m_1)$ and $\mathrm{E}(m_1)$, 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 $\mathfrak{b}_i$, such that it can compute $m_1 \cdot m_2 = \hat{\mathrm{D}}[[m_1 \cdot m_2 - \mathfrak{b}_1 \cdot \mathfrak{b}_2]] + \mathfrak{b}_1 \cdot \mathfrak{b}_2$.

$$m_1 \cdot m_2 - \mathfrak{b}_1 \cdot \mathfrak{b}_2 = (m_1 - \mathfrak{b}_1) \cdot (m_2 - \mathfrak{b}_2) + \mathfrak{b}_1 \cdot (m_2 - \mathfrak{b}_2) + \mathfrak{b}_2 \cdot (m_1 - \mathfrak{b}_1). \tag{9.7}$$

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 $\mathcal{L}$. Then LabHE is defined as a tuple LabHE = (Init, KeyGen, E, Eval, D):

1. $\mathrm{Init}(1^\sigma)$: 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. $\mathrm{E}(\mathrm{mpk}, \mathrm{upk}, \tau, m)$: 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-E(usk, τ): Computes the secret $\mathfrak{b} \leftarrow F(usk, \tau)$ and outputs $\mathrm{C_{off}} = (\mathfrak{b}, [[\mathfrak{b}]])$.
   - $\mathrm{On\text{-}E}(\mathrm{C_{off}}, m)$: Outputs $C = (m - \mathfrak{b}, [[\mathfrak{b}]]) =: (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$.

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

   - $\mathrm{Mlt}(C_1, C_2)$: Takes $C_i = (a_i, \beta_i) \in \mathcal{M} \times \hat{\mathcal{C}}$ for $i = 1, 2$ and outputs $C = [[a_1 \cdot a_2]]\hat{\oplus}(a_1\hat{\otimes}\beta_2)\hat{\oplus}(a_2\hat{\otimes}\beta_1) = [[m_1 \cdot m_2 - \mathfrak{b}_1 \cdot \mathfrak{b}_2]] =: \alpha \in \hat{\mathcal{C}}$.
   - $\mathrm{Add}(C_1, C_2)$: If $C_i = (a_i, \beta_i) \in \mathcal{M} \times \hat{\mathcal{C}}$ for $i = 1, 2$, then outputs $C = (a_1 + a_2, \beta_1\hat{\oplus}\beta_2) =: (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$. If both $C_i = \alpha_i \in \hat{\mathcal{C}}$, for $i = 1, 2$, then outputs $C = \alpha_1\hat{\oplus}\alpha_2 =: \alpha \in \hat{\mathcal{C}}$. If $C_1 = (a_1, \beta_1) \in \mathcal{M} \times \hat{\mathcal{C}}$ and $C_2 = \alpha_2 \in \hat{\mathcal{C}}$, then outputs $C = (a_1, \beta_1\hat{\oplus}\alpha_2) =: (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$.
   - $\mathrm{cMlt}(c, C')$: Takes a plaintext $c \in \mathcal{M}$ and a ciphertext C'. If $C' = (a', \beta') \in \mathcal{M} \times \hat{\mathcal{C}}$, outputs $C = (c \cdot a', c\hat{\otimes}\beta') =: (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$. If $C' = \alpha' \in \hat{\mathcal{C}}$, outputs $C = c\hat{\otimes}\alpha' =: \alpha \in \hat{\mathcal{C}}$.

5. $\mathrm{D}(\mathrm{msk}, \mathrm{usk}_{1,\ldots,t}, \mathcal{P}, C)$: 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:

   - $\mathrm{Off\text{-}D}(\mathrm{msk}, \mathcal{P})$: Parses $\mathcal{P}$ as $(f, \tau_1, \ldots, \tau_t)$. For $i \in [t]$, it computes the secrets $\mathfrak{b}_i = F(\mathrm{usk}_i, \tau_i)$, $\mathfrak{b} = f(\mathfrak{b}_1, \ldots, \mathfrak{b}_t)$ and outputs $\mathrm{msk}_{\mathcal{P}}(\mathrm{msk}, \mathfrak{b})$.
   - $\mathrm{On\text{-}D}(\mathrm{sk}_{\mathcal{P}}, C)$: If $C = (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$: either output (i) $m = a + \mathfrak{b}$ or (ii) output $m = a + \hat{\mathrm{D}}(\mathrm{msk}, \beta)$. If $C \in \hat{\mathcal{C}}$, output $m = \hat{\mathrm{D}}(\mathrm{msk}, C) + \mathfrak{b}$.

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 $\hat{\text{D}}$.

Semantic security characterizes the security of a cryptosystem. The definition of semantic security is sometimes given as a cryptographic game [34].

Definition 9.6 (*Semantic Security* [32, Chap. 5]) An encryption scheme is **semantically secure** if for every probabilistic polynomial-time algorithm, $\mathcal{A}$, there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ such that for every two polynomially bounded functions $f, h : \{0,1\}^* \to \{0,1\}^*$, for any probability ensemble $\{X_\sigma\}_{\sigma \in \mathbb{N}}$ of length polynomial in σ, for any positive polynomial p and sufficiently large σ:

$$\Pr\left[\mathcal{A}(\text{E}(X_\sigma), h(X_\sigma), 1^\sigma) = f(X_\sigma)\right] < \Pr\left[\mathcal{A}'(h(X_\sigma), 1^\sigma) = f(X_\sigma)\right] + \frac{1}{p(\sigma)},$$

The probability is taken over the ensemble X_σ and the internal coin tosses in $\mathcal{A}$, $\mathcal{A}'$.

Under the assumption of decisional composite residuosity [52], the Paillier scheme is semantically secure and has indistinguishable encryptions. Moreover, the LabHE scheme satisfies semantic security given that the underlying homomorphic encryption scheme is semantically secure and the function F is pseudorandom.

In [7] it is proved that LabHE also satisfies 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).

9.4.4 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 [32, Chap. 7]. Party A has k secrets $(\sigma_0, \dots, \sigma_{k-1})$ and party B has an index $i \in \{0, \dots, k-1\}$. 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 9.4). Many improvements in efficiency, e.g., precomputation, and security have been proposed, see e.g., [37, 51].

