Encrypted LQG using Labeled Homomorphic Encryption

Andreea B. Alexandru
Dept. of Electrical and Systems Engineering
University of Pennsylvania
aandreea@seas.upenn.edu

George J. Pappas
Dept. of Electrical and Systems Engineering
University of Pennsylvania
pappasg@seas.upenn.edu

ABSTRACT

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

CCS CONCEPTS

• **Security and privacy** → **Public key encryption**; **Privacy-preserving protocols**; • **Information systems** → *Process control systems*; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

privacy and security, encrypted control

ACM Reference Format:
Andreea B. Alexandru and George J. Pappas. 2019. Encrypted LQG using Labeled Homomorphic Encryption. In *10th ACM/IEEE International Conference on Cyber-Physical Systems (with CPS-IoT Week 2019) (ICCPS '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3302509.3311049

1 INTRODUCTION

In the setting of distributed systems with large number of sensors that collect data over large periods of times, e.g., FitBit data, medical monitoring data or parameters in a plant, the data from the sensors is aggregated, stored and processed at a powerful server, generically called cloud. Requesting parties submit the processing algorithms they want to perform on the data stored at the cloud. Although cloud computing solves the storage and computation problems, it also raises issues about trust and privacy of the data and results [34]. Users agree to participate in the computation if their data is guaranteed to remain concealed from both the cloud and requester. Similarly, a requesting party desires to keep private the parameters of its processing algorithms, as well as the result of the computation.

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

General frameworks under the umbrella of Secure Multiparty Computation (SMPC) [9] have been proposed to solve the problem of private computations with data collected from multiple parties. While their generality is desirable in some cases, it is also advantageous to exploit the particularities of the specific architecture and the computation that is performed. Specifically, one of the solutions in the literature called Labeled Homomorphic Encryption [4], which we will explore in this paper, achieves accelerated complex operations on encrypted data by making use of the knowledge of the algorithm the cloud computes by the requester party.

1.1 Related work

There are several recent approaches in the literature that explore privacy-preserving filters and controllers, with the goal of concealing the private information from untrusted parties. Linear encrypted controllers and filters are presented in [12, 27] using additively homomorphic encryption – the signals are encrypted but the gains are public; using multiplicatively homomorphic encryption in [22, 35] – both the gains and signals are encrypted, but the decryptor is able to find out not just the final result, but also products of scalars before summation, which can leak at least which signals or gain entries are zero; and using fully homomorphic encryption in [20] – both the gains and signals are encrypted and only the final result is revealed, but the scheme is computationally prohibitive.

SMPC approaches, based on secret sharing schemes that can be combined with homomorphic encryption and/or garbled circuits,

ICCPS '19, April 16–18, 2019, Montreal, QC, Canada
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6285-6/19/04…$15.00
https://doi.org/10.1145/3302509.3311049

are considered in the following works: an encrypted Extended Kalman Filter was explored in [16], where the encrypted gains are computed by repeated exchanges between a client and a server that achieve encrypted complex operations; an encrypted linear Finite Impulse Response filter with plaintext coefficients is computed in [31]; and in [2], an encrypted multi-sensor information filtering is proposed, where a grid operator aggregates the encrypted estimates from sensors and sends it to the mobile agent requesting its location.

An approach with a different privacy goal is Differential Privacy (DP), which adds randomness to the inputs, computation or output, so that the inputs cannot be reconstructed from the resulting output. Techniques from works such as Kalman Filter with DP [23] and LQG with DP [17] can be used to augment the output privacy of a secure filter and controller. However, such techniques reduce the accuracy of the result and stability is difficult to guarantee.

1.2 Our contribution

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

1.3 Notation

We denote the $n \times n$ identity matrix by I_n and a matrix of zeros of size $m \times n$ by $0_{m\times n}$. For a positive integer n, we use $[n] := \{1, \dots, n\}$. We call messages that are not encrypted *plaintexts*, and the encrypted messages *ciphertexts*. $\{0,1\}^*$ denotes a sequence of bits of unspecified length. For a set S, $s \overset{\$}{\leftarrow} S$ denotes drawing s uniformly at random from S. The security parameter is called λ.

2 PROBLEM SETUP

We consider agents or sub-systems in the architecture in Figure 1, with local sensors, and an actuator that needs to apply the control inputs based on the measurements and references from the agents. The agents and the actuator employ a cloud server to privately compute a Linear Quadratic Gaussian (LQG) controller, without having access to the model of the system, control gains, the measurements or the desired references. The system has the following model:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + w_k \\ z_k &= Cx_k + v_k, \quad k = 0, \dots, N-1, \end{aligned} \tag{1}$$

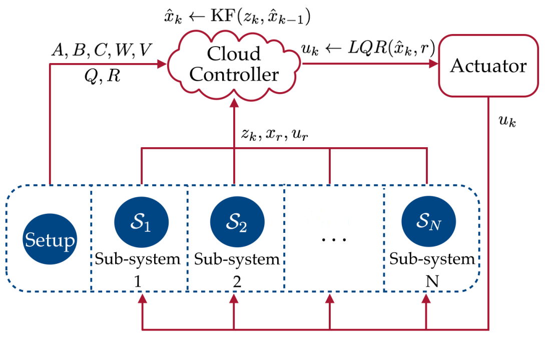

Figure 1: Architecture of the cloud-outsourced LQG problem: the sub-systems send their measurements and desired references to the cloud. The cloud has to run the LQG algorithm on the measurements and the system's matrices and send the result to the actuator. The variables in the figure are described in equations (1)–(4).

where $x_k, w_k \in \mathbb{R}^n, u_k \in \mathbb{R}^m, z_k, v_k \in \mathbb{R}^p$. Each sub-system $i \in [N]$ has a partition of the states $x_k^i \in \mathbb{R}^{n_i}$, a partition of the control inputs $u_k^i \in \mathbb{R}^{m_i}$ and a partition of the measurements $z_k^i \in \mathbb{R}^{p_i}$, such that their union forms system (1). The distributed system has one proxy entity that facilitates the cloud-computation: a setup entity, which holds the model of the system, that is not fully known by each individual sub-system. As we will see, the sub-systems want to conceal their data from the other participants in the computation, hence having partitions of the data is justified in this context.

We assume that the process and measurement noise vectors are uncorrelated i.i.d. random variables with zero mean and known positive semi-definite and respectively, positive definite covariance matrices: $\mathrm{E}[w_k w_k^\intercal] = W$ and $\mathrm{E}[v_k v_k^\intercal] = V$. The initial state is a random Gaussian variable with a finite mean and covariance matrix. Furthermore, we assume that $\{x_0, w_1, \dots, w_k, v_1, \dots, v_k\}$ are mutually independent.

Our first control objective is to achieve stability for the system (1). The separation principle [5] allows the optimal control problem to be divided into two successive steps: the design of an optimal estimator for the system's state, which is the Kalman Filter under the assumption that the process and measurement noise vectors have a Gaussian distribution; and the design of an optimal controller for the system with the perfect information given by the estimator, which we achieve by minimizing a quadratic cost.

Our second control objective is to steer the system to a reference r for the measurements, x_r for the states and u_r for the control inputs, composed by the desired references of each sub-system. We want to determine the control input u_k such that the output deviation $\Delta z_k := z_k - r$, the state deviation $\Delta x_k := x_k - x_r$ and the control deviation $\Delta u_k := u_k - u_r$ are small for all values of k. We can write the system:

$$\begin{aligned} \Delta x_{k+1} &= A\Delta x_k + B\Delta u_k + w_k \\ \Delta z_k &= C\Delta x_k + v_k, \quad k = 0, \dots, N-1, \end{aligned} \tag{2}$$

For simplicity, we will consider the stationary LQG problem in this paper, which is often used in real-time implementations due to the low memory requirements [5]. Assuming that the process and measurement noise processes are white, Gaussian and stationary, the controllability (stabilizability) of the pairs (A, B) and $(A, W^{\frac{1}{2}})$ and the observability (detectability) of the pairs (A, C) and $(A, Q^{\frac{1}{2}})$, the LQG problem with the cost over a horizon T:

$$J = \mathbb{E}\Big[\Delta z_T^\intercal Q \Delta z_T + \sum_{k=0}^{T-1}\left(\Delta z_k^\intercal Q \Delta z_k + \Delta u_k^\intercal R \Delta u_k\right)\Big], \tag{3}$$

allows an infinite horizon solution [3, 5]:

$$\begin{aligned} u_k &= -K(\hat{x}_k - x_r) + u_r, \\ \hat{x}_{k+1} &= A\hat{x}_k + Bu_k + L\left(z_{k+1} - C(A\hat{x}_k + Bu_k)\right), \end{aligned} \tag{4}$$

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

$$\begin{aligned} &L = PC^\intercal\left(CPC^\intercal + V\right)^{-1}, \; K = \left(B^\intercal SB + R\right)^{-1} B^\intercal SA \\ &P = A^\intercal PA - A^\intercal PC^\intercal\left(CPC^\intercal + V\right)^{-1} CPA + W \\ &S = A^\intercal SA - A^\intercal SB\left(B^\intercal SB + R\right)^{-1} B^\intercal SA + Q. \end{aligned} \tag{5}$$

The matrices $A - BK$ and $A - LCA$ determine the stability of the closed-loop system and the quality of the estimation. Notice that an iteration of the LQG (4) can be written as a multivariate polynomial in $\hat{x}_k, u_k, x_r, u_r$ and A, B, C, K, L.

