Digital Implementation of Homomorphically Encrypted Feedback Control for Cyber-Physical Systems

Julian Tran, Farhad Farokhi, Michael Cantoni, Iman Shames

***Abstract*— This paper is about an encryption based approach to the secure implementation of feedback controllers for cyber-physical systems. Specifically, Paillier's homomorphic encryption is used in a custom digital implementation of a class of linear dynamic controllers, including static gain as a special case. The implementation is amenable to Field Programmable Gate Array (FPGA) realization. Experimental results, including timing analysis and resource usage characteristics for different encryption key lengths, are presented for the realization of a controller for an inverted pendulum; as this is an unstable plant, the control is necessarily fast.**

I. Introduction

A. Motivation

Advances in communication, control, and computer engineering have enabled design and implementation of large-scale systems, such as smart infrastructure, with remote monitoring and control, which is often desired due to the geographical spread of the system and requirements for flexibility of design (to accommodate future expansions). These positive features however come at the cost of security threats and privacy invasions [1]–[4].

Security threats can be decomposed into multiple categories based on resources available to adversaries [5]. A basic security attack that requires relatively few resources is eavesdropping in which an adversary monitors communication links to extract valuable information about the underlying system. Eavesdropping is often a starting point for more sophisticated attacks [6]. These attacks have resulted in the use of encryption [7], [8]. Figure 1 (a) illustrates the schematic diagram of a typical secure cyber-physical system with encryption. The actuator, system, and sensor (sometimes together referred to as the plant) form the physical system that must be remotely monitored and controlled. The physical system can be the electricity grid, transportation network, or a building, for example. Note that, although a single node is used in Figure 1 (b) to denote the sensor, in general it can comprise a collection of non-co-located sensors. That is, the sensors can be spread geographically within the underlying physical system to measure appropriate states in different locations, e.g., voltages and frequencies at various locations in an electricity. The same also goes for the actuator. For example in an electricity grid, the underlying physical system is the power network. The sensors (e.g., phasor measurement units or PMUs) measure phase and voltage at generators and major connections. The actuators then adjust the mechanical power of the turbines at the generators to maintain the frequency of the grid at $50\,\mathrm{Hz}$. The addition of the encryption and decryption units in Figure 1 (a) protects the overall system against eavesdroppers on the communication network; however, it does not provide any protection if the eavesdropper infiltrates the controller or if the controller itself is the eavesdropper (in industrial espionage). This is because sensitive information is decrypted prior to entering the controller and is thus readily available there. This motivates the use of a system, depicted in Figure 1 (b), with homomorphic encryption enabling controller computations to be performed on encrypted numbers.

In practice, the (physical) system in Figure 1 (b) is a continuous-time dynamical system. To control the system, the sensors sample the outputs of the system at regular intervals and transmits these sampled measurements to the controller through communication networks (e.g., WiFi or Bluetooth for short ranges or the Internet for longer ranges). The controller computes the necessary commands based on the received measurements and forwards the commands to the actuators for implementation. The actuators then apply and hold the received control signal for a fixed duration. This methodology for digital control of physical systems is often, unsurprisingly, referred to as sample and hold [9]. Before new samples can be processed by the controller, it must process the previous ones, compute the control inputs, and transmit the control inputs to the actuators. Therefore, the sampling rate of the sensors must not be faster than the inverse of the worst-case delay/latency caused by the required computations and communications. In order to guarantee stability and performance of the overall closed-loop system, we must ensure that the sampling occurs regularly and faster than a certain level (related to how fast the dynamics of the system are) [9]. In [10], an embedded processor, specifically a Raspberry Pi, was used to control a differential-wheeled robot in real-time using an encrypted controller. Controlling such a robot is not a complicated task as the underlying system is stable and, if the control signal is not updated with regular timing, the system would not violate safety constraints so long as it is restricted to move very slowly. Further, slowing down the sampling rate in this robot only degrades the performance by making it slower, not resulting in undesirable behaviours. In safety critical applications, however, the timing of the control loop is crucial; if we cannot ensure that the controller is able to provide the correct actuation signal within the sampling time of the system, then safe operation of the system cannot be guaranteed. Having

J. Tran, M. Cantoni, and I. Shames are with the Department of Electrical and Electronic Engineering at the University of Melbourne, Australia. F. Farokhi is with the Department of Electrical and Electronic Engineering at the University of Melbourne and CSIRO's Data61, Australia. e-mail: {julian.tran,ffarokhi,cantoni,ishames}@unimelb.edu.au

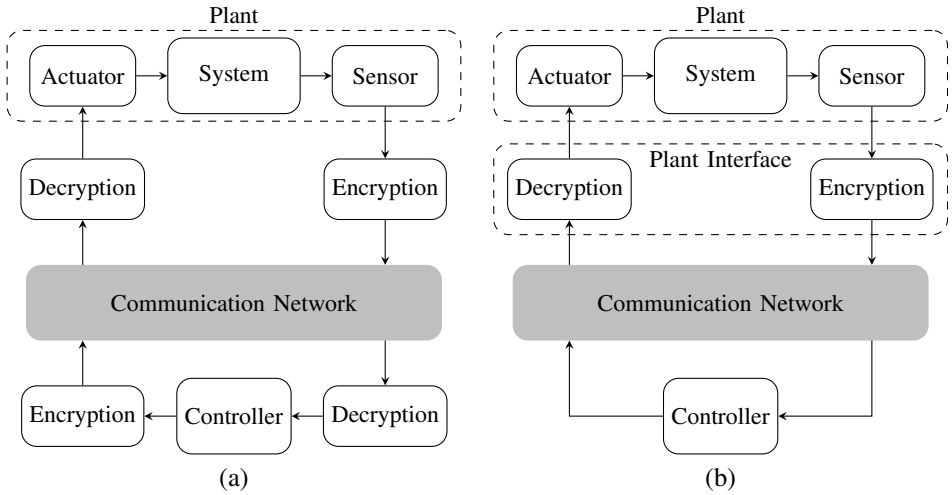

Fig. 1. The schematic diagram of a networked control system with (a) normal encryption and (b) semi-homomorphic encryption-decryption units.

tight control on the timing is unfortunately not entirely possible on embedded processors with operating systems because computations and their timings are subject to the operating system scheduling. Even without an operating system, software implementations can run in variable time due to branching logic. This motivates the design of a custom digital engine, amenable to realization on Field-Programmable Gate Arrays (FPGAs), for performing the necessary computations. This is the focus of the developments presented below.

B. Contributions

In this paper, we use homomorphic encryption, specifically the Paillier encryption [11], to implement linear time-invariant static and dynamic control laws. This includes many popular control laws, such as proportional-integral-derivative (PID) control [12] and linear quadratic regulators (LQR) [13]. Although the paper presents the digital system implementation within in the context of Paillier encryption, the underlying methodology is applicable to other homomorphic encryption methods that rely on exponentiations of large integer numbers, such as RSA and ElGamal encryption [14], [15], with possibly more limited functionality depending on the homomorphic property present. After the quantization and transformation of the controllers for implementation on ciphertexts, modular multipliers and exponentiators are implemented using Montgomery multiplication [16], [17]. We use these modules to design parallel circuits for encryption, controller computation, and decryption. We analyze the timing of the circuits and present experimental results for the control of an unstable system, namely, the inverted pendulum.

C. Related Studies

The study of homomorphic encryption, a form of encryption that enables computations on the encrypted data, dates back to the pioneering result of [18] after observing semi-homomorphic properties in RSA [14]. Semi-homomorphic refers to a simpler form of homomorphic encryption that only allows for a category of operations to be performed on the encrypted data. For example, RSA and ElGamal encryption [15] allows multiplication of plaintext data using multiplication of encrypted data, and Paillier encryption [11] allows summation of plaintext data using multiplication of encrypted data. The Gentry's encryption scheme [19] is the first fully-homomorphic encryption scheme that allows both multiplication and summation of plain data through appropriate arithmetic operations on encrypted data. Subsequently, other fully homomorphic encryption methods have been proposed, e.g., [20], [21]. The computational burden of fully-homomorphic encryption methods is often much greater than that of semi-homomorphic encryption methods.