We will use the standard 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 $\sigma_i \leftarrow \text{OT}([[\sigma_0]], [[\sigma_1]], i, \hat{\text{sk}})$. 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 $[[\sigma_i]] \leftarrow \text{OT}'([[\sigma_0]], [[\sigma_1]], i, \hat{\text{sk}})$.

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: $[[v_0]] := \widehat{\text{Add}}([[\sigma_0]], [[r_0]])$, $[[v_1]] := \widehat{\text{Add}}([[\sigma_1]], [[r_1]])$. B selects v_i and sends back to A the encryption of the index i: $[[i]]$ and $\hat{\text{Add}}([[v_i]], [0])$, such that A cannot obtain information about i by comparing the value it received with the values it sent. Then, A computes:

$$[[\sigma_i]] = \hat{\text{Add}}\left([[v_i]], \text{c}\hat{\text{Mlt}}(r_0, \hat{\text{Add}}([[i]], [[-1]])), \text{c}\hat{\text{Mlt}}(-r_1, [[i]])\right).$$

Proposition 9.1 $[[\sigma_i]] \leftarrow \text{OT}'([[\sigma_0]], [[\sigma_1]], i, \hat{\text{sk}})$ *is private w.r.t. Definition 9.4.*

We will show in the following proof how to construct the simulators in Definition 9.4 in order to prove the privacy of the oblivious transfer variant we use.

Proof Let us construct the view of A, with inputs $\hat{\text{pk}}$, $[[\sigma_0]]$, $[[\sigma_1]]$ and output $[[\sigma_i]]$:

$$V_A(\hat{\text{pk}}, [[\sigma_0]], [[\sigma_1]]) = \left(\hat{\text{pk}}, [[\sigma_0]], [[\sigma_1]], r_0, r_1, [[i]], [[v_i]], [[\sigma_i]], \text{coins}\right),$$

where coins are the random values used for encrypting r_0 and r_1 and $[[-1]]$. The view of party B, that has inputs $i, \hat{\text{pk}}, \hat{\text{sk}}$ and no output, is:

$$V_B(i, \hat{\text{pk}}, \hat{\text{sk}}) = \left(i, \hat{\text{pk}}, \hat{\text{sk}}, [[v_0]], [[v_1]], \text{coins}\right),$$

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 $\hat{\text{pk}}, [[\sigma_0]], [[\sigma_1]], [[\sigma_i]]$ 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 $\hat{\text{pk}}$ and computes $[[\widetilde{v_i}]] = \hat{\text{Add}}\left([[\sigma_i]], \text{c}\hat{\text{Mlt}}(-\widetilde{r_0}, \hat{\text{Add}}([[\widetilde{i}]], [[-1]])), \text{c}\hat{\text{Mlt}}(\widetilde{r_1}, [[\widetilde{i}]])\right)$. It also generates $\widetilde{\text{coins}}$ for three encryptions. S_A outputs:

$$S_A(\hat{\text{pk}}, [[\sigma_0]], [[\sigma_1]], [[\sigma_i]]) = \left(\hat{\text{pk}}, [[\sigma_0]], [[\sigma_1]], \widetilde{r_0}, \widetilde{r_1}, [[\widetilde{i}]], [[\widetilde{v_i}]], [[\sigma_i]], \widetilde{\text{coins}}\right).$$

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, $V_A(\hat{\text{pk}}, [[\sigma_0]], [[\sigma_1]]) \overset{c}{\equiv} S_A(\hat{\text{pk}}, [[\sigma_0]], [[\sigma_1]], [[\sigma_i]])$.

A simulator S_B for party B takes as inputs $i, \hat{\text{pk}}, \hat{\text{sk}}$ 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:

$$S_B(i, \hat{\text{pk}}, \hat{\text{sk}}) = \left(i, \hat{\text{pk}}, \hat{\text{sk}}, [[\widetilde{v_0}]], [[\widetilde{v_1}]], \widetilde{\text{coins}}\right).$$

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, $V_B(i, \hat{\text{pk}}, \hat{\text{sk}}) \overset{c}{\equiv} S_B(i, \hat{\text{pk}}, \hat{\text{sk}})$. □

9.4.5 *Private Comparison*

Consider a two-party computation problem of two inputs encrypted under an encryption scheme that does not preserve the order from plaintexts to ciphertexts. A survey on the state of the art of private comparison protocols on private inputs owned by two parties is given in [18]. In [21, 22], Damgård, Geisler and Krøigaard describe a protocol for secure comparison and, towards that functionality, they propose the DGK additively homomorphic encryption scheme with the property that, given a ciphertext and the secret key, it is efficient to determine if the encrypted value is zero without fully decrypting it. This is useful when making decisions on bits. The authors also prove the semantic security of the DGK cryptosystem under the hardness of factoring assumption. We denote the DGK encryption of a scalar value by $[\cdot]$ and use the same operations notation $\hat{\oplus}$ and $\hat{\otimes}$.

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 1, where $\veebar$ denotes the exclusive or operation. The protocol is described in [67, Protocol 3], where an improvement of the DGK scheme is proposed. By applying some extra steps, as in [67, 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.

PROTOCOL 1: Private two-party comparison with plaintext inputs using DGK

```
Require: A: $\alpha$; B: $\beta$, $sk_{DGK}$
Ensure: 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 $[\alpha_i \veebar \beta_i] \leftarrow [1]\hat{\oplus}(-1)\hat{\otimes}[\beta_i]$ 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 $[c_i] \leftarrow [\alpha_{i+1} \veebar \beta_{i+1}]\hat{\oplus} \dots \hat{\oplus}[\alpha_l \veebar \beta_l])$ .
 9:     A: $[c_i] \leftarrow [1]\hat{\oplus}[c_i] \oplus (-1)\hat{\otimes}[\beta_i]$ if $\delta_A = 0$ and $[c_i] \leftarrow [1]\hat{\oplus}[c_i]$ otherwise.
10: end for
11: A: generate uniformly random non-zero values $r_i$ of $2t$ bits (see [22]), $0 \leq i < l$.
12: for each $0 \leq i < l$ do
13:     A: $[c_i] \leftarrow r_i\hat{\otimes}[c_i]$ 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: Private two-party comparison with encrypted inputs using DGK