The LQG control presented can be abstracted in the following general framework in Figure 2. Consider a cloud server that collects encrypted data from several clients, which in Figure 1 were the sub-systems. The data represents time series and is labeled with the corresponding time. A requester, which in Figure 1 was the actuator, makes queries, e.g., an LQG iteration, that can be written as multivariate polynomials over the data stored at the cloud server and solicits the result. We allow *semi-honest* parties, which are parties that follow the preset protocols, but can locally process the data in order to try to infer private information. This is a reasonable assumption, as cloud services are reputation based and cannot afford to tamper with the clients' data. Furthermore, in this paper, we consider the classical computational privacy definition of an interactive protocol, given in [14, Ch. 7]. We assume that only computations that have been previously agreed upon can be requested, such that the privacy of the data stored at the cloud server is not broken (e.g. the requester cannot simply ask for the data).

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

3 LABELED HOMOMORPHIC ENCRYPTION AND EXTENSION

Labeled Homomorphic Encryption, abbreviated as LabHE, was recently introduced in [4] and is a scheme that allows the computation of multivariate degree-two polynomials on encrypted data. The appeal of LabHE is that it uses a simple addendum to an additively homomorphic scheme in order to obtain, apart from unlimited additions between encrypted values, also a multiplication between two encrypted values. The underlying homomorphic encryption scheme can be instantiated with most of the existing schemes and inherits their properties. LabHE exploits the common trait that the party that requests the result of the encrypted computation knows what the computation is.

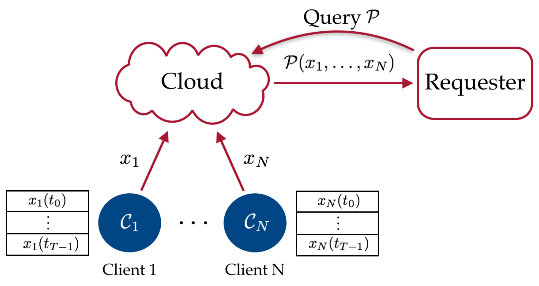

Figure 2: The clients send their private data (collected over T time steps or stored in a buffer) to a cloud server. A requester sends a query to the cloud, which evaluates it on the data and sends the result to the requester.

In what follows, we will first describe the cryptographic preliminaries of LabHE and then present the primitives that compose it. Furthermore, we propose a two-party extension of LabHE that achieves more encrypted multiplications at the expense of offline computation and communication. This extension is critical for the execution of LQG described in Section 2 in an encrypted manner.

3.1 Preliminaries

3.1.1 Additively Homomorphic Encryption. Additively Homomorphic Encryptions schemes, abbreviated as AHE, allow a party that only has encryptions of two messages to obtain an encryption of their sum. AHE can be instantiated by various public key additively homomorphic encryption schemes such as [10, 19, 25].

Let AHE=$(\hat{\text{KeyGen}}, \hat{\text{E}}, \hat{\text{D}}, \hat{\text{Add}}, \hat{\text{cMlt}})$ be an 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 will denote the encryption of a message $m \in \mathcal{M}$ by $[[m]]$ as a shorthand notation for $\hat{\text{E}}$(public key, m).

1. $\hat{\text{KeyGen}}(1^\lambda)$: Takes the security parameter λ and outputs a public key $\hat{\text{pk}}$ and a private key $\hat{\text{sk}}$.
2. $\hat{\text{E}}(\hat{\text{pk}}, m)$: Takes the public key and a message $m \in \mathcal{M}$ and outputs a ciphertext $[[m]] \in \hat{\mathcal{C}}$.
3. $\hat{\text{D}}(\hat{\text{sk}}, c)$: Takes the private key and a ciphertext $c \in \hat{\mathcal{C}}$ and outputs the message that was encrypted $m' \in \mathcal{M}$.
4. $\hat{\text{Add}}(c_1, c_2)$: Takes two 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:
$$\hat{\text{D}}(\hat{\text{sk}}, c) = \hat{\text{D}}(\hat{\text{sk}}, c_1) + \hat{\text{D}}(\hat{\text{sk}}, c_2).$$
5. $\hat{\text{cMlt}}(m_1, c_2)$: 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:
$$\hat{\text{D}}(\hat{\text{sk}}, c) = m_1 \cdot \hat{\text{D}}(\hat{\text{sk}}, c_2).$$

We consider only instantiations of the AHE that are proved to be semantically secure [15], [14, Ch. 4] and circuit-private [7]. Essentially, an encryption scheme achieves semantic security (or equivalently, ciphertext indistinguishability), if an adversary that is given two distinct plaintexts and an encryption of one of them is not able to distinguish which plaintext corresponds to the given ciphertext. Circuit-privacy defines the concept that the result of an evaluation primitive (e.g. $\hat{\text{Add}}$, $\hat{\text{cMlt}}$) does not reveal any information about the inputs of that primitive and about the operation evaluated. More details are provided in Appendix A.

3.1.2 Secret sharing. Secret sharing [26, 29] is a scheme that distributes a private message to a number of parties, by splitting it into 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 an additive abelian group, into two shares in the following way: generate uniformly at random an element $b \in G$, subtract it from the message and then distribute the shares b and $m - b$. This can be also thought of as an one-time pad 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 indistinguishable from one another, i.e., have identical distributions [24].

3.1.3 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 [13, Ch. 3].

3.2 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 scheme makes use of the fact that the decryptor (or requester in Figure 2) 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 Figure 2) assigns a unique label to each message and sends the encrypted data along with the corresponding encrypted labels to the server. Labels can be time instances, locations, id numbers etc.

Denote by $\mathcal{M}$ the message space. A program that has labeled inputs is called a labeled program [4].

Definition 1. *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 $\mathcal{P}_1, \dots, \mathcal{P}_t$.* ⋄

Definition 2. *An admissible function $f : \mathcal{M}^n \to \mathcal{M}$ is a multivariate polynomial of degree d on n variables, where $d = 2$ for the original version of* LabHE *and $d \geq 2$ for the extended version described in Section 3.3.* ⋄

LabHE is constructed from an AHE scheme with the requirement that the message space must be a public ring in which it is possible to efficiently sample elements uniformly at random. The idea is that an encryptor splits their private message as described in Section 3.1.2 into a random value (secret) and the difference between the message and the secret. The encryptor then forms the ciphertext from the encryption of the first share along with the second share. 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.

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 $k = \text{poly}(\lambda)$, where λ is the security parameter, and a label from $\mathcal{L}$. Then LabHE is defined as a tuple LabHE = (Init, KeyGen, E, Eval, D):

(1) $\text{Init}(1^\lambda)$: Takes the security parameter λ 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 usk_i and a user public key upk_i.
(3) $\text{E}(\text{mpk}, \text{upk}, \tau, m)$: 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:
- $\text{Off-E}(\text{usk}, \tau)$: Computes the secret $b \leftarrow F(usk, \tau)$ and outputs $\text{C}_\text{off} = (b, [[b]])$.
- $\text{On-E}(\text{C}_\text{off}, m)$: Outputs $C = (m - b, [[b]]) =: (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$.

(4) $\text{Eval}(\text{mpk}, f, C_1, \dots, C_t)$: 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:
- $\text{Mlt}(C_1, C_2)$: Takes $C_i = (a_i, \beta_i) \in \mathcal{M} \times \hat{\mathcal{C}}$ for $i = 1, 2$ and outputs $C = [[a_1 \cdot a_2]] \hat{\oplus} (a_1 \hat{\otimes} \beta_2) \hat{\oplus} (a_2 \hat{\otimes} \beta_1) = [[m_1 \cdot m_2 - b_1 \cdot b_2]] =: \alpha \in \hat{\mathcal{C}}$.
- $\text{Add}(C_1, C_2)$: 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 $C = (a_1, \beta_1 \hat{\oplus} \alpha_2) =: (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$.
- $\text{cMlt}(c, C')$: 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) $\text{D}(\text{msk}, \text{usk}_{1,\dots,t}, \mathcal{P}, C)$: Takes the master secret key, a vector of t user secret keys, a labeled program $\mathcal{P}$ and a ciphertext C. It is composed of an online and offline part:
- $\text{Off-D}(\text{msk}, \mathcal{P})$: Parses $\mathcal{P}$ as $(f, \tau_1, \dots, \tau_t)$. For $i \in [t]$, it computes the secrets $b_i = F(\text{usk}_i, \tau_i)$, $b = f(b_1, \dots, b_t)$ and outputs $\text{msk}_\mathcal{P}(\text{msk}, \text{b})$.
- $\text{On-D}(\text{sk}_\mathcal{P}, C)$: If $C = (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$: either output (i) $m = a + b$ or (ii) output $m = a + \hat{\text{D}}(\text{msk}, \beta)$. If $C \in \hat{\mathcal{C}}$, output $m = \hat{\text{D}}(\text{msk}, C) + b$.

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

In [4] it is proved that LabHE satisfies correctness (the probability of the scheme not correctly evaluating the allowed class of functions is negligible), succinctness (the ciphertexts have polynomial length in the security parameter), semantic security (the ciphertexts are indistinguishable from one another) and context-hiding (decrypting the ciphertext does not reveal anything about the inputs of the computed function, only the result of the function on those inputs).