Homomorphic encryption has been used previously for third-party cloud-computing services [22]–[27]. More recent studies [10], [28]–[31] have considered challenges associated with the use of homomorphic encryption in closed-loop control of physical systems, such as maintaining stability and performance, albeit without considering timing concerns (by not getting into the computational time of encryption, computation, and decryption and assuming all underlying computations are instantaneous). None of these studies consider dynamic control laws; they are all restricted to static control laws without any form of memory. This is because, in dynamical control laws with an encrypted memory, the number of bits required for representing the state of the controller grows linearly with the number of iterations. This renders the memory useless after a certain number iterations due to an overflow or an underflow[1]. We borrow theoretical results from [32]–[34] to propose an implementation of dynamical systems over ciphertexts.

An alternative to homomorphic encryption is secure multi-party computation based on secret sharing or other forms of encryption (possibly non-homomorphic encryption methodologies). A well-known method for secure multi-party computation is the Yao protocol, which was originally developed

[1]Underflow refers to the case where number of fractional bits required for representing a number becomes larger than the allowed number of fractional bits in a fixed-point number basis.

for secure two-party computations [35]. The protocol provides a method for evaluating a Boolean function without any party being able to observe the bits that flow through the circuit during the evaluation. This has been proved to be secure [36] and efficiently implementable for Boolean functions [37]. However, when dealing with more general mappings, i.e., non-Boolean functions, the efficiency of the protocol is limited as the problem of finding the most efficient Boolean representation of a function, in terms of the efficiency of implementing the Yao protocol [38], is not trivial [39]. Another approach is to utilize secret sharing in which a secret is divided into multiple shares and each party receives one share, which appears random to the receiving party. Then, appropriate computations on the secret shares can be performed to evaluate the outcome [40], [41]. Application of secret sharing to general problems is generally difficult and the digital design becomes problem specific.

Finally, note that the Paillier encryption scheme has been recently implemented on FPGAs in [42]; however, that paper considered the problem of privacy-preserving data mining, which has different requirements in comparison to real-time encrypted control. This differences in requirements resulted in the adoption of different digital design architectures in this paper. In part, the encryption is faster in this paper due to exploiting the binomial expansion for the specific choice of the exponential base. Further, due to the differences between the operations required for data mining and controller computation, the developed implementations differ.

D. Paper Outline

The rest of the paper is organized as follows. In Section II, the building blocks of the networked control systems in Figure 1 (b) are presented and we describe the implementation of the control laws over ciphertexts. In Section III, the digital design for FPGA realization is described. We present the experimental results for the control of an inverted pendulum in Section IV. Finally, we conclude the paper and present avenues for future research in Section V.

II. Secure Cyber-Physical Systems

In this section, we discuss the building blocks of the networked control systems in Figure 1 (b). We start by describing a model for the physical system.

A. Physical Systems

Many physical systems, such as autonomous vehicles, robotic systems, smart buildings, and transport systems, can be modelled as a continuous-time time-invariant dynamical system:

$$\mathcal{P}_c: \quad \dot{x}_{sc}(t) = f_c(x_{sc}(t), u_c(t)), \quad x_{sc}(0) = x_0, \tag{1a}$$
$$y_c(t) = g_c(x_{sc}(t)), \tag{1b}$$

where $x_{sc}(t) \in \mathbb{R}^{n_s}$ is the system state vector (e.g., pose, position, and velocity), $u_c(t) \in \mathbb{R}^{n_u}$ is the vector of control inputs (e.g., acceleration, force, or torque), and $y_c(t) \in \mathbb{R}^{n_y}$ is the vector of system outputs (e.g., position). In short, (1a) describes the state evolution over time, which depends on the previous state and the control input to the system, and (1b) produces the observable system output, which depends on the current system state. Using the sample and hold methodology [9], we can extract a corresponding system in the discrete-time domain:

$$\mathcal{P}: \quad x_s[k+1] = f(x_s[k], u[k]), \quad x_s[0] = x_0, \tag{2a}$$
$$y[k] = g(x_s[k]), \tag{2b}$$

where, for all $k \in \mathbb{N} \cup \{0\}$, $x_s[k] = x_{sc}(k\Delta T)$, $y[k] = y_c(k\Delta T)$, $u_c(t) = u[k]$ for all $k\Delta T \leq t < (k+1)\Delta T$, and ΔT denotes the sampling time. Most often, we are interested in designing and employing a controller that takes the system output as an input, and produces a control input to the system to regulate the system output around a desired setpoint.

B. Homomorphic Encryption

A public key encryption scheme can be described by the tuple $(\mathbb{P}, \mathbb{C}, \mathbb{K}, \mathcal{E}, \mathcal{D})$, where $\mathbb{P}$ is the set of plaintexts, $\mathbb{C}$ is the set of ciphertexts, $\mathbb{K}$ is the set of keys, $\mathcal{E}$ is the encryption algorithm, and $\mathcal{D}$ is the decryption algorithm. As such encryption schemes are asymmetric, each key $\kappa = (\kappa_p, \kappa_s) \in \mathbb{K}$ is composed of a public key κ_p (which is shared with everyone and is used to encrypt plaintexts), and a private key κ_s (which is kept secret and is used to decrypt ciphertexts). The algorithms $\mathcal{E}$ and $\mathcal{D}$ are publicly known, and use the keys as parameters, which are generated for each new use-case. It is obviously required that $\mathcal{D}(\mathcal{E}(x, \kappa_p), \kappa_p, \kappa_s) = x$.

Definition 1 (Homomorphism in Cryptography): A public key encryption scheme $(\mathbb{P}, \mathbb{C}, \mathbb{K}, \mathcal{E}, \mathcal{D})$ is homomorphic if there exist operators $\circ$ and $\diamond$ such that $(\mathbb{P}, \circ)$ and $(\mathbb{C}, \diamond)$ are algebraic groups and $\mathcal{E}(x_1, \kappa_p) \diamond \mathcal{E}(x_2, \kappa_p) = \mathcal{E}(x_1 \circ x_2)$.

Typically, the sets $\mathbb{P}$ and $\mathbb{C}$ are finite rings of integers $\mathbb{Z}_{n_P}$ and $\mathbb{Z}_{n_C}$ respectively. Then, the modular addition operation ($x_1 \circ x_2 = (x_1 + x_2) \bmod n_P$) and the modular multiplication operation ($x_1 \circ x_2 = x_1 x_2 \bmod n_P$) both form groups with $\mathbb{P}$. If there exists an operator $\diamond$ that satisfies the definition of a homomorphic encryption scheme when $\circ$ is defined as modular addition, we call the encryption scheme additively homomorphic. Likewise, if there exists an operation $\diamond$ that satisfies the definition of a homomorphic encryption scheme when $\circ$ is defined as modular multiplication, we call the encryption scheme multiplicatively homomorphic. If both descriptions apply, the encryption scheme is fully-homomorphic; if only one description applies, it is semi-homomorphic. Importantly, the properties of fully-homomorphic and semi-homomorphic encryption schemes allow additions and multiplications of plaintexts to be performed through the generation of a ciphertext from other ciphertexts, without any intermediate decryptions and encryptions.

Encryption schemes, such as the Paillier [11], RSA [14], and ElGamal [15], are examples of semi-homomorphic encryption schemes. The Paillier encryption scheme is additively homomorphic, while the RSA and ElGamal encryption schemes are multiplicatively homomorphic. These homomorphic encryption schemes have been used in the literature to ensure privacy and security when various computational

tasks, such as computing set intersections, data mining, executing arbitrary programs, and controlling dynamical systems, are performed by untrusted parties; see, e.g., [10], [22]–[26], [34] and references there-in for examples. The above-mentioned homomorphic encryption schemes involve calculating modular exponentiations (i.e., $b^a \bmod M$ for positive integers a, b, and M), which is a computationally expensive operation. The time required to perform encryption, decryption, and homomorphic operations on ciphertexts, depends largely on the speed with which modular exponentiation can be calculated. This can potentially limit the usability of homomorphic encryption schemes for real-time control of physical systems.

Definition 2 (Indistinguishability under Chosen Plaintext): Consider a scenario in which a polynomial-time-bounded adversary provides two plaintexts. One of these plaintexts is randomly chosen and encrypted. An encryption scheme is said to be indistinguishable under chosen plaintext attack, if the adversary has a negligible advantage[2] over guessing which of the two plaintexts were encrypted, using any information apart from the private key.

Indistinguishability under chosen plaintext is a desirable property because an adversary is unable to determine the decryption of a ciphertext, by trialling encryption of likely plaintexts. The RSA encryption scheme does not have this property unless modified to OAEP-RSA [44]. The Paillier and ElGamal encryption schemes have this property, as they introduce a large random number during encryption, allowing a single plaintext to encrypt non-deterministically to many possible ciphertexts, which removes any significant advantage in trialling encryption of likely plaintexts [11], [15].