```
Require: A: $[[a]], [[b]]$; B: $\hat{\mathrm{sk}}, sk_{DGK}$
Ensure: B: $(\delta = 1) \equiv (a \leq b)$
1: A: choose uniformly at random $r$ of $l + 1 + \lambda$ bits, compute $[[z]] \leftarrow [[b]]\hat{\ominus}[[a]]\hat{\oplus}[[2^l + r]]$
   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 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 $[[(\beta < \alpha)]] \leftarrow [[1]]\hat{\ominus}[[\delta_B]]$ otherwise.
6: A: compute $[[\delta]] \leftarrow [[z \div 2^l]]\hat{\ominus}([[r \div 2^l]]\hat{\oplus}[[(\beta < \alpha)]])$ and send it to B.
7: B: decrypts $\delta$.
```

Proposition 9.2 *([21, 67]) Protocol 2 is private w.r.t. Definition 9.4.*

9.5 MPC with Encrypted Model and Encrypted Signals

As remarked 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 MPC on encrypted data. LabHE has a useful property called unbalanced efficiency that can be observed from Sect. 9.4.3.2 and was described in [15], which states that only the evaluator 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.

Our protocols will 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.

Figure 9.4 represents the private version of the MPC diagram from Fig. 9.1, where the quantities will be briefly described next and more in detail in Sect. 9.5.1. The actuator holds the MPC query functionality, denoted by f_{MPC}. Offline, the actuator generates a pair of master keys, as described in Sect. 9.4.3.2 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. As explained in Sect. 9.4.3.2, 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

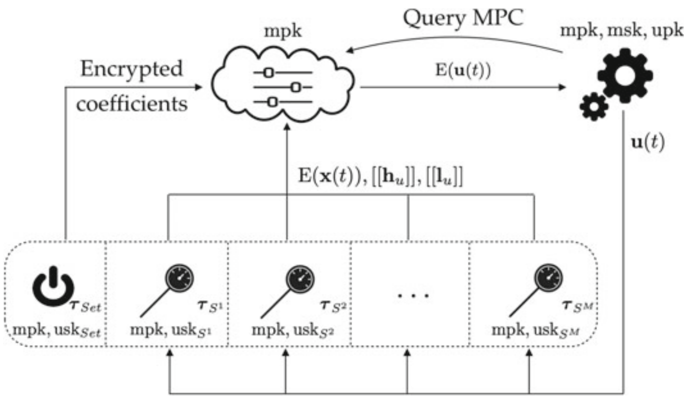

Fig. 9.4 The setup and subsystems send their encrypted data to the cloud. The cloud has to run the MPC algorithm on the private measurements and the system's private matrices and send the encrypted result to the actuator. The latter then actuates the system with the decrypted inputs

steps and sizes corresponding to each signal, with no other complex synchronization process necessary between the actors. This is shown in Protocol 3.

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 4, we describe how the encrypted MPC query is performed by the parties.

We will now show how to transform the FGM (9.4) into a private version. The message space $\mathcal{M}$ we choose for the encryption schemes 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. For now, assume that this preprocessing step has been already performed. We postpone the details to Sect. 9.5.3.

First, let us write $\mathbf{t}_k$ in (9.4a) as a function of $\mathbf{U}_k$ and $\mathbf{U}_{k-1}$:

$$\begin{aligned}
\mathbf{t}_k &= \left(\mathbf{I}_{Mm} - \frac{1}{L}\mathbf{H}\right) z_k - \frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t) \\
&= \left(\mathbf{I}_{Mm} - \frac{1}{L}\mathbf{H}\right) \left[(1+\eta)\mathbf{U}_k - \eta \mathbf{U}_{k-1}\right] - \frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t) \\
&= \mathbf{U}_k + \eta(\mathbf{U}_k - \mathbf{U}_{k-1}) - \frac{1}{L}\mathbf{H}\mathbf{U}_k - \frac{\eta}{L}\mathbf{H}(\mathbf{U}_k - \mathbf{U}_{k-1}) - \frac{1}{L}\mathbf{F}^\intercal \mathbf{x}(t).
\end{aligned}$$

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 $\mathrm{E}\left(-\frac{1}{L}\mathbf{H}\right)$, $\mathrm{E}\left(-\frac{\eta}{L}\mathbf{H}\right)$, $\mathrm{E}\left(\frac{1}{L}\mathbf{F}^\intercal\right)$, $\mathrm{E}(\mathbf{x}(t))$, $[[\mathbf{h}_u]]$, $[[\mathbf{l}_u]]$ can be written as follows. Denote the computation on the inputs mentioned previously as f_{iter}. We use both Add and $\oplus$, Mlt and $\otimes$ for a better visual representation.

$$\begin{aligned}
[[\mathbf{t}_k - \boldsymbol{\rho}_k]] = &\,\mathrm{Mlt}\left(\mathrm{E}\left(\frac{-1}{L}\mathbf{F}^\intercal\right), \mathrm{E}(\mathbf{x}(t))\right) \\
&\oplus \mathrm{Mlt}\left(\mathrm{Add}\left(\mathbf{I}_{Mm}, \mathrm{E}\left(\frac{1}{L}\mathbf{H}\right)\right), \mathrm{E}\left(\mathbf{U}_k\right)\right) \oplus \\
&\oplus \mathrm{Mlt}\left(\mathrm{Add}\left(\mathrm{E}(\eta) \otimes \mathbf{I}_{Mm}, \mathrm{E}\left(\frac{-\eta}{L}\mathbf{H}\right)\right), \big(\mathrm{E}(\mathbf{U}_k) \ominus \mathrm{E}(\mathbf{U}_{k-1})\big)\right),
\end{aligned} \tag{9.8}$$

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, let us address how to perform (9.4b) in a private way. 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. One solution to the private comparison was described in Sect. 9.4.5. The cloud has $[[\mathbf{t}_k]]$ and assume it also has the AHE encryptions of the limits $[[\mathbf{h}_u]]$ and $[[\mathbf{l}_u]]$. The cloud and the actuator will engage in two instances of the DGK protocol and oblivious transfer: first, to compare $\mathbf{t}_k$ to $\mathbf{h}_u$ and obtain an intermediate value of $\mathbf{U}_{k+1}$, and second, to compare $\mathbf{U}_{k+1}$ to $\mathbf{l}_u$ and to update the iterate $\mathbf{U}_{k+1}$.