3.3 New extension of LabHE to degree d-polynomials

In [7], the authors show how to obtain the evaluation of degree-3 and 4 polynomials over encrypted data by using, instead of AHE, schemes that are level-2 homomorphic, i.e., already support encrypted additions and one encrypted multiplication, such as BGN [6]. In general, higher level homomorphic schemes involve more complex computations than AHE schemes. The solution proposed in [7] modifies the BGN scheme, and the resulting scheme allows only a small message space, which is a problem since the secrets generated by the pseudorandom generator are very large and require expensive decryption (solving a discrete logarithm problem).

We propose an alternative extension for LabHE that achieves the evaluation of degree-d polynomials over encrypted data. The advantage of our method is that the online computations and communication are replaced by offline computations and communication. However, offline communication will not be polynomial in d.

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

$$\begin{aligned} m_1m_2m_3 - b_1b_2b_3 &= (m_1 - b_1)(m_2 - b_2)(m_3 - b_3)+ \\ &+ (m_1 - b_1)b_2b_3 + (m_2 - b_2)b_1b_3 + (m_3 - b_3)b_1b_2+ \\ &+ (m_1 - b_1)(m_2 - b_2)b_3 + (m_1 - b_1)(m_3 - b_3)b_2+ \\ &+ (m_2 - b_2)(m_3 - b_3)b_1. \end{aligned}$$

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

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

We denote by $\tilde{g}(b)$ the vector of all $\tilde{g}_k(b)$ corresponding to all monomials of order k in f for $2 < k < d$.

Then, we can overload the primitive Eval to compute admissible functions f that consist of multivariate polynomials of degree d on encrypted data and denote it Eval$_2$:

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

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

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

- Add(C_1, C_2): Same as before.
- cMlt(c, C'): Same as before.

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

Theorem 1. *The* eLabHE *scheme is correct.* ◇

Theorem 2. *The* eLabHE *scheme is semantically secure.* ◇

Theorem 3. *The* eLabHE *scheme is context-hiding.* ◇

The proofs are given in Appendix A.

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

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

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

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

If we use the extension of LabHE proposed in [7], which uses a level-$(d-1)$ homomorphic scheme, E has to perform $2^d - 1$ encrypted additions, $2^d - 2$ plaintext-ciphertext multiplications and $2^d - d - 2$ encrypted multiplications and send to D the final result in $\hat{C}$ online. However, the decryption that D is required to perform is in the level-$(d-1)$ homomorphic scheme and has substantially high complexity.

4 ENCRYPTED EXECUTION OF LQG

4.1 Encrypted LQG with public system model

If the state matrices A, B, C, the noise covariances W, V, the costs Q, R can be public, the setup sends them in plaintext to the cloud controller. Then, the cloud can compute the Kalman gain L and feedback gain K as in (5). The sub-systems encrypt with the public key of the actuator their measurements and desired reference, along with the initial state, and send them to the cloud controller. Using only the additively homomorphic property of the encryption scheme, the cloud can compute locally, at each time step, the encrypted control input u_k as in (6). After this computation, the cloud sends the encrypted control input to the actuator, which decrypts it.

$$\begin{aligned} \mathrm{E}(u_k) &= \mathrm{Add}(\mathrm{cMlt}(-K, \mathrm{Add}(\mathrm{E}(\hat{x}_k), \mathrm{E}(-x_r))), \mathrm{E}(u_r)) \\ \mathrm{E}(\hat{x}_{k+1}) &= \mathrm{Add}(\mathrm{cMlt}(L, \mathrm{E}(z_{k+1})), \mathrm{cMlt}(I - LC, \\ &\quad \mathrm{Add}(\mathrm{cMlt}(A, \mathrm{E}(\hat{x}_k)), \mathrm{cMlt}(B, \mathrm{E}(u_k)))))). \end{aligned} \tag{6}$$

This case is linear in the encrypted data and AHE is sufficient to ensure privacy of the measurements, states, reference and control inputs. This setup where the state matrices are public is considered, for example, in [12].

4.2 Encrypted LQG with private system model

As justified in the Introduction, in many situations it is important to protect not only the signals (e.g., the states, measurements), but also the system model. To this end, we propose a solution that uses Labeled Homomorphic Encryption to achieve encrypted multiplications and the private execution of LQG on encrypted data. LabHE has a useful property called unbalanced efficiency that can be observed from Section 3 and was described in [8], which states that only one server is required to perform operations on ciphertexts, while the decryptor performs computations only on the plaintext messages. We will employ this property by having the cloud perform the more complex operations and the actuator the more efficient ones.

In Figure 3, the actuator holds the LQG query, denoted by f_{LQG}, which describes the functionality of LQG. Offline, the actuator generates a pair of master keys, as described in Section 3.2 and distributes the master public key to the rest of the parties. The setup and sub-systems generate their secret keys and send the corresponding public keys to the actuator. Still offline, these parties generate the labels corresponding to their data with respect to the time stamp and the size of the data. As explained in Section 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 steps and sizes corresponding

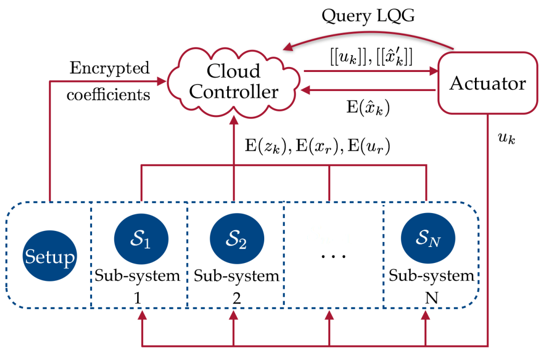

Figure 3: The setup and sub-systems send their encrypted data to the cloud. The cloud has to run the LQG algorithm on the private measurements and the system's private matrices and send the encrypted result to the actuator. The latter then actuates the system with the decrypted inputs it received.

to each signal, with no other complex synchronization process necessary between the actors. This is shown in Protocol 1.

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

The setup sends the LabHE encryptions of the state matrices and control gains to the cloud controller, once, before the execution begins. The sub-systems send the encryptions of their initial states and desired reference to the cloud controller, also once, at the onset of the execution. Then, at every time step, the sub-systems encrypt their measurements and send them to the cloud. After the cloud performs the encrypted LQG query for one time step, it sends the encrypted control input at the current time step to the actuator, which decrypts it and inputs it to the system. In Protocol 2, we describe how the encrypted LQG query is performed by the parties, which involves the actuator sending an encryption of the processed result back to the cloud such that the computation can continue in the future time steps.

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

$$\begin{aligned} \Gamma_1 &:= (I - LC)(A - BK), \\ \Gamma_2 &:= (I - LC)BK, \quad \Gamma_3 := (I - LC)B, \end{aligned} \tag{7}$$

then, the multiplication depth in terms of full LabHE ciphertexts for the state estimate at iteration 1 is one and for the control input is two. We assume for the moment that the cloud has access to $\mathrm{E}(\Gamma_1), \mathrm{E}(\Gamma_2), \mathrm{E}(\Gamma_3)$ and discuss how to achieve these products in Section 5.

For the subsequent time steps, $k \geq 1$, Protocol 2 ensures that the actuator receives the control input at iteration k, corresponding to the program $\mathcal{P}_k$. For the computation of the state estimation at time $k+1$, the cloud needs to have the full LabHE ciphertext of $\mathrm{E}(\hat{x}_k) \in \mathcal{M} \times \hat{\mathcal{C}}$, but as a result of the polynomial evaluations at time step k, it has the AHE ciphertext $[[\hat{x}_k]]$. To privately refresh the encryption, which happens in lines 2–4 of Protocol 2, the cloud uses a one-time pad r_k, as described in Section 3.1.2, and sends $[[\hat{x}'_k - r_k]]$ to the actuator, which is possible since the scheme is additively homomorphic. The actuator calls the decryption primitive and sends back $\mathrm{E}(\hat{x}_k - r_k)$, from which the cloud retrieves $\mathrm{E}(\hat{x}_k)$.