In what follows, we use Paillier encryption scheme as it is additively homomorphic and satisfies indistinguishability under chosen plaintext attack. Note that the ideas of this paper can be readily used for other homomorphic encryption relying on modular exponentiation and is not restricted to the Paillier encryption. Paillier encryption scheme is as follows. First, two large prime numbers p and q are randomly chosen to generate keys. The public key is $\kappa_p = N = pq$ and the private key is $\kappa_s = (\lambda, \mu) = (\operatorname{lcm}(p-1, q-1), \lambda^{-1} \bmod N)$ where $\operatorname{lcm}(a, b)$ denotes the least common multiple of integers a and b. Note that $\lambda^{-1} \bmod N$ is a unique integer μ in $\mathbb{Z}_N$ such that $\lambda\mu \bmod N = 1$. In the Paillier encryption scheme, the set of plaintexts and ciphertexts are, respectively, $\mathbb{P} = \mathbb{Z}_N$ and $\mathbb{C} = \mathbb{Z}_{N^2}$. Encrypting a plaintext t is done by calculating $\mathcal{E}(t) = (N+1)^t r^N \bmod N^2$, where $r \in \{x \in \mathbb{Z}_N \mid \gcd(x, N) = 1\}$ is randomly chosen. Note that, because of using $N+1$ as the exponentiation basis in the encryption algorithm, it can be rewritten as $\mathcal{E}(t) = (Nt+1)r^N \bmod N^2$. This property follows from the use of binomial expansion because $(N+1)^t r^N \bmod N^2 = (\sum_{i=0}^{t} \binom{t}{i} N^i) r^N \bmod N^2 = (Nt+1)r^N \bmod N^2 + (N^2 \sum_{i=2}^{t} \binom{t}{i} N^{i-2}) r^N \bmod N^2 = (Nt+1)r^N \bmod N^2$. Using this property makes our implementation of the encryption considerably faster than [42]. Decryption of a ciphertext c is done by calculating $\mathcal{D}(c) = L(c^\lambda \bmod N^2)\mu \bmod N$, where $L(u) = (u-1)/N$.

The additive homomorphic property follows from $\mathcal{D}(\mathcal{E}(t_1, \kappa_p)\mathcal{E}(t_2, \kappa_p), \kappa_p, \kappa_s) = t_1 + t_2 \bmod N$. Further, we have $\mathcal{D}(\mathcal{E}(t_1, \kappa_p)^{t_2}, \kappa_p, \kappa_s) = t_1 t_2 \bmod N$. Note that this is not a true multiplicative homomorphic property, as t_2 is not encrypted; the encrypted result is formed from one ciphertext and one plaintext, rather than two ciphertexts. In the remainder of this paper, we use $\oplus$ to denote the additive homomorphic operator on ciphertexts and $\otimes$ to denote the pseudo-multiplicative homomorphic operator, i.e., $c_1 \oplus c_2 := (c_1 c_2) \bmod N^2$ and $t \otimes c := c^t \bmod N^2$.

C. Secure Controller Implementation

Often feedback controllers take the form of static gain from the error to then control input:

$$\mathcal{C}: \quad u[k] = D(s[k] - y[k]), \tag{3}$$

where the control signal input $u[k]$ is proportional to the error between the system output $y[k]$ and the setpoint $s[k]$, with constant gain $D \in \mathbb{R}^{n_u \times n_y}$; methods for the design of such control laws can be found in [45] and references therein. Note that the computations required to implement the static controller are additions and multiplications. We restrict the controller input to fixed-point numbers and use the mapping from fixed point numbers to the integers in [10]. This allows the equivalent operations of addition and multiplication to be effectively applied to fixed point numbers and integers over the ciphertext. The effect of the quantization error can be made arbitrarily small by increasing the number of bits used to represent the underlying numbers (specifically the number of fractional bits), at the expense of increased computational cost [10], given bounds on the size of disturbances that can act on the system. Quantizing also introduces saturation, which can be quite problematic. However, the negative effects of saturation may also be managed by increasing the number of bits (specifically the number of integer bits) used to represent the underlying numbers [10].

To provide more detail about the quantization process and its effect on the control law, we introduce the set of fractional numbers as

$$\mathbb{Q}(n, m) := \Bigg\{ b \in \mathbb{Q} \,|\, b = -b_n 2^{n-m-1} + \sum_{i=1}^{n-1} 2^{i-m-1} b_i, \\ b_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \Bigg\}.$$

The quantization operator $\mathcal{Q}: \mathbb{R} \to \mathbb{Q}$ is defined as $\mathcal{Q}(z) := \arg\min_{z' \in \mathbb{Q}(n,m)} |z - z'|$. With slight abuse of notation, we use $\mathcal{Q}(A)$ and $\mathcal{Q}(x)$ to denote the entry-wise quantization of any $A \in \mathbb{R}^{n \times m}$ and $x \in \mathbb{R}^n$, respectively. The quantized controller is then given by

$$\bar{\mathcal{C}}: \quad \bar{u}[k] = \bar{D}(\bar{s}[k] - \bar{y}[k]), \tag{4}$$

where $\bar{D} = \mathcal{Q}(D)$, $\bar{s}[k] = \mathcal{Q}(s[k])$, and $\bar{y}[k] = \mathcal{Q}(y[k])$. We use the bar, e.g., $\bar{x}$, to denote the quantized version of any

[2]Negligible advantage means that the difference between the probability of guessing the correct plaintext and the probability of guessing the wrong plaintext goes to zero rapidly as the key length goes to infinity [43].

variable, e.g., x. The map from fixed point numbers to the integers $\mathbb{Z}_{2^{n'}}$, borrowed from [10], is defined by

$$\hat{s}_i[k] = (2^m \bar{s}_i[k]) \bmod 2^{n'} \tag{5a}$$
$$\hat{y}_i[k] = (2^m \bar{y}_i[k]) \bmod 2^{n'} \tag{5b}$$
$$\hat{D}_{ij} = (2^m \bar{D}_{ij}) \bmod 2^{n'} \tag{5c}$$
$$\hat{u}_i[k] = (2^{2m} \bar{u}_i[k]) \bmod 2^{n'} \tag{5d}$$

for all i, j, where $n' = 2n + \lceil \log_2(n_y) \rceil$ to prevent overflows. Here, for simplicity, we assume that all variables use the same n and m, but these values can differ for various parts of the controller in general [10]. Now, we can rewrite the control law in (4) on ciphertexts as

$$\tilde{\mathcal{C}}: \quad \tilde{u}_i[k] = \oplus_{j=1}^{n_y} (\hat{D}_{ij} \otimes (\tilde{s}_j[k] - \tilde{y}_j[k])), \tag{6}$$

where the tilde is used to denote the encrypted integers; i.e.,

$$\tilde{u}_i[k] = \mathcal{E}(\hat{u}_i[k], \kappa_p), \tag{7a}$$
$$\tilde{s}_j[k] = \mathcal{E}(\hat{s}_j[k], \kappa_p), \tag{7b}$$
$$\tilde{y}_j[k] = \mathcal{E}(\hat{y}_j[k], \kappa_p). \tag{7c}$$

Using (6), an encrypted control input is determined without decrypting the quantized system output sample. Finally, the control input is decrypted at the actuator as

$$\hat{u}_i[k] = \mathcal{D}(\tilde{u}_i[k], \kappa_p, \kappa_s) \bmod 2^{n'}, \tag{8a}$$
$$\bar{u}_i[k] = 2^{-2m} (\hat{u}_i[k] - 2^{n'} \mathbb{1}_{\hat{u}_j[k] \geq 2^{n'-1}}). \tag{8b}$$