Before calling the comparison Protocol 2, described in Sect. 9.4.5, the cloud should randomize the order of the inputs, such that, after obtaining the comparison bit, the actuator does not learn whether the iterate was feasible or not. This is done in lines 8 and 11 in Protocol 4. Upon the completion of the comparison, the cloud and actuator perform the oblivious transfer variant, described in Sect. 9.4.4, such that the cloud obtains the intermediate value of $[[\mathbf{U}_{k+1}]]$ and subsequently, update the AHE encryption of the iterate $[[\mathbf{U}_{k+1}]]$. At the last iteration, the cloud and actuator perform the standard oblivious transfer for the first m positions in the values such that the actuator obtains $\mathbf{u}(t)$. Finally, because the next iteration can proceed only if the cloud has access to the full LabHE encryption $\mathrm{E}(\mathbf{U}_{k+1})$, 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 $\mathrm{E}(\mathbf{U}_{k+1} - \mathbf{r}_{k+1}) = \left(\mathbf{U}_{k+1} - \mathbf{r}_{k+1}, \left[\left[\mathfrak{b}^U_{k+1}\right]\right]\right)$. Then, the cloud recovers the LabHE encryption as $\mathrm{E}(\mathbf{U}_{k+1}) = \mathrm{Add}(\mathrm{E}(\mathbf{U}_{k+1} - \mathbf{r}_{k+1}), \mathbf{r}_{k+1})$. In what follows, we will outline the private protocols obtained by integrating the above observations.

9.5.1 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 3: Initialization of encrypted MPC
Require: Actuator: $f_{\text{MPC}}$; Subsystems: $\mathcal{U}^i$; Setup: $\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}$.
Ensure: Subsytems: mpk, $\text{upk}_{S^i}$, $\text{usk}_{S^i}$, $\boldsymbol{\tau}_{S^i}$, $\mathfrak{b}_{S^i}$, $[[\mathfrak{b}_{S^i}]]$, $\mathfrak{R}_{S^i}$; Setup: $\mathbf{H}$, $\mathbf{F}$, $\eta$, $L$, mpk, $\text{upk}_{Set}$, $\text{usk}_{Set}$, $\boldsymbol{\tau}_{Set}$, $\mathfrak{b}_{Set}$, $[[\mathfrak{b}_{Set}]]$, $\mathfrak{R}_{Set}$; Actuator: usk, upk, $\boldsymbol{\tau}_A$, $\mathfrak{b}_A$, $[[\mathfrak{b}_A]]$, $\mathfrak{R}_A$; Cloud: mpk, $\text{E}\left(-\frac{1}{L}\mathbf{H}\right)$, $\text{E}\left(-\frac{\eta}{L}\mathbf{H}\right)$, $\text{E}\left(\frac{1}{L}\mathbf{F}^\intercal\right)$, $\text{E}(\eta)$, $[[\mathbf{h}_u]]$, $[[\mathbf{l}_u]]$, $\mathfrak{R}_C$.
    Offline:
 1: Actuator: Generate (mpk, msk) $\leftarrow \text{Init}(1^\sigma)$ and distribute mpk to the others. Also generate a key usk for itself.
 2: Subsystems, Setup: Each get (usk, upk) $\leftarrow$ KeyGen(mpk) and send upk to the actuator.
 3: Subsystems, Setup, Actuator: Allocate labels to the inputs of function $f_{\text{MPC}}$: $\tau_1, \dots, \tau_v$, 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 $\tau_{\mathbf{x}^i(t)} = [0\ 1\ n^i \dots n^i - 1]^\intercal$.
    Setup: for matrix $\mathbf{H} \in \mathbb{R}^{Mm \times Mm}$, set $l = 0$, generate
 4: $\tau_{\mathbf{H}} = \begin{bmatrix} l & l+1 & \dots & l+Mm-1 \\ \vdots & & & \vdots \\ l+(Mm-1)Mm & l+(Mm-1)Mm+1 & \dots & l+M^2m^2-1 \end{bmatrix}$ and update $l = M^2m^2$, 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 $k = 0, \dots, K-1$.
 5: Subsystems, Setup, Actuator, Cloud: Generate randomness for blinding and encryptions $\mathfrak{R}$.
 6: 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 $\mathfrak{b}$, $[[\mathfrak{b}]]$.
 7: Actuator: Generate initializations for the initial iterate $\mathbf{U}_0'$.
 8: Actuator: Form the program $\mathcal{P} = (f_{\text{MPC}}, \tau_1, \dots, \tau_v)$.
    Initialization:
 9: 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: $\text{E}\left(-\frac{1}{L}\mathbf{H}\right)$, $\text{E}\left(-\frac{\eta}{L}\mathbf{H}\right)$, $\text{E}\left(\frac{1}{L}\mathbf{F}^\intercal\right)$, $\text{E}(\eta)$.
10: 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 4: Encrypted MPC step
Require: Actuator: $f_{\text{MPC}}$; Subsystems: $\mathbf{x}^i(t), \mathcal{U}^i$; Setup: $\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}$.
Ensure: Actuator: $\mathbf{u}(t)$
    Offline + Initialization:
 1: Subsystems, Setup, Cloud, Actuator: Run Protocol 3.
    Online:
 2: Cloud: $\left[\left[\frac{1}{L}\mathbf{F}^\intercal\mathbf{x}(t)\right]\right] \leftarrow \text{Mlt}\left(\text{E}\left(\frac{1}{L}\mathbf{F}^\intercal\right), \text{E}(\mathbf{x}(t))\right)$.
 3: Actuator: Send the initial iterate to the cloud: $\text{E}(\mathbf{U}_0')$.
 4: Cloud: Change the initial iterate: $\text{E}(\mathbf{U}_0) = \text{Add}\left(\text{E}\left(\mathbf{U}_0'\right), \mathbf{r}_0\right)$.
 5: Cloud: $\text{E}(\mathbf{U}_{-1}) \leftarrow \text{E}(\mathbf{U}_0)$.
 6: for $k = 0, \dots, K-1$ do
 7:     Cloud: Compute $[[\mathbf{t}_k]]$ as in Equation (9.8).
```

```
8:     Cloud: $([[\mathbf{a}_k]], [[\mathbf{b}_k]]) \leftarrow$ randomize $([[\mathbf{h}_u]], [[\mathbf{t}_k]])$.
9:     Cloud, Actuator: Execute comparison Protocol 2; Actuator obtains $\boldsymbol{\delta}_k$.
10:    Cloud, Actuator: $\left[\left[\mathbf{U}_{k+1}\right]\right] \leftarrow \mathrm{OT}'\,([[\mathbf{a}_k]], [[\mathbf{b}_k]], \boldsymbol{\delta}_k, \mathrm{msk})$.
11:    Cloud: $([[\mathbf{a}_k]], [[\mathbf{b}_k]]) \leftarrow$ randomize $\left([[\mathbf{l}_u]], \left[\left[\mathbf{U}_{k+1}\right]\right]\right)$.
12:    Cloud, Actuator: Execute comparison Protocol 2; Actuator obtains $\boldsymbol{\delta}_k$.
13:    if $k! = K - 1$ then
14:        Cloud, Actuator: $\left[\left[\mathbf{U}_{k+1}\right]\right] \leftarrow \mathrm{OT}'\,([[\mathbf{a}_k]], [[\mathbf{b}_k]], \boldsymbol{\delta}_k \veebar \mathbf{1}, \mathrm{msk})$. Cloud receives $\left[\left[\mathbf{U}_{k+1}\right]\right]$.