Protocol 1: Initialization of LQG

```
Input: Actuator: $f_{LQG}$; Sub-systems: $x_0, x_r, u_r$; Setup: $K, L, \Gamma_1, \Gamma_2, \Gamma_3$.
Output: Actuator: $u_0$; Cloud: $\mathrm{E}(x_0), \mathrm{E}(K), \mathrm{E}(L), \mathrm{E}(\Gamma_1), \mathrm{E}(\Gamma_2), \mathrm{E}(\Gamma_3)$,
    $[[\tilde{u}_r]], [[\hat{x}_\Gamma]]$.
    Offline:
 1: Actuator: Generate $(\mathrm{mpk}, \mathrm{msk}) \leftarrow \mathrm{Init}(1^\lambda)$ and distribute mpk
    to the others.
 2: Sub-systems, Setup: Get $(\mathrm{usk}, \mathrm{upk}) \leftarrow \mathrm{KeyGen}(\mathrm{mpk})$ and send
    upk to the actuator.
 3: Sub-systems, Setup, Actuator: Allocate labels to the inputs of
    function $f_{LQG}$ $\tau_1, \ldots, \tau_t$ as follows:
    Sub-system $i$: for each measurement at time $k \in \{0, \ldots, T-1\}$,
    $z^i_k$ of size $p^i$, where $i$ denotes a sub-system, generate the corre-
    sponding labels $\tau_{z^i_k} = [kp^i \quad kp^i+1 \quad \ldots \quad (k+1)p^i - 1]^\intercal$;
    then similarly for $x^i_0, x^i_r, u^i_r$ with the labels starting from where
    the previous signals ended.
    Setup: for matrix $K \in \mathbb{R}^{m \times n}$, set $l = 0$, generate $\tau_K = \begin{bmatrix} l & l+1 & \ldots & l+n-1 \\ \vdots & & & \vdots \\ l+(m-1)n & l+(m-1)n+1 & \ldots & l+mn-1 \end{bmatrix}$ and update $l = mn$, then
    follow the same steps for the rest of the matrices, starting from
    $l$ and updating it.
    Actuator: follow the same steps as the sub-systems and setup,
    and then generate similar labels for the state estimates $\hat{x}_k$ start-
    ing from the last $l$.
 4: Sub-systems, Setup, Actuator: Perform the offline part of the
    LabHE encryption primitive and decryption for the actuator.
 5: Cloud: Generate randomness for Protocol 2.
 6: Actuator: Form the program $\mathcal{P} = (f_{LQG}, \tau_1, \ldots, \tau_t)$.
    Initialization:
 7: Setup: Perform the online part of LabHE encryption and send
    to the cloud: $\mathrm{E}(\Gamma_1), \mathrm{E}(\Gamma_2), \mathrm{E}(\Gamma_3), \mathrm{E}(K), \mathrm{E}(L)$.
 8: Sub-systems: Perform the online part of LabHE encryption and
    send to the cloud: $\mathrm{E}(x_0), \mathrm{E}(x_r), \mathrm{E}(u_r)$.
 9: Cloud: Compute $[[\tilde{u}_r]] \leftarrow \mathrm{Add}(\mathrm{Mlt}(\mathrm{E}(K), \mathrm{E}(x_r)), \mathrm{E}(u_r))$;
    $[[\hat{x}_\Gamma]] \leftarrow \mathrm{Add}(\mathrm{Mlt}(\mathrm{E}(\Gamma_2), \mathrm{E}(x_r)), \mathrm{Mlt}(\mathrm{E}(\Gamma_3), \mathrm{E}(u_r)))$.
    Online:
10: Cloud: $[[u_0]] \leftarrow \mathrm{Add}(\mathrm{Mlt}(\mathrm{E}(-K), \mathrm{E}(\hat{x}_0)), [[\tilde{u}_r]])$.
11: Cloud: Send to the actuator $[[u_0]]$.
12: Actuator: Decrypt $u_0$.
```

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

Protocol 2: Encrypted LQG at time step k

```
Input: Actuator: msk, mpk, $\mathcal{P}_k$; Cloud: $\mathrm{E}(z_k), \mathrm{E}(\hat{x}_{k-1}), \mathrm{E}(\Gamma_1), \mathrm{E}(L)$,
    $\mathrm{E}(K), [[\tilde{u}_r]], [[\hat{x}_\Gamma]]$.
Output: Actuator: $u_k$; Cloud: $\mathrm{E}(\hat{x}_k)$.
    Online:
1: Cloud: Compute $[[\hat{x}_k]] \leftarrow \mathrm{Add}(\mathrm{Mlt}(\mathrm{E}(\Gamma_1), \mathrm{E}(\hat{x}_{k-1})), \mathrm{Mlt}(\mathrm{E}(L),$
    $\mathrm{E}(z_k)), [[\hat{x}_\Gamma]])$.
2: Cloud: Send to the actuator $[[\hat{x}'_k]] \leftarrow \mathrm{Add}([[\hat{x}_k]], [[-r_k]])$,
    where $r_k \xleftarrow{\$} \mathcal{M}^n$ is a random vector.
3: Actuator: Decrypt $[[\hat{x}_k - r_k]]$ and send back a LabHE encryp-
    tion of $\mathrm{E}(\hat{x}_k - r_k)$.
4: Cloud: $\mathrm{E}(\hat{x}_k) \leftarrow \mathrm{Add}(\mathrm{E}(\hat{x}_k - r_k), r_k)$.
5: Cloud: $[[u_k]] \leftarrow \mathrm{Add}(\mathrm{Mlt}(\mathrm{E}(-K), \mathrm{E}(\hat{x}_k)), [[\tilde{u}_r]])$.
6: Cloud: Send to the actuator $[[u_k]]$ and output $\mathrm{E}(\hat{x}_k)$.
7: Actuator: Decrypt $u_k$.
```

Protocol 3: Encrypted LQG

```
Input: Actuator: $f_{LQG}$; Sub-systems: $x_0$; Setup: $\Gamma_1, \Gamma_2, \Gamma_3, K, L$.
Output: Actuator: $\{u_k\}_{k=0,\ldots,T-1}$.
    Offline + Initialization:
1: Sub-systems, Setup, Cloud and Actuator: Run Protocol 1.
    Online:
2: for k=1,...,T-1 do
3:     Cloud and Actuator: Run Protocol 2.
4:     Actuator: Input $u_k$ to the system.
5:     Sub-systems: Measure $z_{k+1}$, encrypt and send it to the cloud.
6: end for
```

As can be seen from Protocols 1 and 2, the actuator does not need to know the system matrices in order to compute the program and decrypt the results, but only their labels.

Informally, two-party privacy with respect to semi-honest parties is achieved if neither party can infer anything about the private data of the other party from processing its data and messages received, other than what it can infer from its inputs and outputs. Multi-party privacy is achieved if the same holds for a coalition of parties agains the non-colluding parties. A formal definition is given in Appendix B.

Theorem 4. *Protocol 3 achieves privacy of the encrypted LQG with respect to semi-honest parties.* ◇

Let us now consider possible coalitions between the parties shown in Figure 3. The cloud and the actuator are not allowed to collude, since the cloud has all the data in the system encrypted with the actuator's master key. Furthermore, all the sub-systems and setup cannot be corrupted by an adversary at the same time. Under these assumptions, the following theorem holds:

Theorem 5. *Protocol 3 achieves privacy of the encrypted LQG with respect to collusions.* ◇

The privacy of Protocol 3, composed of Protocol 1 and $T-1$ runs of Protocol 2 follows from the semantic security and context-hiding property of LabHE and the perfect secrecy of the one-time pad. The proofs are given in Appendix B.

5 ENCRYPTED COMPUTATION OF LQG COEFFICIENTS

The setup party does not need to be online for the computation of LQG. For a given system, the encryptions of the system model and gains $\mathrm{E}(A), \mathrm{E}(B), \mathrm{E}(C), \mathrm{E}(K), \mathrm{E}(L)$ might have been given offline to the cloud or distributed among the sub-systems. In such cases, the encryptions of $\Gamma_1, \Gamma_2, \Gamma_3$ or other polynomial functions of these matrices are not available, but they can be computed at the beginning of the protocol by the cloud and actuator as the output of a program evaluating a degree-3 polynomial, using eLabHE, and stored as $\mathrm{E}(\Gamma_1), \mathrm{E}(\Gamma_2), \mathrm{E}(\Gamma_3)$. Notice from (7) that:

$$\Gamma_3 = B - LCB, \quad \Gamma_2 = \Gamma_3 K, \quad \Gamma_1 = A - LCA + \Gamma_2. \tag{8}$$

Then, the cloud need to compute two products of three encrypted matrices and one product of two encrypted matrices.

Protocol 4: Initialization of LQG, extended version

```
Input: Actuator: $f_{LQG}, f_{\Gamma_1}, f_{\Gamma_2}, f_{\Gamma_3}$; Sub-systems: $x_0, x_r, u_r$; Setup: $A, B, C, K, L$.
Output: Actuator: $u_0$; Cloud: $\mathrm{E}(x_0), \mathrm{E}(K), \mathrm{E}(L), \mathrm{E}(\Gamma_1), \mathrm{E}(\Gamma_2), \mathrm{E}(\Gamma_3)$, $[[\tilde{u}_r]]$, $[[\hat{x}_\Gamma]]$.
    Offline:
1:  Perform lines 1–4 from Protocol 1.
2:  Actuator: Form the programs $\mathcal{P} = (f_{LQG}, \tau_1, \dots, \tau_t)$, $\mathcal{P}_{\Gamma_3} = (f_{\Gamma_3}, \tau_B, \tau_C, \tau_L, \tau_K)$, $\mathcal{P}_{\Gamma_2} = (f_{\Gamma_2}, \tau_{\Gamma_3}, \tau_K)$ and $\mathcal{P}_{\Gamma_1} = (f_{\Gamma_1}, \tau_A, \tau_C, \tau_L, \tau_K, \tau_{\Gamma_3})$. Compute $\tilde{f_i} \leftarrow \mathrm{Eval}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \mathcal{P}_{\Gamma_i})$, for $i = 1, 2, 3$, corresponding to (8) and send it to the cloud. Create labels $\tau_{\Gamma_i}$ for refreshing $\Gamma_i$.
3:  Cloud: Generate randomness for Protocol 2 and Initialization.
    Initialization:
4:  Setup: Encrypt the matrices and send them to the cloud: $\mathrm{E}(A)$, $\mathrm{E}(B), \mathrm{E}(C), \mathrm{E}(K), \mathrm{E}(L)$.
5:  Cloud: Compute $[[\Gamma_3]] \leftarrow \mathrm{Eval}_2(\mathrm{mpk}, \tilde{f_3}, \mathrm{E}(B), \mathrm{E}(C), \mathrm{E}(L))$, share it and send it to the actuator a share $[[\Gamma_3']]$.
6:  Actuator: Decrypt $\Gamma_3'$, allocate it the label $\tau_{\Gamma_3}$ and send the LabHE encryption to the cloud: $\mathrm{E}(\Gamma_3')$.
7:  Cloud: Reconstruct and obtain $\mathrm{E}(\Gamma_3)$.
8:  Cloud: Compute $[[\Gamma_2]] \leftarrow \mathrm{Eval}_2(\mathrm{mpk}, \tilde{f_2}, \mathrm{E}(\Gamma_3), \mathrm{E}(K))$, $[[\Gamma_1]] \leftarrow \mathrm{Eval}_2(\mathrm{mpk}, \tilde{f_1}, \mathrm{E}(A), \mathrm{E}(C), \mathrm{E}(L), \mathrm{E}(K), \mathrm{E}(\Gamma_3))$, share them and send to the actuator the shares $[[\Gamma_2']], [[\Gamma_1']]$.
9:  Actuator: Decrypt $\Gamma_i$, allocate it the label $\tau_{\Gamma_i}$ and send the LabHE encryptions to the cloud: $\mathrm{E}(\Gamma_i')$, for i = 1,2.
10: Cloud: Reconstruct and obtain $\mathrm{E}(\Gamma_1), \mathrm{E}(\Gamma_2)$.
11: The rest follows as in lines 7–12 in Protocol 1.
```

The privacy of Protocol 4 follows from the context-hiding and semantic security of the eLabHE scheme described in Section 3.3 and perfect privacy of the one-time pads used.