The static controller in (3) can be generalized to the following dynamic form, which covers reset integral, lead and lag controllers:

$$\mathcal{C}: x[k+1] = \begin{cases} Ax[k] + B(s[k] - y[k]), & k+1 \bmod T > 0, \\ 0, & k+1 \bmod T = 0, \end{cases} \tag{9a}$$
$$u[k] = Cx[k] + D(s[k] - y[k]), \tag{9b}$$

where $x[k] \in \mathbb{R}^{n_x}$ is the controller state and T is the number of time steps between controller state resets. In this paper, we consider resetting dynamic control laws because implementing encrypted controllers over an infinite horizon is impossible due to memory issues (through repeated multiplication of fixed point numbers, the numbers of the bits required for representing the fractional and integer parts continuously grow). Resetting controllers have been previously studied in [32]–[34]. Again, the computations performed by the dynamical controller are additions and multiplications, so if we restrict the controller inputs and state to fixed-point numbers, then by using the ciphertext operators in a homomorphic encryption scheme, the controller can perform its computations on ciphertext inputs and states. To do so, define

$$\hat{s}_i[k] = (2^m \bar{s}_i[k]) \bmod 2^{n'}, \tag{10a}$$
$$\hat{y}_i[k] = (2^m \bar{y}_i[k]) \bmod 2^{n'}, \tag{10b}$$
$$\hat{A}_{ij} = (2^m \bar{A}_{ij}) \bmod 2^{n'}, \tag{10c}$$
$$\hat{B}_{ij}[k] = (2^{(k \bmod T+1)m} \bar{B}_{ij}[k]) \bmod 2^{n'}, \tag{10d}$$
$$\hat{C}_{ij} = (2^m \bar{C}_{ij}) \bmod 2^{n'}, \tag{10e}$$
$$\hat{D}_{ij}[k] = (2^{(k \bmod T+1)m} \bar{D}_{ij}[k]) \bmod 2^{n'}, \tag{10f}$$
$$\hat{x}_i[k] = (2^{(k \bmod T+1)m} \bar{x}_i[k]) \bmod 2^{n'}, \tag{10g}$$
$$\hat{u}_i[k] = (2^{(k \bmod T+2)m} \bar{u}_i[k]) \bmod 2^{n'}, \tag{10h}$$

where $n' = (n_x + 1)T + n_u + n(T + 2)$. The controller equations can then be rewritten to operate on ciphertexts as in (11). Finally, the control signal at the actuator can be extracted by

$$\hat{u}_i[k] = \mathcal{D}(\tilde{u}_i[k], \kappa_p, \kappa_s) \bmod 2^{n'}, \tag{12a}$$
$$\bar{u}_i[k] = 2^{-(k \bmod T+2)m} (\hat{u}_i[k] - 2^{n'} \mathbb{1}_{\hat{u}_i[k] \geq 2^{n'-1}}). \tag{12b}$$

III. Digital Design

Timing is an important issue when implementing controllers in real-time. While the maximum amount of computation to be performed by the controller is the same in every iteration, implementations on platforms running an operating system are subject to variable timing performance dependent on operating system scheduling. Even without an operating system, software implementations can run in variable time due to branching. Such implementations are therefore not acceptable for systems with strict deadlines. This motivates the development of a custom digital engines for performing the computations. Hardware implementation of homomorphic encryption based secure feedback control can result in faster sampling rates than software implementations. The speedup of a digital design in hardware over a software design can be from many aspects. Hardware designs are able to take advantage of full parallelism, while software designs typically run sequentially on a few parallel threads, and are thus limited in their parallelism. Hardware designs can also introduce pipelining into data paths, where the computation is divided into a pipeline of sequential stages, with stages all running at the same time, and each stage passing its result to the next stage. This can be used to increase achievable data throughput compared to sequential software designs, as new data can be passed through the first stage of the pipeline while there is still data to be processed in the subsequent stages.

A. Modular Multiplication and Exponentiation

In many homomorphic encryption schemes, including Paillier encryption scheme, efficient implementation of modular exponentiation is essential for fast encryption, decryption, and homomorphic operations. Within the context of secure feedback control implementation, the time it take to perform encryption, decryption and homomorphic operations on cyphertexts, is a lower bound on the control loop sample

$$\tilde{\mathcal{C}}: \quad \tilde{x}_i[k+1] = \begin{cases} \left[\oplus_{j=1}^{n_x}(\hat{A}_{ij} \otimes \tilde{x}_j[k])\right] \oplus \left[\oplus_{j=1}^{n_y}(\hat{B}_{ij}[k] \otimes (\tilde{s}_j[k] - \tilde{y}_j[k]))\right], & k+1 \bmod T > 0, \\ \mathcal{E}(0, \kappa_p), & k+1 \bmod T = 0, \end{cases} \tag{11a}$$

$$\tilde{u}_i[k] = \left[\oplus_{j=1}^{n_x}(\hat{C}_{ij} \otimes \tilde{x}_j[k])\right] \oplus \left[\oplus_{j=1}^{n_y}(\hat{D}_{ij}[k] \otimes (\tilde{s}_j[k] - \tilde{y}_j[k]))\right]. \tag{11b}$$

Pseudocode 1 Right-to-left method for modular exponentiation using Montgomery multiplication in modulus $M = N^2$

Parameters
- N Paillier public key
- R Montgomery radix

Inputs
- B Integer base in Montgomery form $B = bR \bmod N^2$
- E Integer exponent with l bits

Outputs
- P Power in Montgomery form $P = b^E R \bmod N^2$

function MONTEXP(B, E)
  $P \leftarrow R \bmod N^2$
  for $i = 1, ..., l$ **do**
    if $E \bmod 2 = 1$ **then**
      $P \leftarrow$ MONTMULT$[M = N^2](P, B)$
    end if
    $E \leftarrow \lfloor E/2 \rfloor$
    $B \leftarrow$ MONTMULT$[M = N^2](B, B)$
  end for
  return P
end function

Pseudocode 2 [46] Modified Coarsely Integrated Operand Scanning (CIOS) method variant of Montgomery multiplication using 16 bits per word and without the final conditional subtraction.

Parameters
- M Odd modulus
- w Number of 16 bit words such that $M < 2^{16w}$
- M' such that $MM' \bmod 2^{16} = 2^{16} - 1$

Inputs
- X Input such that $X < 2M$
- Y Input such that $Y < 2M$

Outputs
- T Such that $T \bmod M = XYR^{-1} \bmod M, T < 2M$ where $R = 2^{16(w+1)}$ and $RR^{-1} \bmod M = 1$

function MONTMULT(X, Y)
  $T = 0$
  for $i = 1, ..., w + 1$ **do**
    $Z \leftarrow X(Y \bmod 2^{16})$
    $Y \leftarrow \lfloor Y/2^{16} \rfloor$
    $m \leftarrow ((T \bmod 2^{16}) + (Z \bmod 2^{16}))M' \bmod 2^{16}$
    $T \leftarrow (T + Z + mM)/2^{16}$
  end for
  return T
end function

period, which when reduced, typically leads to improved performance for systems with fast dynamics (e.g., an unstable inverted pendulum). Note that, in principle, it is possible to decrease the time required for computations by decreasing the encryption key length; however, this would reduce the security of the system which is not desirable.

We utilize the right-to-left binary method for calculating modular exponentiation, which is summarized in Pseudocode 1. The algorithm is particularly useful for our application as it allows for the parallelization of the two modular multiplications in each iteration. This gives a speedup of up to two times, and results in a constant latency as the conditional modular multiplication is performed in parallel to the modular multiplication that must be always performed in each iteration. The right-to-left binary method for exponentiation involves calculating many sequential modular multiplications. The algorithm best suited for this purpose is Montgomery multiplication [16]. It removes the need to perform a trial division by the modulus which is an expensive operation in hardware, and instead only involves additions, multiplications, and right shifts; e.g., see Pseudocode 2. However, for it to be useful for implementing modular multiplications, its operands must be converted to Montgomery form, and the result must be converted back from Montgomery form. These conversions can be done using additional Montgomery multiplications. The Montgomery form of an integer a when using a modulus of M is $(aR) \bmod M$, where the Montgomery radix R is typically a power of 2, and is larger than M. When using Montgomery multiplication in the right-to-left binary method for modular exponentiation (subsequently, referred to as Montgomery exponentiation), the conversions to and from the Montgomery form only occur before and after the exponentiation, as the intermediate (theoretical) conversions between the sequential multiplications within the exponentiation cancel out.

Many hardware designs for computing Montgomery multiplications exist. A design involving the Karatsuba multiplication algorithm can be used to evaluate very large multiplications [47]. While this proved to be computationally effective in [47], such a method may not be suitable for some applications due to prohibitive hardware resource required for evaluating Montgomery multiplications even with relatively small operands. Another method for implementing Montgomery multiplication involves using the Coarsely Integrated Operand Scanning (CIOS) variant [17] with a word size of a single bit. Implementations of this algorithm are described in [48], [49]. The bitwise approach greatly

Pseudocode 3 Encryption of the system outputs in the plant interface (or of the setpoints elsewhere) using the Montgomery multiplication and the Montgomery exponentiation.