15:        Cloud: Send to the actuator $\left[\left[\mathbf{U}'_{k+1}\right]\right] \leftarrow$ Add $\left(\left[\left[\mathbf{U}_{k+1}\right]\right], [[-\mathbf{r}_k]]\right)$, 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 $\mathrm{E}(\mathbf{U}'_{k+1})$.
17:        Cloud: $\mathrm{E}\,(\mathbf{U}_{k+1}) \leftarrow$ Add $\left(\mathrm{E}\left(\mathbf{U}'_{k+1}\right), \mathbf{r}_k\right)$.
18:    else
19:        Cloud, Actuator: $\mathbf{u}(t) \leftarrow \mathrm{OT}\,([[\mathbf{a}_k]]_{1:m}, [[\mathbf{b}_k]]_{1:m}, \{\boldsymbol{\delta}_k\}_{1:m} \veebar \mathbf{1}, \mathrm{msk})$. 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.

9.5.2 Privacy of Protocol 4

Assumption 2 An adversary cannot corrupt at the same time both the cloud controller and the virtual actuator or more than $M-1$ subsystems.

Theorem 9.1 *Under Assumption 2, the encrypted MPC solution presented in Protocol 4 achieves multi-party privacy (Definition 9.4).*

Proof 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 9.4. We are going to 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:

$$\begin{aligned} V_C(\emptyset) = \Bigg(&\mathrm{mpk}, \mathrm{E}\left(-\frac{1}{L}\mathbf{H}\right), \mathrm{E}\left(-\frac{\eta}{L}\mathbf{H}\right), \mathrm{E}\left(\frac{1}{L}\mathbf{F}^{\intercal}\right), \mathrm{E}(\eta), \\ &[[\mathbf{h}_u]], [[\mathbf{l}_u]], \mathfrak{R}_C, \mathrm{E}(\mathbf{U}'_0), \\ &\left\{\mathrm{E}(\mathbf{U}_k), [[\mathbf{a}_k]], [[\mathbf{b}_k]], \left[\left[U_{k+1}\right]\right], \mathrm{msg}_{\mathrm{Pr.2}}, \mathrm{msg}_{\mathrm{OT}}\right\}_{k\in\{0,\ldots,K-1\}}\Bigg). \end{aligned} \tag{9.9}$$

The actuator's input is the function f_{MPC} and the output is $\mathbf{u}(t)$. Then, its view is:

$$V_A(f_{\text{MPC}}) = \Big(f_{\text{MPC}}, \text{mpk}, \text{msk}, \text{upk}, \mathfrak{R}_A, \\ \Big\{\mathbf{U}'_{k+1}, \text{msg}_{\text{Pr}.2}, \text{msg}_{\text{OT}}\Big\}_{k\in\{0,\dots,K-1\}}, \mathbf{u}(t)\Big), \tag{9.10}$$

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 4, since it is just a helper entity. Its view is:

$$V_{Set}(\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}) = \Big(\mathbf{A}, \mathbf{B}, \mathbf{P}, \mathbf{Q}, \mathbf{R}, \text{mpk}, \text{usk}_{Set}, \mathfrak{R}_{Set}\Big). \tag{9.11}$$

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 4. Its view is:

$$V_{S^i}(\mathcal{U}^i, \mathbf{x}^i(t)) = \Big(\mathcal{U}^i, \mathbf{x}^i(t), \text{mpk}, \text{usk}_{S^i}, \mathfrak{R}_{S^i}\Big). \tag{9.12}$$

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 [1], 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 $(\widetilde{\mathcal{T}}, \widetilde{\mathfrak{b}}, \widetilde{[[\mathfrak{b}]]})_j$. 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 $\text{E}\,\widetilde{\left(-\frac{1}{L}\mathbf{H}\right)}$, $\text{E}\,\widetilde{\left(-\frac{\eta}{L}\mathbf{H}\right)}$, $\text{E}\,\widetilde{\left(\frac{1}{L}\mathbf{F}^{\intercal}\right)}$, $\widetilde{\text{E}(\eta)}$, $\widetilde{[[\mathbf{h}_u]]}$, $\widetilde{[[\mathbf{l}_u]]}$, $\widetilde{\text{E}(\mathbf{U}'_0)}$. S_C generates the coins $\widetilde{\mathfrak{R}}_C$ as in line 4 in Protocol 3 and obtains $\text{E}(\mathbf{U}_0)$ as in line 4 in Protocol 4. Then, for each $k = \{0, \dots, K-1\}$, it computes $\widetilde{[[\mathbf{t}_k]]}$ as in line 7 in Protocol 4 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 and gets $\widetilde{\text{msg}}_{\text{Pr}.2}$. Furthermore, S_C generates an encryption of random bits $\widetilde{\boldsymbol{\delta}}_k$ and of $\widetilde{\text{E}(\mathbf{U}_k)}$ and performs the same steps as the simulator for party A as in the proof of Proposition 9.1 (or the simulator for the standard OT) and gets $\widetilde{\text{msg}}_{\text{OT}}$. It then outputs:

$$S_C(\emptyset) = \Bigg(\widetilde{\mathrm{mpk}}, \widetilde{\mathrm{E}\left(-\frac{1}{L}\mathbf{H}\right)}, \widetilde{\mathrm{E}\left(-\frac{\eta}{L}\mathbf{H}\right)}, \widetilde{\mathrm{E}\left(\frac{1}{L}\mathbf{F}^\intercal\right)}, \widetilde{\mathrm{E}(\eta)},$$
$$\widetilde{[[\mathbf{h}_u]]}, \widetilde{[[\mathbf{l}_u]]}, \widetilde{\mathfrak{R}}_C, \widetilde{\mathrm{E}(\mathbf{U}_0')},$$
$$\left\{\widetilde{\mathrm{E}(\mathbf{U}_k)}, \left(\widetilde{[[\mathbf{a}_k]]}, \widetilde{[[\mathbf{b}_k]]}\right), \widetilde{[[U_{k+1}]]}, \widetilde{\mathrm{msg}}_{\mathrm{Pr.2}}, \widetilde{\mathrm{msg}}_{\mathrm{OT}}\right\}_{k\in\{0,\ldots,K-1\}}\Bigg). \tag{9.13}$$