Furthermore, we can modify Protocol 2 such that the cloud sends the control input to the actuator before requesting a refreshed encryption of the state estimate. Specifically, the cloud can compute first, in an encrypted fashion:

$$u_k = -K\Gamma_1 \hat{x}_{k-1} - KLz_k - K(\Gamma_2 x_r + \Gamma_3 u_r) + Kx_r + u_r,$$

and then follow with computing the state estimate $\hat{x}_k$. Such a change is recommended when timing is crucial, because it allows the actuator to send the control input to the plant faster, at the initial expense of four extra encrypted multiplication $\mathrm{E}(K\Gamma_1)$, $(K\Gamma_2)$, $(K\Gamma_3)$, $\mathrm{E}(KL)$ that can be performed in Protocol 4.

6 NUMERICAL RESULTS

6.1 Implementation details

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

Most public space AHE schemes have the ring of integers $\mathbb{Z}_N$ as message space, where N is an RSA modulus. To prevent brute force factorization, N is required to be at least 1024 bits. Hence, the message space is large enough to represent messages with precision of 128 bits, which is the standard quadruple precision. In this case, the quantization and round-off errors can be considered negligible.

Under stability conditions of the quantized matrices, the stability of quantized Kalman Filter can be proved [30]. The quantization effects of encryption over control performance are analyzed in [1, 21, 27, 28].

We assume the channels are reliable and the packets cannot be tampered with. The memory of the parties is finite, so we cannot consider that Protocol 3 runs for an infinite time. In the offline phase, the labels are generated for a fixed number of time steps T. Once these T time steps elapse, the parties have to generate new labels for the signals (not for the state matrices and gains, if those are not desired to be changed). This can be done in parallel with the computations that take place in the first T time steps.

6.2 Case study

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

We consider the data from the HAMLab ISE model in [33], available at [32], which models the building as one zone. We use a modified model that considers the building to be split into two zones, which we assume have two different owners, who want different set points for the temperature in the zones, as well as privacy with respect to their presence there and desired settings. The state consists of the temperatures for each of the two zones (indoors air, floor, separating wall, internal facade and external facade temperature), the control input represents the heating/cooling for the two zones and the disturbances consist of the outdoors air temperature, the occupancy generated heat and the heat caused by the sun through the windows for the two zones. The measurements are considered to be the same as the states. The system has state dimension $n = 10$ and control input dimension $m = 2$. We simulate the results for a sampling time of 420 seconds. Assume the states corresponding to the indoors air temperature for the two zones have to be steered to 15°C and 25°C during the day (considered between 6 am to 8 pm), and to 10°C and 20°C during the night.

Each zone has a local device that generates secret keys for encrypting the labels. The matrices corresponding to the state model and the corresponding Kalman and control gains have been encrypted with labels generated by a different secret key by another machine in the building, called setup. The actuator generates a pair of master key and secret key and sends the master key to the three parties described so far, which encrypt their secret keys with the master key and send them to the actuator. The local devices for the two zones send the corresponding encrypted measurements and encrypted references to the cloud controller. If the cloud readily receives all the coefficients encrypted from the Setup, it executes Protocol 1, and otherwise, it executes Protocol 4 to obtain full LabHE encryptions of the gains and rest of the coefficients. The cloud then computes the estimate of the current state, refreshes the encryption with the help of the actuator, then computes the control input and sends it to the actuator, as described in Protocol 2.

We instantiated the AHE scheme with the Paillier scheme [25], for which we chose a modulus N of 1024 bits. The pseudorandom generator is chosen to be the SHA-3 hash function with 224 output bits. The protocols were run on a standard MacBook Pro laptop with a 2.2 GHz Intel Core i7 and 16 GB of RAM.

Figure 4 shows the performance of estimation and tracking for a fixed-point representation with $l_i = 24$ integer bits and $l_f = 24$ fractional bits, where r represents the set point, T^i_{in} is the true temperature in zone i and $\hat{T}^i_{in}$ is the temperature estimate.

For a simulation of $T = 100$ time steps, corresponding roughly to 12 hours with the chosen sampling period, the offline execution times corresponding to the Protocol 1 and 4 and online execution times are shown in Table 1. The online time corresponding to computing the estimate and control input in a time step are under 0.3 seconds for all actors. Since the sampling times for buildings are usually large, (other examples for the same HAMLab ISE model include 300 seconds in [18] and 444 seconds in [11]), the encrypted computations finish much before a new measurement is acquired.

The execution times for varying system dimensions, considering one agent with all the states and control inputs for ease of comparison, are presented in Figure 5 and Figure 6. Figure 5 compares which actor performs the most computational intense task for each

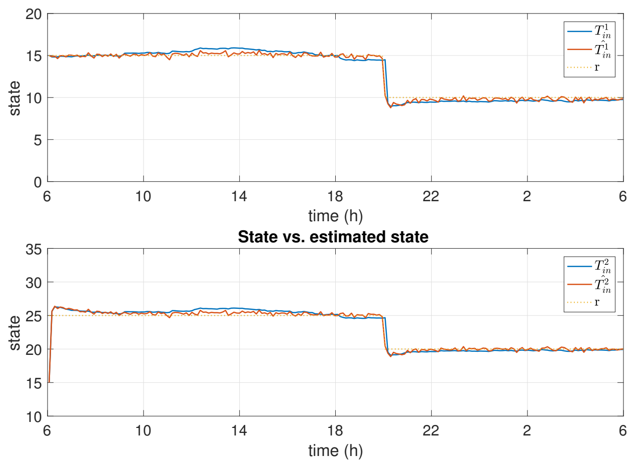

Figure 4: Performance of the estimation and tracking for encrypted LQG implemented with $l_i = 24$, $l_f = 24$.

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

Table 1: Average times for the encrypted LQG computation for $l_i = 24$, $l_f = 24$, $N_\sigma = 1024$, 224-bit secrets, 100 time steps.

phase of Protocol 3, while Figure 6 shows how much time each actor spends on each phase using a logarithmic scale for visualization purposes. In order to evaluate the time performance of the proposed protocols, one should look at the online times for the sub-systems and actuator, since these are the most resource-sensitive actors. We observe that even for larger systems of 100 states and 20 inputs, the online time for one iteration for the agent is 0.0005 seconds and 0.27580 seconds for the actuator, which are competitive times. Notice that in the initialization phase and in the online iterations, the cloud has the most computational requirements, which is as desired, since the cloud is a powerful server, and it has the necessary resources to improve the times. On the other hand, the rest of the parties are expected to perform more offline computations than the cloud, which correspond to the label generation and LabHE encryption process. The actuator is not required to do anything in the initialization phase corresponding to Protocol 1 and the setup does not play a role in the online iterations.

For larger system dimensions ($n \geq 50$), the offline and initialization phases become computationally and memory intensive for the execution of Protocol 4, due to computing $\Gamma_1, \Gamma_2, \Gamma_3$ as products of encrypted matrices. As described in Section 3.3, the actuator is required to compute the encryptions of products of the monomials in the polynomial evaluations and transmit them to the cloud, which can take 4 hours on a 2.2 GHz Intel Core i7 processor. However, the online execution times remain the same as in Figures 5 and 6.

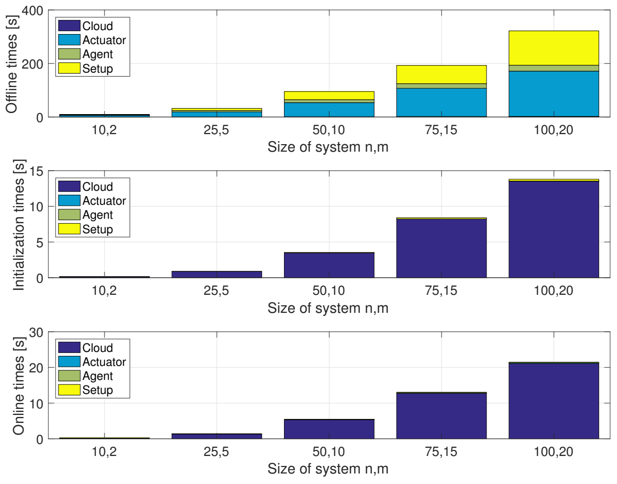

Figure 5: Execution times for Protocol 3 for different system dimensions, with $l_i = 24$, $l_f = 24$, $N_\sigma = 1024$, 224-bit secrets, and 100 time steps.