Parameters
N Paillier public key
R Montgomery radix
Inputs
$\hat{y}$ System outputs $\hat{y}_1, ..., \hat{y}_{n_y}$
z Values $z_1, ..., z_{n_y}$ where $z_i = {r'_i}^N R \bmod N^2$
Outputs
$\check{y}$ Encrypted system outputs in Montgomery form $\check{y}_1, ..., \check{y}_{n_y}$
function ENCRYPT($\hat{y}, z$)
for $i = 1, ..., n_y$ **do**
$temp \leftarrow$ MONTMULT$[M = N^2](NR \bmod N^2, \hat{y}_i)$
$temp \leftarrow$ MONTMULT$[M = N^2](temp + 1, R^2 \bmod N^2)$
$\check{y}_i \leftarrow$ MONTMULT$[M = N^2](z_i, temp)$
end for
return $\check{y}$
end function

simplifies the architecture of the Montgomery multiplier, as it is only required to perform additions and right shifts. However, the bitwise design does not make use of the embedded multipliers available on most modern FPGA devices.

A blockwise implementation of the CIOS method of Montgomery multiplication is ideal for the purposes of this paper as it is amenable to the use of embedded multipliers in FPGAs to perform smaller multiplications. Some implementations of this algorithm are discussed in [50]–[52]. These implementations range from using a constant number of embedded multipliers to the case where the number of embedded multipliers scales linearly with the number of bits in the operands to perform large parallel multiplications. Therefore, based on the amount of the available hardware resources, an appropriate implementation of the blockwise CIOS-based Montgomery multiplier can be designed to ensure the resources are utilized effectively.

In Pseudocode 2, we present the modified CIOS method in [46] with a word size of 16 bits. The modified CIOS method removes the conditional final subtraction in typical Montgomery multiplication implementations to reduce hardware resource consumption. Pseudocode 2 also differs from the conventional Montgomery multiplication in that it produces outputs that possibly have the modulus M added to it, rather than an output in $\mathbb{Z}_M$. This is acceptable as long as an explicit conversion from this modified Montgomery form, through Montgomery multiplication by 1, is used to produce the final result [46].

Across all Montgomery multipliers, we use the same value of the Montgomery radix $R = 2^{16(w+1)}$, where w is the smallest integer such that $N^2 + 2 < 2^{16w}$; note that $N^2 + 2$ is the largest modulus used in the system.

Pseudocode 4 Computing $r^N \bmod N^2$ in the plant interface using the Montgomery multiplication and the Montgomery exponentiation.

Parameters
N Paillier public key
Inputs
r Random values $r_1, ..., r_{n_y}$
Outputs
z Values $z_1, ..., z_{n_y}$ where $z_i = {r'_i}^N R \bmod N^2$
function CALCULATERANDOM(r)
for $i = 1, ..., n_y$ **do**
$z_i \leftarrow$ MONTEXP(r_i, N)
end for
return z
end function

Throughout the encrypted control system, there are only three different values used as modulus, so these values can be coded into the Montgomery multipliers required, with an input allowing for the selection of the modulus. In the Paillier encryption scheme, all modular exponentiations have modulus $M = N^2$, where N is the public key.

In what follows, using the digital designs of the Montgomery multipliers and the Montgomery exponentiators as the underlying arithmetic blocks, we design plant interface and controller modules for an encrypted control system secured with the Paillier encryption scheme. As shown in Figure 1, the plant interface performs encryptions of system outputs and decryptions of control inputs, and the controller evaluates the control law securely over encrypted data. The ciphertexts transmitted between the plant interface and the controller are in the Montgomery form, possibly with the modulus added: $\check{a} = (\tilde{a}R \bmod N^2) + bN^2, b \in \{0, 1\}$.

Parallelization of the digital design can be done both in the designs the building blocks of the Montgomery multiplier and the Montgomery exponentiator, and in the designs of the plant interface and controller. Adding parallelization increases the resource consumption of the hardware design, which is a limiting factor. To offset this, resources are reused whenever possible. In particular, the Montgomery multipliers used to implement the Montgomery exponentiators can also be used whenever single modular multiplications are required, rather than instantiating separate Montgomery multipliers.

B. Plant Interface Module Design

The plant interface's role in the encrypted control system is to encrypt the system outputs and decrypt the control inputs. There is no requirement for a single plant interface that performs both encryptions and decryptions, as these functionalities can be trivially separated into distinct modules if the actuators and the sensors are physically apart. However, a single plant interface module allows for the reuse of hardware resources for both encryption and decryption, reducing the hardware cost of the system.

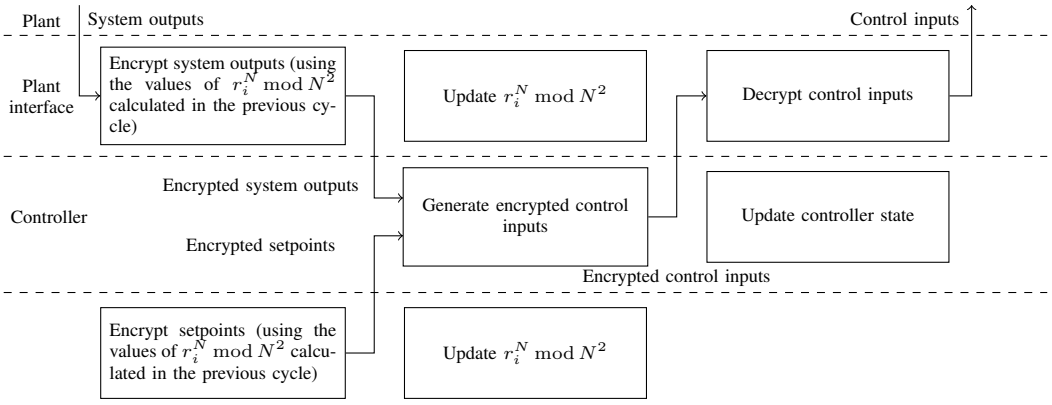

Fig. 2. Illustration of the parallel tasks performed by the plant interface and the controller within a single sampling period.

Pseudocode 5 Decryption of the control inputs in the plant interface using the Montgomery multiplication and the Montgomery exponentiation.

Parameters
N Paillier public key
R Montgomery radix
n' Number of bits in mapping from fixed point numbers $\mathbb{Q}(n,m)$ to integers $\mathbb{Z}_{2^{n'}}$
μ Part of Paillier private key
λ Part of Paillier private key
N^{-1} Value $\in \mathbb{Z}_{N^2+2}$ such that $NN^{-1} \bmod (N^2+2) = 1$
Inputs
$\check{u}$ Encrypted control inputs in Montgomery form $\check{u}_1, ..., \check{u}_{n_u}$
Outputs
$\hat{u}$ Control inputs $\hat{u}_1, ..., \hat{u}_{n_u}$
function DECRYPT($\check{u}$)
for $i = 1, ..., n_u$ **do**
$temp \leftarrow \text{MONTEXP}(\check{u}_i, \lambda)$
$temp \leftarrow \text{MONTMULT}[M = N^2](temp, 1)$
$temp \leftarrow \text{MONTMULT}[M = N^2 + 2](temp - 1, N^{-1}R^2 \bmod (N^2+2))$
$temp \leftarrow \text{MONTMULT}[M = N^2 + 2](temp, 1)$
$temp \leftarrow \text{MONTMULT}[M = N](temp, \mu R^2 \bmod N)$
$\hat{u}_i \leftarrow \text{MONTMULT}[M = N](temp, 1) \bmod 2^{n'}$
end for
return $\hat{u}$
end function

Paillier encryption algorithm involves the calculation of values for $r^N \bmod N^2$, which is independent of the plaintext being encrypted. Note that it is possible to generate the value of r^N needed to encrypt the next system output sample in parallel with the controller computations involving the encryption of the current sample. This parallelization between the plant interface and controller decreases the time required for completing the necessary tasks within a sampling period without utilizing extra resources. The diagram in Figure 2 illustrates this with the task of "update $r_i^N \bmod N^2$". If separate interfaces are used for encryption and decryption, further parallelization can be introduced by starting decryption while values for $r^N \bmod N^2$ are still being computed, assuming that the encrypted control inputs are available from the controller at that such a time.