All the values in the view of the cloud in (9.9)—with the exception of the random values $\mathfrak{R}_C$ and the key mpk, which are statistically indistinguishable from $\widetilde{\mathfrak{R}}_C$ and $\widetilde{\mathrm{mpk}}$ 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 (9.13), even with different keys. 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 $f_{\mathrm{MPC}}, \mathbf{u}(t)$. S_A will take the same steps as in lines 1, 3–7 in Protocol 3, obtaining $\widetilde{\mathrm{mpk}}, \widetilde{\mathrm{msk}}, \widetilde{\mathrm{upk}}, \widetilde{\mathrm{usk}}, \widetilde{\boldsymbol{\tau}}_A, \widetilde{\mathfrak{b}}_A, \widetilde{[[\mathfrak{b}_A]]}, \widetilde{\mathfrak{R}}_A, \widetilde{\mathbf{U}}_0'$ and instead of line 2, it generates $\widetilde{\mathrm{upk}}_j, \widetilde{\mathrm{usk}}_j \leftarrow \mathrm{KeyGen}(\widetilde{\mathrm{mpk}})$ 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 and gets $\widetilde{\mathrm{msg}}_{\mathrm{Pr.2}}$. Furthermore, S_A performs the same steps as the simulator for party B as in the proof of Proposition 9.1 (or the simulator for the standard OT) and gets $\widetilde{\mathrm{msg}}_{\mathrm{OT}}$. It then outputs:

$$S_A(f_{\mathrm{MPC}}, \mathbf{u}(t)) = \Bigg(f_{\mathrm{MPC}}, \widetilde{\mathrm{mpk}}, \widetilde{\mathrm{msk}}, \widetilde{\mathrm{upk}}, \widetilde{\mathfrak{R}}_A,$$
$$\left\{\widetilde{\mathbf{U}_{k+1}'}, \widetilde{\mathrm{msg}}_{\mathrm{Pr.2}}, \widetilde{\mathrm{msg}}_{\mathrm{OT}}\right\}_{k\in\{0,\ldots,K-1\}}, \mathbf{u}(t)\Bigg). \tag{9.14}$$

All the values in the view of the actuator in (9.10)—with the exception of the random values $\mathfrak{R}_A$ and the keys upk, which are statistically indistinguishable from $\widetilde{\mathfrak{R}}_A$ and $\widetilde{\mathrm{upk}}$ 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 (9.14). 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, $V_A(f_{\mathrm{MPC}}) \overset{c}{\equiv} S_A(f_{\mathrm{MPC}}, \mathbf{u}(t))$.

The setup and subsystems do not receive any other messages apart from the master public key (9.11), (9.12). Hence, a simulator S_{Set} for the setup and a simulator S_{S^i} for a subsystem i can simply generate $\widetilde{\mathrm{mpk}} \leftarrow \mathrm{Init}(1^\sigma)$ and then proceed with the execution of lines 2–5 in Protocol 3 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 a number l of subsystems is the following:

$$V_{ASl}\big(f_{\text{MPC}},\mathbf{u}(t),\ \mathbf{A},\mathbf{B},\mathbf{P},\mathbf{Q},\mathbf{R},\ \{\mathcal{U}^i,\mathbf{x}^i(t)\}_{i\in i_1,\dots,i_l}\big) = V_A(f_{\text{MPC}},\mathbf{u}(t))\cup$$
$$\cup V_{Set}(\mathbf{A},\mathbf{B},\mathbf{P},\mathbf{Q},\mathbf{R})\cup V_{S^{i_1}}(\mathcal{U}^{i_1},\mathbf{x}^{i_1}(t))\cup\ldots\cup V_{S^{i_l}}(\mathcal{U}^{i_l},\mathbf{x}^{i_l}(t)).$$

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. □

We can also have the private MPC scheme run for multiple time steps. Protocol 3 can be modified to also generate the labels and secrets for T time steps. Protocol 4 can be run for multiple time steps, and warm starts can be included by adding two lines such that the cloud obtains $\mathrm{E}\left(\left\{\mathbf{U}_K^t\right\}_{m+1:M}\right)$ and sets $\mathrm{E}(\mathbf{U}_0^{t+1}) = \left[\,\mathbf{U}_K^{t\,\intercal}\ \mathbf{0}_m^\intercal\,\right]^\intercal$.

9.5.3 Analysis of Errors

As mentioned at the beginning of the section, the values that are encrypted, 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. We multiply the values by 2^{l_f} then truncate to obtain integers prior to encryption, and, after decryption, we divide the result by the appropriate quantity (e.g., we divide the result of a multiplication by 2^{2l_f}). Furthermore, the operations can increase the number of bits of the result, hence, before the comparisons in Protocol 4 an extra step that performs interactive truncation has to be performed, because Protocol 2 requires a fixed number of bits for the inputs. Also, notice that when the encryption is refreshed, i.e., a ciphertext is decrypted and re-encrypted, the accumulation of bits due to the operations is truncated back to the desired size.

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 the FGM algorithm in (9.4). In order to bound it, we need to ensure that the number of fractional bits l_f is sufficiently large such that the feasibility of the fixed-point precision solution is preserved, that the strong convexity of the fixed-point objective function still holds and that the fixed-point step size is such that the FGM converges. The errors can be written as states of a stable linear system with bounded disturbances. Bounds on the errors for the case of public model are derived in [2] and similar bounds can be obtained for the private model. The bounds on this deviation can be used in an offline step to choose an appropriate fixed-point precision for the desired performance of the system.

9.6 Discussion

Secure multi-party computation protocols require many rounds of communication in general. This can also be observed in Protocol 4. Therefore, in order to be able to use this proposed 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 (9.4) 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 [3] 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 [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.

References

1. Alexandru AB, Gatsis K, Shoukry Y, Seshia SA, Tabuada P, Pappas GJ (2018) Cloud-based quadratic optimization with partially homomorphic encryption. arXiv preprint arXiv:1809.02267
2. Alexandru AB, Morari M, Pappas GJ (2018) Cloud-based MPC with encrypted data. In: IEEE conference on decision and control (CDC), pp 5014–5019