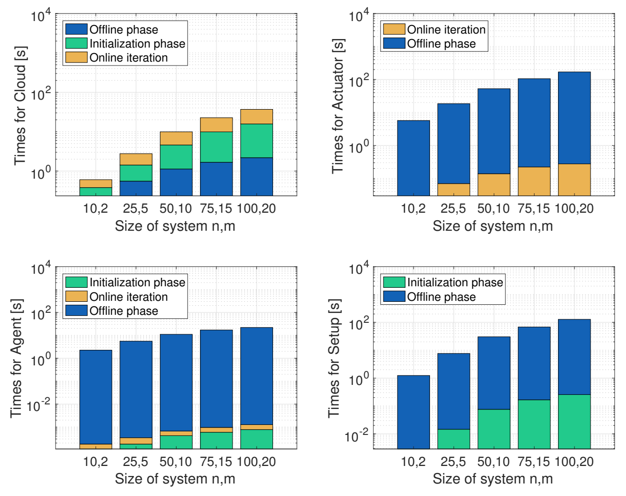

Figure 6: Execution times on a logarithmic scale for Protocol 3 for different system dimensions, with $l_i = 24$, $l_f = 24$, $N_\sigma = 1024$, 224-bit secrets, and 100 time steps.

7 FUTURE WORK

Computing the Kalman and feedback gains when the system matrices are encrypted involves multiplications and divisions between encrypted data, as well as comparisons and matrix inversions, which have high computational complexity. In Section 4, we assumed that the cloud is given the encryptions of the feedback gains and state matrices by a different party, which knows the system parameters. As those parameters do not depend on the measurements, they can be stored and the gains precomputed offline. If the existence of such a party cannot be guaranteed, we can use encrypted communication between the cloud and a second server, such that the cloud obtains the encrypted gains through secure multiparty computation. We will address the fully encrypted design of LQG in future work.

ACKNOWLEDGMENTS

This work was supported in part by ONR N00014-17-1-2012, and by NSF CNS-1505799 grant and the Intel-NSF Partnership for Cyber-Physical Systems Security and Privacy.

REFERENCES

[1] Andreea B Alexandru, Manfred Morari, and George J Pappas. 2018. Cloud-Based MPC with Encrypted Data. In *57th Conference on Decision and Control (CDC)*. IEEE, Miami Beach, FL, USA, 5014–5019.
[2] Mikhail Aristov, Benjamin Noack, Uwe D Hanebeck, and Jörn Müller-Quade. 2018. Encrypted Multisensor Information Filtering. In *21st International Conference on Information Fusion*. IEEE, Cambridge, UK, 1631–1637.
[3] Michael Athans. 1972. The discrete time Linear-Quadratic-Gaussian stochastic control problem. In *Annals of Economic and Social Measurement*. Vol. 1, no. 4. NBER, New York, NY, USA, 449–491.
[4] Manuel Barbosa, Dario Catalano, and Dario Fiore. 2017. Labeled Homomorphic Encryption. In *European Symposium on Research in Computer Security*. Springer, Oslo, Norway, 146–166.
[5] Dimitri P Bertsekas. 2012. *Dynamic programming and optimal control*. Vol. 1. Athena scientific, Belmont, MA, USA.
[6] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography*, Vol. 3378. Springer, Cambridge, MA, USA, 325–341.
[7] Dario Catalano and Dario Fiore. 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.
[8] Dario Catalano and Dario Fiore. 2015. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, Denver, CO, USA, 1518–1529.
[9] Ronald Cramer, Ivan B Damgård, and Jesper B Nielsen. 2015. *Secure multiparty computation*. Cambridge University Press, New York, NY, USA.
[10] Ivan B Damgård and Mads Jurik. 2001. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*. Springer, Paris, France, 119–136.
[11] Ján Drgoňa and Michal Kvasnica. 2013. Comparison of MPC strategies for building control. In *International Conference on Process Control*. IEEE, Strbske Pleso, Slovakia, 401–406.
[12] Farhad Farokhi, Iman Shames, and Nathan Batterham. 2017. Secure and private control using semi-homomorphic encryption. *Control Engineering Practice* 67 (2017), 13–20.
[13] Oded Goldreich. 2003. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, Cambridge, UK.
[14] Oded Goldreich. 2004. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, Cambridge, UK.
[15] Shafi Goldwasser and Silvio Micali. 1982. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *14th ACM symposium on Theory of Computing*. ACM, San Francisco, CA, USA, 365–377.
[16] Francisco J Gonzalez-Serrano, Adrian Amor-Martın, and Jorge Casamayon-Anton. 2014. State estimation using an extended Kalman filter with privacy-protected observed inputs. In *IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, Atlanta, GA, USA, 54–59.
[17] Matthew Hale, Austin Jones, and Kevin Leahy. 2018. Privacy in feedback: The differentially private lqg. In *American Control Conference (ACC)*. IEEE, Milwaukee, WI, USA, 3386–3391.
[18] Achin Jain, Madhur Behl, and Rahul Mangharam. 2017. Data Predictive Control for building energy management. In *American Control Conference (ACC)*. IEEE, Seattle, WA, USA, 44–49.
[19] Marc Joye and Benoît Libert. 2013. Efficient cryptosystems from 2^k-th power residue symbols. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Athens, Greece, 76–92.
[20] Junsoo Kim, Chanhwa Lee, Hyungbo Shim, Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2016. Encrypting controller using fully homomorphic encryption for security of cyber-physical systems. *IFAC-PapersOnLine* 49, 22 (2016), 175–180.
[21] Kiminao Kogiso. 2018. Upper-bound analysis of performance degradation in encrypted control system. In *American Control Conference (ACC)*. IEEE, Milwaukee, WI, USA, 1250–1255.
[22] Kiminao Kogiso and Takahiro Fujita. 2015. Cyber-security enhancement of networked control systems using homomorphic encryption. In *54th Conference on Decision and Control (CDC)*. IEEE, Osaka, Japan, 6836–6843.
[23] Jerome Le Ny and George J Pappas. 2014. Differentially private filtering. *IEEE Trans. Automat. Control* 59, 2 (2014), 341–354.
[24] Tal G Malkin. 2016. Lecture notes on Introduction to Cryptography.
[25] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Prague, Czech Republic, 223–238.
[26] Torben P Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*. Springer, Santa Barbara, CA, USA, 129–140.
[27] Moritz Schulze Darup, Adrian Redder, and Daniel E Quevedo. 2019. Encrypted Cooperative Control Based on Structured Feedback. *IEEE Control Systems Letters* 3, 1 (2019), 37–42.
[28] Moritz Schulze Darup, Adrian Redder, Iman Shames, Farhad Farokhi, and Daniel Quevedo. 2018. Towards Encrypted MPC for Linear Constrained Systems. *IEEE Control Systems Letters* 2, 2 (2018), 195–200.
[29] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[30] Shuli Sun, Jianyong Lin, Lihua Xie, and Wendong Xiao. 2007. Quantized kalman filtering. In *IEEE 22nd International Symposium on Intelligent Control*. IEEE, Singapore, Singapore, 7–12.
[31] Juan R Troncoso-Pastoriza and Fernando Pérez-González. 2011. Secure adaptive filtering. *IEEE Transactions on Information Forensics and Security* 6, 2 (2011), 469–485.
[32] Adrianus W M van Schijndel. 2007. HAMLab (Heat, Air and Moisture Laboratory). https://archbpswiki.bwk.tue.nl/bpswiki/index.php/Hamlab.
[33] Adrianus W M van Schijndel. 2007. *Integrated heat air and moisture modeling and simulation*. Ph.D. Dissertation. T. U. Eindhoven.
[34] Zheng Yan, Peng Zhang, and Athanasios V Vasilakos. 2014. A survey on trust management for Internet of Things. *Journal of network and computer applications* 42 (2014), 120–134.
[35] Zhenyong Zhang, Junfeng Wu, David Yau, Peng Cheng, and Jiming Chen. 2018. Secure Kalman Filter State Estimation by Partially Homomorphic Encryption. In *9th ACM/IEEE International Conference on Cyber-Physical Systems*. IEEE, Porto, Portugal, 345–346.

A FURTHER DETAILS ON LabHE

For two random variables X, Y over a finite set U, the statistical distance is defined as follows:

$$\mathrm{SD}[X,Y] := 1/2 \sum_{u \in U} \Big|\Pr[X=u] - \Pr[Y=u]\Big|.$$

A function $\mathrm{negl}(\lambda)$ is said to be *negligible* if it vanishes faster than the inverse of any polynomial in λ. A simulator Sim is a probabilistic polynomial-time (PPT) algorithm.

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

The definitions of correctness, semantic security, circuit privacy and context-hiding are taken from [4, 7].