There are various approaches for generating cryptographically secure random or pseudo-random values for r. Random methods involve sampling a noise source, such as oscillator jitter; examples can be found in [53]–[55]. Pseudo-random methods are algorithms that generate numbers from an initial seed, which should be generated from a random method; examples can be found in [56], [57]. Depending on the method used, the generator can be implemented on the FPGA, or external to it. The generated random numbers are used as the input to Pseudocode 4, which first converts them to the Montgomery form, in order to compute r^N. Note that, for larger encryption key lengths, checking that $\gcd(r, N) = 1$ is not required, as the probability that this is not the case is negligible. We also do not need to convert random numbers to Montgomery form before performing Montgomery exponentiation. Assume that we are given a uniformly distributed random number r in $\mathbb{Z}_N$. With $r' = (rR^{-1}) \bmod N^2$, where R is the Montgomery radix, it can be seen $r' \bmod N = (rR^{-1}) \bmod N$ is also uniformly distributed random number in $\mathbb{Z}_N$ because $r \mapsto (rR^{-1}) \bmod N$ is bijective (R and N are coprime). Further, $(r' \bmod N)^N \bmod N^2 = (r')^N \bmod N^2$ because $(\alpha + kN)^N \bmod N^2 = \alpha^N \bmod N^2$ for any $\alpha, k \in \mathbb{Z}$. That is, the Montgomery form of r' is in fact r. Therefore, using the Montgomery exponentiation algorithm without first converting to Montgomery form, we can compute $r^N \bmod N^2$ which is equal to $(r' \bmod N)^N R \bmod N^2$.

The tasks performed by the plant interface are described in Pseudocodes 3, 4, and 5, expressed as a sequence of the Montgomery exponentiations and the Montgomery multiplications. The inputs to all of these Montgomery operations are either constants (as the algorithm parameters do not change within any given implementation), algorithm inputs, or the result of the previous operations (possibly with 1 added to or

Pseudocode 6 Computing the control input of the static controller using the Montgomery multiplication and the Montgomery exponentiation.

Parameters
- N Paillier public key
- R Montgomery radix
- n' Number of bits in mapping from fixed point numbers $\mathbb{Q}(n, m)$ to integers $\mathbb{Z}_{2^{n'}}$
- $\hat{D}$ Controller matrix

Inputs
- $\check{y}$ Encrypted system outputs in Montgomery form $\check{y}_1, ..., \check{y}_{n_y}$
- $\check{s}$ Encrypted setpoints in Montgomery form $\check{s}_1, ..., \check{s}_{n_y}$

Outputs
- $\check{u}$ Encrypted control inputs in Montgomery form $\check{u}_1, ..., \check{u}_{n_u}$

function GENERATECONTROL($\check{y}, \check{s}$)
  for $i = 1, ..., n_y$ **do** ▷ Generate encrypted error values
    $temp \leftarrow \text{MONTEXP}(\check{y}, 2^{n'} - 1)$
    $\check{e}_i \leftarrow \text{MONTMULT}[M = N^2](temp, \check{s})$
  end for
  for $i = 1, ..., n_u$ **do** ▷ Generate encrypted scalar products
    for $j = 1, ..., n_y$ **do**
      $temp_{ij} \leftarrow \text{MONTEXP}(\check{e}_j, \hat{D}_{ij})$
    end for
  end for
  for $i = 1, ..., n_u$ **do** ▷ Homomorphically sum up encrypted scalar products
    for $j = 2, ..., n_y + n_u$ **do**
      $temp_{i1} \leftarrow \text{MONTMULT}[M = N^2](temp_{i1}, temp_{ij})$
    end for
    $\check{u}_i \leftarrow temp_{i1}$
  end for
  return $\check{u}$
end function

subtracted from it, which can be evaluated within the clock cycle used for routing the result to the next stage). Because of this, the plant interface can be implemented by simply selecting these inputs to Montgomery exponentiator modules.

Every loop in Pseudocodes 3, 4, and 5 can be parallelized, as the iterations are independent of each other. For example, the encryptions of plaintexts are independent of each other, so individual encryptions can all be performed in parallel. The same applies to the calculation of values for $r^N \bmod N^2$, and to decryptions of the ciphertexts. These parallelizations allow physical systems with more inputs and outputs to be controlled, without increasing the time required to perform the encryptions and decryptions. However, as a trade-off more hardware resources are required, and so in resource limited scenarios, these computations can be performed sequentially if a longer sampling period is acceptable. In the case that the computations all be performed sequentially, the plant interface would then require only one Montgomery exponentiator module with which it performs the prescribed Montgomery exponentiations and multiplications one by one, using the same module. If on the other hand the plant interface is fully parallelized, then it would require $\max(n_y, n_u)$ Montgomery exponentiators, as the maximum number of encryptions or decryptions to be performed in parallel depends on whether there are more system outputs to encrypt or more control inputs to decrypt.

C. Controller Module Design

First, we consider static controllers in (6). The controller uses the encrypted system output to generate encrypted control inputs that stabilize the system output about a setpoint. Pseudocode 6 implements the operations on ciphertexts in (6). Each product, in the plaintext, is replaced with a modular exponentiation with the ciphertext as the base and the plaintext as the exponent. These exponentiations can all be performed in parallel. Afterwards, the results in each row can be combined in a binary tree structure using modular multiplications, replacing summations in the plaintext. The total latency of the modular multiplications is equal to the time required to perform $\lceil \log_2(n_y) \rceil$ Montgomery multiplications, where n_y is the number of the system outputs.

Now, consider dynamic controllers in (11). There are now additional computations for updating the state of the controller (described in Pseudocode 7) and for incorporating the state of the controller into the generated control inputs. The update of the controller state can be performed independently of the generation of the control inputs. Therefore, the controller performs the state update in parallel with system

Pseudocode 7 Computing the control input of the dynamic controller using the Montgomery multiplication and the Montgomery exponentiation.

Parameters
- N Paillier public key
- R Montgomery radix
- n' Number of bits in mapping from fixed point numbers $\mathbb{Q}(n, m)$ to integers $\mathbb{Z}_{2^{n'}}$
- $\hat{C}$ Controller matrix
- $\hat{D}[k]$ Controller matrix

Inputs
- $\check{y}$ Encrypted system outputs in Montgomery form $\check{y}_1, ..., \check{y}_{n_y}$
- $\check{s}$ Encrypted setpoints in Montgomery form $\check{s}_1, ..., \check{s}_{n_y}$
- $\check{x}$ Encrypted controller state in Montgomery form $\check{x}_1, ..., \check{x}_{n_x}$

Outputs
- $\check{u}$ Encrypted control inputs in Montgomery form $\check{u}_1, ..., \check{u}_{n_u}$
- $\check{e}$ Encrypted error values in Montgomery form $\check{e}_1, ..., \check{e}_{n_y}$

function GENERATECONTROL($\check{y}, \check{s}, \check{x}$)
  for $i = 1, ..., n_y$ **do** ▷ Generate encrypted error values
    $temp \leftarrow \text{MONTEXP}(\check{y}, 2^{n'} - 1)$
    $\check{e}_i \leftarrow \text{MONTMULT}[M = N^2](temp, \check{s})$
  end for
  for $i = 1, ..., n_u$ **do** ▷ Generate encrypted scalar products
    for $j = 1, ..., n_x$ **do**
      $temp_{ij} \leftarrow \text{MONTEXP}(\check{x}_j, \hat{C}_{ij})$
    end for
    for $j = 1, ..., n_y$ **do**
      $temp_{i(j+n_x)} \leftarrow \text{MONTEXP}(\check{e}_j, \hat{D}[k]_{ij})$
    end for
  end for
  for $i = 1, ..., n_u$ **do** ▷ Homomorphically sum up encrypted scalar products
    for $j = 2, ..., n_x + n_y$ **do**
      $temp_{i1} \leftarrow \text{MONTMULT}[M = N^2](temp_{i1}, temp_{ij})$
    end for
    $\check{u}_i \leftarrow temp_{i1}$
  end for
  return $(\check{u}, \check{e})$
end function

control input decryption in the plant interface. This task is labelled by "update controller state" in Figure 2, and is described in Pseudocode 8. The additional matrix multiplications are performed and can be parallelized similarly to the static controller case. The modular exponentiations can also be performed in parallel, and the results are multiplied together afterwards in a binary tree structure with a latency of $\lceil \log_2(n_x + n_y) \rceil$ times the latency of the Montgomery multiplication.

For both the static and dynamic controller cases, the controller requires at least one Montgomery exponentiator module; all tasks can operate with just a single Montgomery exponentiator if their steps are executed sequentially. If the static controller's tasks are to be fully parallelized, then it requires $n_y n_u$ Montgomery exponentiator modules, to perform the scalar multiplication in the matrix products on ciphertexts. For fully parallelizing the dynamic controller, we require $(n_x + n_y) \max(n_x, n_u)$ Montgomery exponentiation modules.