3. Alexandru AB, Pappas GJ (2019) Encrypted LQG using Labeled Homomorphic Encryption. In: 10th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), pp 129–140
4. Ali M, Khan SU, Vasilakos AV (2015) Security in cloud computing: opportunities and challenges. Inf Sci 305:357–383
5. Archer D, Chen L, Cheon JH, Gilad-Bachrach R, Hallman RA, Huang Z, Jiang X, Kumaresan R, Malin BA, Sofia H, Song Y, Wang S (2017) Applications of homomorphic encryption. Technical report, Microsoft Research
6. Aslett LJ, Esperança PM, Holmes CC (2015) A review of homomorphic encryption and software tools for encrypted statistical machine learning. arXiv preprint arXiv:1508.06574
7. Barbosa M, Catalano D, Fiore D (2017) Labeled homomorphic encryption. In: European Symposium on Research in Computer Security, pp 146–166. Springer, Cham
8. Beimel A (2011) Secret-sharing schemes: a survey. In: International conference on coding and cryptology, pp 11–46. Springer, Berlin
9. Bellare M, Hoang VT, Rogaway P (2012) Foundations of garbled circuits. In: Conference on computer and communications security, pp 784–796. ACM
10. Bellovin SM (2011) Frank Miller: inventor of the one-time pad. Cryptologia 35(3):203–222
11. Borrelli F, Bemporad A, Morari M (2017) Predictive control for linear and hybrid systems. Cambridge University Press
12. Bost R, Popa RA, Tu S, Goldwasser S (2015) Machine learning classification over encrypted data. In: Network & distributed system security symposium (NDSS)
13. Botta A, De Donato W, Persico V, Pescapé A (2016) Integration of cloud computing and internet of things: a survey. Future Gener Comput Syst 56:684–700
14. Catalano D, Fiore D (2015) Boosting linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. Cryptology ePrint Archive, Report 2014/813. https://eprint.iacr.org/2014/813
15. Catalano D, Fiore D (2015) Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In: 22nd ACM SIGSAC conference on computer and communications security, pp 1518–1529. ACM
16. Chase M, Gilad-Bachrach R, Laine K, Lauter K, Rindal P (2017) Private collaborative neural network learning. Technical report, Cryptology ePrint Archive, Report 2017/762 . https://eprint.iacr.org/2017/762
17. Chen H, Gilad-Bachrach R, Han K, Huang Z, Jalali A, Laine K, Lauter K (2018) Logistic regression over encrypted data from fully homomorphic encryption. BMC Med Genomics 11(4):81
18. Couteau G (2016) Efficient secure comparison protocols. Cryptology ePrint Archive, Report 2016/544. http://eprint.iacr.org/2016/544

19. Cramer R, Damgård I, Nielsen JB (2012) Secure multiparty computation and secret sharing-an information theoretic approach. Book draft
20. Cramer R, Damgård IB, Nielsen JB (2015) Secure multiparty computation. Cambridge University Press
21. Damgård I, Geisler M, Krøigaard M (2007) Efficient and secure comparison for on-line auctions. In: Australasian conference on information security and privacy, pp 416–430. Springer, Berlin
22. Damgård I, Geisler M, Krøigaard M (2009) A correction to "Efficient and secure comparison for on-line auctions". Int J Appl Cryptogr 1(4):323–324
23. Damgård I, Orlandi C (2010) Multiparty computation for dishonest majority: from passive to active security at low cost. In: Annual cryptology conference, pp 558–576. Springer
24. Damgård IB, Jurik M (2001) A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: International workshop on public key cryptography, pp 119–136. Springer, Berlin
25. Dwork C (2008) Differential privacy: a survey of results. In: International conference on theory and applications of models of computation, pp 1–19. Springer, Berlin
26. Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M Our data, ourselves: privacy via distributed noise generation. In: Annual international conference on the theory and applications of cryptographic techniques, pp 486–503. Springer (2006)
27. Dwork C, Roth A et al (2014) The algorithmic foundations of differential privacy. Found Trends® Theor Comput Sci **9**(3–4), 211–407
28. Farokhi F, Shames I, Batterham N (2017) Secure and private control using semi-homomorphic encryption. Control Eng Pract 67:13–20
29. Gentry C (2009) A fully homomorphic encryption scheme. Ph.D. thesis, Department of Computer Science, Stanford University. http://www.crypto.stanford.edu/craig
30. Gentry C, Boneh D (2009) A fully homomorphic encryption scheme, vol 20, no 09. Stanford University Stanford
31. Goldreich O (2003) Foundations of cryptography: basic tools, vol 1. Cambridge University Press, New York
32. Goldreich O (2004) Foundations of cryptography: basic applications, vol 2. Cambridge University Press, New York
33. Goldreich O, Micali S, Wigderson A (1987) How to play any mental game. In: 19th annual ACM symposium on theory of computing, pp 218–229. ACM
34. Goldwasser S, Micali S (1982) Probabilistic encryption & how to play mental poker keeping secret all partial information. In: 14th annual ACM symposium on Theory of Computing, pp 365–377. ACM
35. Gonzalez-Serrano FJ, Amor-Martın A, Casamayon-Anton J (2014) State estimation using an extended Kalman filter with privacy-protected observed inputs. In: IEEE international workshop on information forensics and security (WIFS), pp 54–59. IEEE