Definition 3. *(Correctness) A multi-user Labeled Homomorphic Encryption scheme correctly evaluates a family of functions $\mathcal{F}$ if for all honestly generated keys* $(\mathrm{mpk}, \mathrm{msk}) \xleftarrow{\$} \mathrm{Init}(1^\lambda)$*, all user keys* $(\mathrm{upk_k}, \mathrm{usk_k}) \xleftarrow{\$} \mathrm{KeyGen}(\mathrm{mpk})$*, $k \in [l]$, for all $f \in \mathcal{F}$, all labels $\tau_1, \dots, \tau_t \in \mathcal{L}$, messages $m_1, \dots, m_t \in \mathcal{M}$, any $C_i \leftarrow$* $\mathrm{E}(\mathrm{mpk}, \mathrm{usk}_{j_i}, \tau_i, m_i)$ *and* $\tilde{g} \leftarrow (f, \mathrm{Eval}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, (f, \tau_1, \dots, \tau_t)))$*, $\forall i \in [t], j_i \in [l]$:*

$$\Pr\Big[\mathrm{D}(\mathrm{msk},\mathrm{upk}, \mathcal{P}, \mathrm{Eval}_2(\mathrm{mpk}, \tilde{g}, C_1, \dots, C_t)) = f(m_1, \dots, m_t)\Big] > \\ > 1 - \mathrm{negl}(\lambda),$$

where the probability is taken over the random choices. ◇

Proof. (Theorem 1) The proof can be easily extended from the proof of correctness of LabHE. The correctness of the eLabHE scheme holds as long as $f(m_1, \dots, m_t) \in \mathcal{M}$ and $\prod_{i \in [d]} b_i \in \mathcal{M}$, where d is the degree of f. Consider ciphertexts produced by the E primitive. For every $(\tau, m) \in \mathcal{L} \times \mathcal{M}$, the encryption primitive yields $\mathrm{E}(\mathrm{mpk}, \mathrm{upk}, \tau, m) \to C = (a, \beta)$, where $a = m - f_{id}(F(\mathrm{usk}, \tau))$ and it follows by the correctness of AHE that $m \leftarrow \mathrm{D}(\mathrm{msk}, \mathcal{I}_\tau, C)$.

Consider ciphertexts produced by Eval_2. For $i \in [t]$, consider any t labeled programs $\mathcal{P}_i = \left(f_i, \tau_1^{(i)}, \dots, \tau_{t_i}^{(i)}\right)$ and t ciphertexts C_i such that $m_i \leftarrow \mathrm{D}(msk, usk, \mathcal{P}_i, C_i)$. Then, for any $f \in \mathcal{F}$, we want to evaluate $f(\mathcal{P}_1, \dots, \mathcal{P}_t) =: \mathcal{P}^*$ on ciphertexts $C_1, \dots, C_t$. For that we run Eval_1 and obtain $\tilde{g} \leftarrow \Big(f, \mathrm{Eval}_1\Big(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \Big\{\tau_1^{(i_j)}, \dots, \tau_{t_{i_j}}^{(i_j)}\Big\}_{j \in [t]}\Big)\Big)$. Denote the resulting ciphertext as $C \leftarrow \mathrm{Eval}_2(\mathrm{mpk}, \tilde{g}, C_1, \dots, C_t)$. By construction, a ciphertext C_i is either $(m_i - b_i, [[b_i]])$ or $[[m_i - b_i]]$, with $b_i \leftarrow f_i\Big(F\Big(\mathrm{usk}, \tau_1^{(i)}\Big), \dots, F\Big(\mathrm{usk}, \tau_{t_i}^{(i)}\Big)\Big)$, obtained from Eval_2 applied to the inputs of $\mathcal{P}_i$ and the corresponding $\tilde{f}_i \leftarrow \Big(f_i, \mathrm{Eval}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \tau_1^{(i)}, \dots, \tau_{t_i}^{(i)})\Big)$, $i \in [t]$. It is clear that $\tilde{g}$ will contain $\tilde{f}_i$ for $i \in [t]$. After the evaluation of f, we obtain the ciphertext C that is either $(f(m_1, \dots, m_t) - f(b_1, \dots, b_t), [[f(b_1, \dots, b_t)]])$ or $[[f(m_1, \dots, m_t) - f(b_1, \dots, b_t)]]$. Then, by construction of D, correctness of AHE and using $\mathcal{P}^* = \Big(f_1\Big(\mathcal{I}_{\tau_1^{(1)}}, \dots, \mathcal{I}_{\tau_{t_1}^{(1)}}\Big), \dots, f_t\Big(\mathcal{I}_{\tau_1^{(t)}}, \dots, \mathcal{I}_{\tau_{t_t}^{(t)}}\Big)\Big)$, we obtain the correctness of the eLabHE scheme, i.e., $\mathrm{D}(\mathrm{msk}, \mathrm{usk}, \mathcal{P}^*, C) \leftarrow f\Big(m_1, \dots, m_t\Big)$. □

The definition of semantic security [15], [14, Ch. 5] is adapted for labeled homomorphic encryption schemes in [4]:

Definition 4. *(Semantic security) Let* eLabHE = (Init, KeyGen, E, Eval_1, Eval_2, D) *be a multi-user labeled homomorphic encryption scheme and $\mathcal{A}$ be a PPT adversary. Consider the following experiment where $\mathcal{A}$ is given access to an oracle* $\mathrm{E}(\mathrm{mpk}, \mathrm{usk}, \cdot, \cdot)$*, for* $\mathrm{usk} = (\mathrm{usk_1}, \dots, \mathrm{usk_l})$ *that on input a pair (τ, m) outputs* $\mathrm{E}(\mathrm{mpk}, \mathrm{usk}, \tau, m)$*:*

$$\mathbf{Exp}_{\mathrm{eLabHE}, \mathcal{A}}(\lambda) : b \xleftarrow{\$} \{0,1\}; (\mathrm{mpk}, \mathrm{msk}) \xleftarrow{\$} \mathrm{Init}(1^\lambda)$$
$$(\mathrm{upk}, \mathrm{usk}) \xleftarrow{\$} \mathrm{KeyGen}(\mathrm{mpk})$$
$$(m_0, \tau_0^*, m_1, \tau_1^*) \xleftarrow{\$} \mathcal{A}^{\mathrm{E}(\mathrm{mpk}, \mathrm{usk}, \cdot, \cdot)}(\mathrm{mpk}, \mathrm{upk})$$
$$C \xleftarrow{\$} \mathrm{E}(\mathrm{mpk}, \mathrm{usk}, \tau_b^*, m_b)$$
$$b' \leftarrow \mathcal{A}^{\mathrm{E}(\mathrm{mpk}, \mathrm{usk}, \cdot, \cdot)}(C)$$
$$\textit{If } b' = b \textit{ return 1. Else, return 0.}$$

We say $\mathcal{A}$ is a legitimate adversary if it queries the encryption oracle on distinct labels (each label τ is never queried more than once) and never on the two challenge labels τ_0^*, τ_1^*. We define the adversary's advantage as $\mathrm{Adv}_{\mathrm{eLabHE}, \mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}_{\mathrm{eLabHE}, \mathcal{A}}(\lambda) = 1] - \frac{1}{2}$*. Then, we say that* eLabHE *has semantic security if for any PPT legitimate algorithm $\mathcal{A}$, the following holds* $\mathrm{Adv}_{\mathrm{eLabHE}, \mathcal{A}}(\lambda) = \mathrm{negl}(\lambda)$. ◇

Proof. (Theorem 2) The scheme eLabHE has the same encryption and decryption primitives as LabHE and identically looking ciphertexts. Hence, the proof follows the semantic security of the LabHE scheme, which depends on the semantic security of the underlying AHE scheme and on the pseudorandomness of F. □

Semantic security is equivalent to ciphertext indistinguishability [15], [14, Ch. 4], so we can write it as:

$$\mathrm{SD}[\mathrm{E}(\mathrm{mpk}, \mathrm{usk}, \tau, m), \mathrm{Sim}(1^\lambda, \mathrm{mpk}, \mathrm{usk})] \le \mathrm{negl}(\lambda),$$

where Sim is a PPT simulator that simply outputs a LabHE (or AHE) encryption of zeros. The same can be written for the secret sharing scheme which has perfect secrecy, with Sim outputting a random value sampled from $\mathcal{M}$.

Definition 5. *(Context-hiding) A multi-user labeled homomorphic encryption scheme satisfies context-hiding for a family of functions $\mathcal{F}$ if there exists a PPT simulator* Sim *and a negligible function* $\mathrm{negl}(\lambda)$ *such that the following holds: for any $\lambda \in \mathbb{N}$, any pair of master keys* $\mathrm{mpk}, \mathrm{msk} \xleftarrow{\$} \mathrm{Init}(1^\lambda)$*, any l user keys* $(\mathrm{upk_k}, \mathrm{usk_k}) \xleftarrow{\$} \mathrm{KeyGen}(\mathrm{mpk})$*, $k \in [l]$, any function $f \in \mathcal{F}$ with t inputs, any tuple of messages $m_1, \dots, m_t \in \mathcal{M}$, labels $\tau_1, \dots, \tau_t \in \mathcal{L}$, and ciphertexts* $C_i \xleftarrow{\$} \mathrm{E}(\mathrm{mpk}, \mathrm{usk}_{j_i}, \tau_i, m_i)$*, $i \in [t]$ and $j_i \in [l]$:*

$$\mathrm{SD}[\mathrm{Eval}_2(\mathrm{mpk}, \tilde{f}, C_1, \dots, C_t), \mathrm{Sim}(1^\lambda, \mathrm{msk}, \mathrm{upk}, \mathcal{P}, m)] \le \\ \le \mathrm{negl}(\lambda),$$

where we defined $\mathcal{P} = (f, \tau_1, \dots, \tau_t)$, $m = f(m_1, \dots, m_t)$ and $\tilde{f} = (f, \mathrm{Eval}_1(\mathrm{mpk}, \mathrm{msk}, \mathrm{upk}, \mathcal{P}))$. ◇

Context-hiding describes the property that a party that decrypts a ciphertext C as $m \leftarrow \mathrm{D}(\mathrm{msk}, \mathrm{upk}, \mathcal{P}, C)$, with $\mathcal{P} = (f, \tau_1, \dots, \tau_t)$, does not find out anything about the inputs $m_1', \dots, m_t'$, except for the fact that $m = f(m_1', \dots, m_t')$. In order to prove that eLabHE is context-hiding, we need to make use of the concept of circuit privacy.