IV. EXPERIMENT

To demonstrate the system, we have implemented encrypted balance control of an inverted pendulum using our plant interface and controller digital designs on an FPGA. Inverted pendulum systems are unstable and require a dynamic controller to be robustly stabilized. We use the Quanser QUBE-Servo 2 as the plant and the Terasic C5P Development Board (equipped with the Cyclone V GX 5CGXFC9D6F27C7 FPGA) to implement the plant interface and the encrypted controller. The setup is shown in Figure 3.

We use the following dynamic controller with a control sampling frequency of $500\,\text{Hz}$ to stabilize the inverted pen-

Pseudocode 8 Update of the controller state of dynamic controller using the Montgomery multiplication and the Montgomery exponentiation.

Parameters

N	Paillier public key
R	Montgomery radix
n'	Number of bits in mapping from fixed point numbers $\mathbb{Q}(n,m)$ to integers $\mathbb{Z}_{2^{n'}}$
T	Controller reset period
$\hat{A}$	Controller matrix
$\hat{B}[k]$	Controller matrix

Inputs

$\check{x}$	Encrypted old controller state in Montgomery form $\check{x}_1, ..., \check{x}_{n_x}$
$\check{e}$	Encrypted error values in Montgomery form $\check{e}_1, ..., \check{e}_{n_y}$
k	Timestep of old controller state $\bmod T$

Outputs

$\check{x}'$	Encrypted new controller state in Montgomery form $\check{x}'_1, ..., \check{x}'_{n_x}$

function UPDATESTATE($\check{x}, \check{e}, k$)
  if $k = 0$ **then** ▷ Controller reset
    for $i = 1, ..., n_x$ **do**
      $\check{x}'_i \leftarrow R \bmod N^2$ ▷ Encrypted value of 0, in Montgomery form
    end for
  else
    for $i = 1, ..., n_x$ **do** ▷ Generate encrypted scalar products
      for $j = 1, ..., n_x$ **do**
        $temp_{ij} \leftarrow \text{MONTEXP}(\check{x}'_j, \hat{A}_{ij})$
      end for
      for $j = 1, ..., n_y$ **do**
        $temp_{i(j+n_x)} \leftarrow \text{MONTEXP}(\check{e}, \hat{B}[k]_{ij})$
      end for
    end for
    for $i = 1, ..., n_x$ **do** ▷ Homomorphically sum up encrypted scalar products
      for $j = 2, ..., n_x + n_y$ **do**
        $temp_{i1} \leftarrow \text{MONTMULT}[M = N^2](temp_{i1}, temp_{ij})$
      end for
      $\check{x}'_i \leftarrow temp_{i1}$
    end for
  end if
  return $\check{x}'$
end function

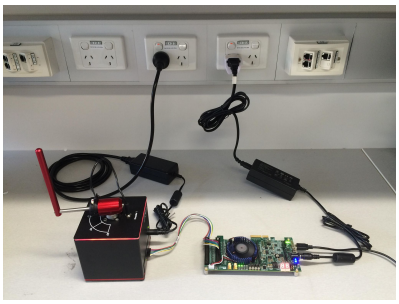

Fig. 3. Inverted pendulum balance control experimental setup.

dulum:

$$\mathcal{C}: \; x[k+1] = \frac{125\pi}{3072} \begin{bmatrix} 500 \\ 0 \\ 625 \end{bmatrix}^T (s[k] - y[k]), \tag{13a}$$

$$u[k] = x[k] + \frac{125\pi}{3072} \begin{bmatrix} -500 \\ -2 \\ -655 \end{bmatrix}^T (s[k] - y[k]), \tag{13b}$$

$$s[k] = \begin{bmatrix} 0 \\ \theta_s[k] \\ 1024 \end{bmatrix}, \quad y[k] = \begin{bmatrix} \theta[k] \\ \theta[k] \\ \alpha[k] \end{bmatrix}, \tag{13c}$$

where θ is the measured rotational arm angle, θ_s is the rotational arm angle setpoint, and α is the measured pendulum angle, all in encoder counts (with 2048 encoder counts measured per revolution). The resulting control input u is a number between -999 and 999, representing a duty cycle and direction. We implement this controller using $n' = 32$ bits, $m = 7$ bits, and an encryption key length of 256 bits. In Section II, as there were no assumptions on the integer or fractional nature of the parameters, all parameters were multiplied by 2^m to generate equivalent integer numbers. However, in this experiment, the sensor measurements and the C matrix are already integers, so we use the following substitutions in our encrypted system: $\hat{s}_i[k] = \bar{s}_i[k] \bmod 2^{32}$, $\hat{y}_i[k] = \bar{y}_i[k] \bmod 2^{32}$, $\hat{B}_{ij} = 2^7 \bar{B}_{ij} \bmod 2^{32}$, $\hat{C}_{ij} = \bar{C}_{ij} \bmod 2^{32}$, $\hat{D}_{ij} = 2^7 \bar{D}_{ij} \bmod 2^{32}$, $\hat{x}_i[k] = 2^7 \bar{x}_i[k] \bmod 2^{32}$, and $\hat{u}_i[k] = 2^7 \bar{u}_i[k] \bmod 2^{32}$.

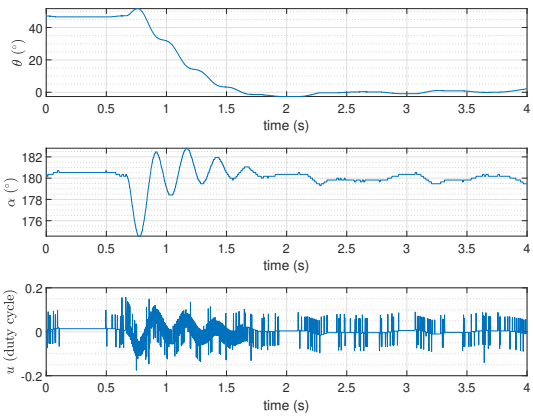

Fig. 4. The inverted pendulum system stabilizing to its setpoint. Note that the control input duty cycle is signed to specify the direction of rotation for the motor.

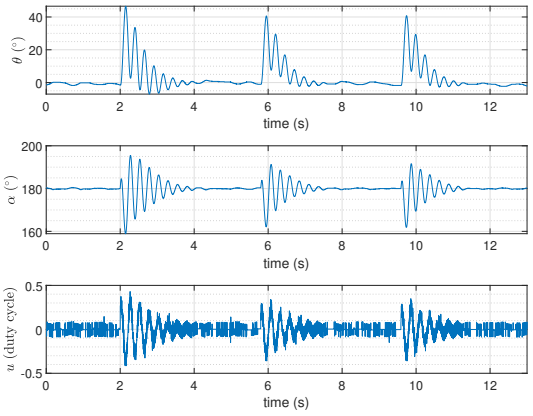

Fig. 5. The inverted pendulum system with disturbances introduced at the tip of the pendulum. Note that the control input duty cycle is signed to specify the direction of rotation for the motor.

Since the state of the contrller is simply a one step delay used to calculate velocities from position measurements by first order, there is no state evolution, and consequently resetting the controller state is not required. Rounding and clamping of the generated control input is performed externally from the plant interface and controller.

We utilize the Montgomery multiplier design in Pseudocode 2, which has an embedded multiplier usage that scales linearly with encryption key length. We run two Montgomery multipliers in parallel in each Montgomery exponentiator, and run a single Montgomery exponentiator in the plant interface and controller modules. We neglect the generation of random numbers, but still calculate a number to the power N in each control sampling period. We also neglect instantiating a separate module to encrypt setpoints, and instead encrypt setpoint in the controller, without the use of random numbers. Neither of these simplifications affect the synthesis, timing, or synthesis of the digital design, as the random number generation can be done outside of the digital engine using commercially available integrated circuits for random number generation, and the encryption

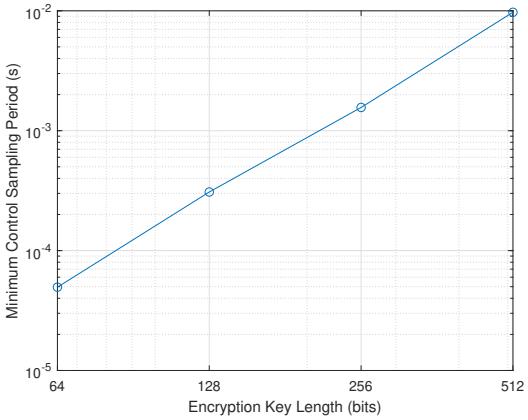

Fig. 6. Graph depicting how the minimum control sampling period increases with greater security.

of setpoints with random numbers can occur in parallel with the encryption of system outputs, thus not extending the minimum control sampling period. Importantly, on the FPGA we have distinct plant interface and controller modules and use an abstracted network to communicate encrypted data between them.