36. Hamlin A, Schear N, Shen E, Varia M, Yakoubov S, Yerukhimovich A (2016) Cryptography for big data security. In: Hu F (ed) Big data: storage, sharing, and security, Chap 10, pp 241–288. Taylor & Francis LLC, CRC Press
37. Ishai Y, Prabhakaran M, Sahai A (2008) Founding cryptography on oblivious transfer—efficiently. In: Annual international cryptology conference, pp 572–591. Springer, Berlin
38. Jeckmans A, Peter A, Hartel P (2013) Efficient privacy-enhanced familiarity-based recommender system. In: Proceedings of European symposium on research in computer security, pp 400–417. Springer, Berlin
39. Joye M, Libert B (2013) Efficient cryptosystems from 2^k-th power residue symbols. In: International conference on the theory and applications of cryptographic techniques, pp 76–92. Springer, Berlin
40. Kim J, Lee C, Shim H, Cheon JH, Kim A, Kim M, Song Y (2016) Encrypting controller using fully homomorphic encryption for security of cyber-physical systems. IFAC-PapersOnLine 49(22):175–180

41. Lindell Y (2017) How to simulate it–a tutorial on the simulation proof technique. In: Tutorials on the foundations of cryptography, pp 277–346. Springer International Publishing
42. Martins P, Sousa L, Mariano A (2018) A survey on fully homomorphic encryption: an engineering perspective. ACM Comput Surv (CSUR) 50(6):83
43. Mayne DQ, Rawlings JB, Rao CV, Scokaert PO (2000) Constrained model predictive control: stability and optimality. Automatica 36(6):789–814
44. Mell P, Grance T et al (2011) The NIST definition of cloud computing
45. Mirhoseini A, Sadeghi AR, Koushanfar F (2016) Cryptoml: secure outsourcing of big data machine learning applications. In: IEEE International symposium on hardware oriented security and trust (HOST), pp 149–154. IEEE
46. Mohassel P, Zhang Y (2017) SecureML: a system for scalable privacy-preserving machine learning. Cryptology ePrint Archive, Report 2017/396. http://eprint.iacr.org/2017/396
47. Murguia C, Farokhi F, Shames I (2018) Secure and private implementation of dynamic controllers using semi-homomorphic encryption. arXiv preprint arXiv:1812.04168
48. Naehrig M, Lauter K, Vaikuntanathan V (2011) Can homomorphic encryption be practical? In: 3rd ACM workshop on cloud computing security workshop, pp 113–124. ACM
49. Naor M, Pinkas B (2001) Efficient oblivious transfer protocols. In: 12th annual ACM-SIAM symposium on discrete algorithms, pp 448–457. SIAM
50. Nesterov Y (2013) Introductory lectures on convex optimization: a basic course, vol 87. Springer Science & Business Media
51. Nielsen JB, Nordholt PS, Orlandi C, Burra SS (2012) A new approach to practical active-secure two-party computation. In: Advances in cryptology–CRYPTO, pp 681–700. Springer, Berlin
52. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: Annual international conference on the theory and applications of cryptographic techniques, pp 223–238. Springer, Berlin
53. Pedersen TP (1991) Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference, pp 129–140. Springer, Berlin
54. Pettai M, Laud P (2015) Combining differential privacy and secure multiparty computation. In: 31st Annual computer security applications conference, pp 421–430. ACM
55. Rabin MO (2005) How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187. https://eprint.iacr.org/2005/187
56. Rastogi V, Nath S (2010) Differentially private aggregation of distributed time-series with transformation and encryption. In: ACM SIGMOD International Conference on Management of data, pp 735–746. ACM
57. Riazi MS, Rouhani BD, Koushanfar F (2018) Deep learning on private data. IEEE Secur Privacy Mag
58. Rittinghouse JW, Ransome JF (2016) Cloud computing: implementation, management, and security. CRC Press
59. Rivest RL, Adleman L, Dertouzos ML (1978) On data banks and privacy homomorphisms. Found Secure Comput 4(11):169–180
60. Schulze Darup M, Redder A, Shames I, Farokhi F, Quevedo D (2018) Towards encrypted MPC for linear constrained systems. IEEE Control Syst Lett 2(2):195–200
61. Shamir A (1979) How to share a secret. Commun ACM 22(11):612–613
62. Shi E, Chan HTH, Rieffel E, Chow R, Song D (2011) Privacy-preserving aggregation of time-series data. In: Network & distributed system security symposium (NDSS)
63. Singh S, Jeong YS, Park JH (2016) A survey on cloud computing security: issues, threats, and solutions. J Netw Comput Appl 75:200–222
64. Vadhan S (2018) Multiparty differential privacy. In: Differential privacy meets multi-party computation (DPMPC) workshop. https://www.bu.edu/hic/dpmpc-2018/
65. Vernam GS (1926) Cipher printing telegraph systems: for secret wire and radio telegraphic communications. J AIEE 45(2):109–115
66. Veugen T (2010) Encrypted integer division. In: International workshop on information forensics and security, pp 1–6. IEEE

67. Veugen, T.: Improving the DGK comparison protocol. In: International workshop on information forensics and security, pp 49–54. IEEE (2012)
68. Yao AC (1982) Protocols for secure computations. In: 23rd Annual symposium on foundations of computer science, pp 160–164. IEEE
69. Zhu T, Li G, Zhou W, Philip SY (2017) Differential privacy and applications. Springer, Cham