For any admissible linear function f, we can abstract the evaluation primitives in AHE as $\hat{\text{Eval}}(\hat{\text{pk}}, f, C_1, \dots, C_t)$. We will use this notation when defining circuit privacy:

Definition 6. *(Circuit privacy) A homomorphic encryption scheme is circuit private for a family of circuits $\mathcal{F}$ if there exists a PPT simulator* Sim *and a negligible function* $\text{negl}(\lambda)$ *such that the following holds. For any λ, any pair of keys $(\hat{\text{pk}}, \hat{\text{sk}}) \xleftarrow{\$} \hat{\text{KeyGen}}(1^\lambda)$, any circuit $f \in \mathcal{F}$, any tuple of messages $m_1, \dots, m_t \in \mathcal{M}$ and $m = f(m_1, \dots, m_t)$ and ciphertexts $C_1, \dots, C_t$ satisfying $\forall i \in [t]$: $C_i \xleftarrow{\$} \hat{\text{E}}(\hat{\text{pk}}, m_i)$, then:*

$$\text{SD}\Big[\hat{\text{Eval}}(\hat{\text{pk}}, f, C_1, \dots, C_t), \hat{\text{Sim}}(1^\lambda, \hat{\text{pk}}, m)\Big] \leq \text{negl}(\lambda). \quad \diamond$$

The difference between circuit privacy and context-hiding is that in context-hiding, the decryptor has access to the function, whereas in circuit privacy, it does not.

Proof. (Theorem 3) The proof relies on the circuit privacy of the underlying AHE scheme. Let $\hat{\text{Sim}}$ be the simulator for the circuit privacy of AHE. A simulator $\text{Sim}(1^\lambda, \text{msk}, \text{upk}, (f, \tau_1, \dots, \tau_t), m)$ computes for $\text{upk} = (\text{upk}_1, \dots, \text{upk}_l)$, $\text{usk}_j \leftarrow \hat{\text{D}}(\text{msk}, \text{upk}_j)$, $j \in [l]$, and $b_i \leftarrow F(\text{usk}_{j_i}, \tau_i)$, $i \in [t]$. Then, it computes $b \leftarrow f(b_1, \dots, b_t)$. Notice that Sim has the inputs necessary to compute $\tilde{f}$. If f is a degree-1 polynomial, then Sim outputs $C = (m - b, \hat{\text{Sim}}(1^\lambda, \text{mpk}, b))$. Else, if f is degree-d, with $d \geq 2$, Sim outputs $C = \hat{\text{Sim}}(1^\lambda, \text{mpk}, m - b)$. Using the circuit privacy of AHE, the outputs of Sim are distributed identically to the corresponding outputs produced by Eval_2. □

B PRIVACY OF PROTOCOL 3

Definition 7. *(Two-party privacy w.r.t. semi-honest behavior [14, Ch. 7]) Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a functionality, and $f_1(x_1, x_2)$, $f_2(x_1, x_2)$ denote the first and second components of $f(x_1, x_2)$, for any inputs $x_1, x_2 \in \{0,1\}^*$. Let Π be a two-party protocol for computing f. The view of the i-th party ($i = 1, 2$) during an execution of Π on the inputs (x_1, x_2), denoted $V_i^\Pi(x_1, x_2)$, is $(x_i, coins, m_1, \dots, m_t)$, with the i'th party's internal coin tosses, and m_j represents the j-th message it has received. For a deterministic functionality f, we say that Π privately computes f if there exist probabilistic polynomial-time algorithms, called simulators, denoted* Sim_i*, such that:*

$$\text{SD}[\{\text{Sim}_i(1^\lambda, x_i, f_i(x_1, x_2))\}_{x_{1,2} \in \{0,1\}^*}, \{V_i^\Pi(x_1, x_2)\}_{x_{1,2} \in \{0,1\}^*}] \leq \\ \leq \text{negl}(\lambda). \quad \diamond$$

The definition of multi-party privacy can be obtained from Definition 7 by replacing one party with a coalition of parties and the other party with the rest of non-colluding parties.

Proof. (Theorem 4) We can build simulators for Protocol 3 from the simulators for semantic security, context hiding and perfect privacy and contrast them to the views of the parties, as in Definition 7.

The view of the actuator is:

$$V_A(f_{LQG}) = \Big(f_{LQG}, \text{upk}, \{u_i\}_{i \in 0 \cup [T-1]}, \{\hat{x}_i - r_i\}_{i \in [T-1]}, \text{coins}\Big).$$

We build a simulator Sim_A, that on inputs $\Big(1^\lambda, f_{LQR}, \{u_i\}_{i \in 0 \cup [T-1]}\Big)$ runs $(\text{mpk}, \text{msk}) \leftarrow \text{Init}(1^\lambda)$, $(\widetilde{\text{usk}}, \widetilde{\text{upk}}) \leftarrow \text{KeyGen}(\text{mpk})$, samples random values $\{\widetilde{r}_i\}_{i \in [T-1]} \in \mathcal{M}^n$ and then outputs:

$$\Big(f_{LQG}, \widetilde{\text{upk}}, \{u_i\}_{i \in 0 \cup [T-1]}, \{\widetilde{r}_i\}_{i \in [T-1]}, \widetilde{\text{coins}}\Big).$$

The indistinguishability between $V_A(f_{LQG})$ and $\text{Sim}_A\Big(1^\lambda, f_{LQR}, \{u_i\}_{i \in 0 \cup [T-1]}\Big)$ follows from the context-hiding property of LabHE and perfect secrecy of the one-time pad.

The cloud's view is: $V_C(\emptyset) =$

$$\Big(\text{E}(x_0), \text{E}(x_r), \text{E}(u_r), \text{E}(\Omega), \{\text{E}(z_i)\}_{i \in [T]}, \{\text{E}(\hat{x}_i)\}_{i \in [T-1]}, \text{coins}\Big),$$

where Ω are $K, L, \Gamma_1, \Gamma_2, \Gamma_3$. We build a simulator Sim_C that on input (1^λ) runs $\text{KeyGen}(1^\lambda)$, generates randomness for the shares, and creates encryptions of random values for all model parameters and signals received from the sub-systems and actuators, and outputs:

$$\Big(\widetilde{\text{E}(x_0)}, \widetilde{\text{E}(x_r)}, \widetilde{\text{E}(u_r)}, \widetilde{\text{E}(\Omega)}, \{\widetilde{\text{E}(z_i)}\}_{i \in [T]}, \{\widetilde{\text{E}(\hat{x}_i)}\}_{i \in [T-1]}, \widetilde{\text{coins}}\Big).$$

The indistinguishability between $V_C(\emptyset)$ and $\text{Sim}_A(1^\lambda)$ follows from the semantic security of LabHE.

The setup's view is $V_S(K, L, \Gamma_1, \Gamma_2, \Gamma_3) = (K, L, \Gamma_1, \Gamma_2, \Gamma_3)$. We build a simulator Sim_S that simply outputs its inputs $(K, L, \Gamma_1, \Gamma_2, \Gamma_3)$, which is trivially indistinguishable from $V_S(K, L, \Gamma_1, \Gamma_2, \Gamma_3)$. Similarly, the view of each sub-systems is $V_{S^i}(x_0^i, x_r^i, u_r^i) = (x_0^i, x_r^i, u_r^i)$. We then build simulators Sim_{S^i} that simply output their inputs (x_0^i, x_r^i, u_r^i), being trivially indistinguishable from $V_{S^i}(x_0^i, x_r^i, u_r^i)$.

The correctness of Protocol 3 follows from the correctness of the LQG algorithm and of the LabHE scheme. The proof is now complete. □

Proof. (Theorem 5) The proof of multi-party privacy uses the fact that neither the cloud nor the actuator can be in a coalition that has all the private data in the system and they cannot extract any information from the communication between themselves: the cloud receives encrypted data from the actuator, and does not have access to the key, and the actuator receives only random shares from the cloud.

A coalition between the cloud and the setup reduces to the case presented in Section 4.1. Consider now a coalition between the cloud, the setup and $\bar{N}$ sub-systems, where $0 \leq \bar{N} < N$. Because the data of the non-colluding sub-systems is encrypted with a semantically secure encryption scheme, and the coalition has access neither to the master key nor to the secret key of the non-colluding sub-systems, the coalitions cannot infer anything new about the private data of the non-colluding parties.

Similarly, a coalition between the actuator and the setup reduces to the case presented in Section 4.1. Consider now a coalition between the actuator, the setup and $\bar{N}$ sub-systems, where $0 \leq \bar{N} < N$. The outputs of the actuator consist in $u_k = -K\hat{x}_k + Kx_r + u_r$, where K is known because of the collusion with the setup and some entries in x_r and u_r are known from the collusion with some sub-systems. However, because the communication with the cloud is additively blinded by large random numbers (the secrets from the shares), the actuator cannot infer anything more about $\hat{x}_k$ than what it can infer solely from the colluding parties' data.

With this observations, one can construct simulators on the inputs of coalitions like in the proof of Theorem 4. □