Figure 7 shows the hardware resource usage of the plant interface module as the encryption key length increases, for our implementation. Figure 6 shows the minimum control sampling period as the encryption key length increases from 64 bits to 512 bits, which affects the speed with which physical systems can be controlled. For the key length of 512 bits, the sampling time of system is 10 ms. Implementations using other Montgomery multiplier architectures can potentially result in completely different hardware resource usages and speeds. Such issue are the topic of future work.

Figure 4 shows the system behaviour converging to its setpoint. Figure 5 shows the system behaviour when disturbances are introduced at the tip of the pendulum. Evidently, the controller successfully attenuates large disturbances (of peak magnitude of twenty degrees).

In the experiments, we found that the latency of the plant interface determines the maximum control sampling frequency. This is due to Montgomery exponentiations with the large exponents N and λ, which require more Montgomery multiplications compared to the Montgomery exponentiations in the controller, where the exponents are shorter. If a larger control sampling frequency is required, then the plant interface digital design could make use of the Chinese Remainder Theorem [58] to reduce the size of the modulus in Montgomery exponentiations, speeding up each calculation.

The hardware description language (HDL) code used for synthesizing the encryption, secure controller, and decryption in the experiment can be found at `https://github.com/availn/EncryptedControl`. A video of the experiment can also be found at `https://youtu.be/ATM0tcecst0`.

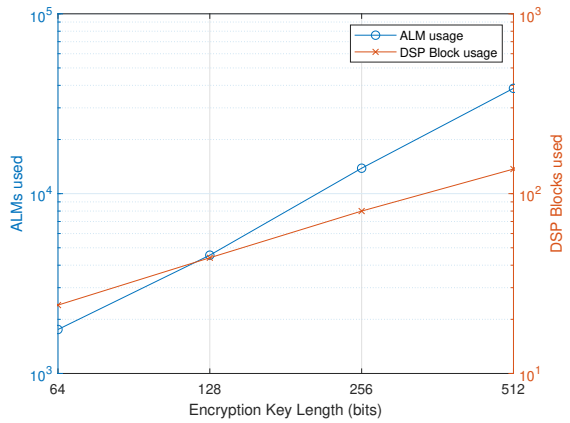

Fig. 7. Graph depicting how the usage of hardware resources in the plant interface increases with greater security.

V. Conclusions and Future Work

We presented an experimental setup to demonstrate a powerful framework for encrypted dynamic control of unstable systems using digital designs on FPGAs with deterministic latency. The framework is scalable and can be applied to large-scale cyber-physical systems. Future work includes investigation of methods for speeding up the computations and studying the effect of uncertain communication systems on the performance of the system.

References

[1] A. Teixeira, K. C. Sou, H. Sandberg, and K. H. Johansson, "Secure control systems: A quantitative risk management approach," *Control Systems, IEEE*, vol. 35, no. 1, pp. 24–45, 2015.

[2] L. Yang, X. Chen, J. Zhang, and H. V. Poor, "Cost-effective and privacy-preserving energy management for smart meters," *IEEE Transactions on Smart Grid*, vol. 6, no. 1, pp. 486–495, 2015.

[3] J. Qi, Y. Kim, C. Chen, X. Lu, and J. Wang, "Demand response and smart buildings: A survey of control, communication, and cyber-physical security," *ACM Transactions on Cyber-Physical Systems*, vol. 1, no. 4, p. 18, 2017.

[4] R. Dong, L. J. Ratliff, A. A. Cárdenas, H. Ohlsson, and S. S. Sastry, "Quantifying the utility–privacy tradeoff in the internet of things," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 2, p. 8, 2018.

[5] A. Teixeira, I. Shames, H. Sandberg, and K. H. Johansson, "A secure control framework for resource-limited adversaries," *Automatica*, vol. 51, pp. 135–148, 2015.

[6] W. Wang and Z. Lu, "Cyber security in the smart grid: Survey and challenges," *Computer Networks*, vol. 57, no. 5, pp. 1344–1371, 2013.

[7] J. Wan, A. Lopez, and M. A. A. Faruque, "Physical layer key generation: Securing wireless communication in automotive cyber-physical systems," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, p. 13, 2018.

[8] S. C. Patel, G. D. Bhatt, and J. H. Graham, "Improving the cyber security of SCADA communication networks," *Communications of the ACM*, vol. 52, no. 7, pp. 139–142, 2009.

[9] G. F. Franklin, M. L. Workman, and D. Powell, *Digital Control of Dynamic Systems*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[10] F. Farokhi, I. Shames, and N. Batterham, "Secure and private control using semi-homomorphic encryption," *Control Engineering Practice*, vol. 67, pp. 13–20, 2017.

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

[12] K. Aström and R. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. NJ, USA: Princeton University Press, 2010.

[13] M. Green and D. J. N. Limebeer, *Linear Robust Control*, ser. Dover Books on Electrical Engineering. NY, USA: Dover Publications, Incorporated, 2012.

[14] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[15] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, July 1985.

[16] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[17] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, June 1996.

[18] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178.

[20] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin: Springer, 2010, pp. 24–43.

[21] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[22] J. Yu, K. Wang, D. Zeng, C. Zhu, and S. Guo, "Privacy-preserving data aggregation computing in cyber-physical social systems," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 1, p. 8, 2018.

[23] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 1219–1234.

[24] C. Gentry, "Computing arbitrary functions of encrypted data," *Communications of the ACM*, vol. 53, no. 3, pp. 97–105, 2010.

[25] Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang, "Scalable and secure logistic regression via homomorphic encryption," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '16. New York, NY, USA: ACM, 2016, pp. 142–144.

[26] F. Li, B. Luo, and P. Liu, "Secure information aggregation for smart grids using homomorphic encryption," in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. NJ, USA: IEEE, 2010, pp. 327–332.

[27] F. Farokhi, I. Shames, and K. H. Johansson, "Private and secure coordination of match-making for heavy-duty vehicle platooning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7345–7350, 2017.

[28] Y. Shoukry, K. Gatsis, A. Alanwar, G. J. Pappas, S. A. Seshia, M. Srivastava, and P. Tabuada, "Privacy-aware quadratic optimization using partially homomorphic encryption," in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. NJ, USA: IEEE, 2016, pp. 5053–5058.

[29] K. Kogiso and T. Fujita, "Cyber-security enhancement of networked control systems using homomorphic encryption," in *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*. NJ, USA: IEEE, 2015, pp. 6836–6843.

[30] J. Kim, C. Lee, H. Shim, J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Encrypting controller using fully homomorphic encryption for security of cyber-physical systems," *IFAC-PapersOnLine*, vol. 49, no. 22, pp. 175–180, 2016.

[31] K. Kogiso, "Upper-bound analysis of performance degradation in encrypted control system," in *2018 Annual American Control Conference (ACC)*. NJ, USA: IEEE, 2018, pp. 1250–1255.

[32] P. Tam and J. Moore, "Stable realization of fixed-lag smoothing equations for continuous-time signals," *IEEE Transactions on Automatic Control*, vol. 19, no. 1, pp. 84–87, February 1974.

[33] J. Moore, "Fixed-lag smoothing results for linear dynamical systems," *Australian Telecommunications Research*, vol. 7, no. 2, pp. 16–21, 1973.

[34] C. Murguia, F. Farokhi, and I. Shames, "Secure and private implementation of dynamic controllers using semi-homomorphic encryption," 2018, preprint. arXiv preprint arXiv:1812.04168.

[35] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, ser. SFCS '82. Washington, DC, USA: IEEE Computer Society, 1982, pp. 160–164.

[36] Y. Lindell and B. Pinkas, "A proof of security of Yaos protocol for two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[37] ——, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin: Springer, 2007, pp. 52–78.

[38] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *International Colloquium on Automata, Languages, and Programming*. Berlin, Heidelberg: Springer, 2008, pp. 486–498.

[39] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-gate secure computation with malicious adversaries," in *USENIX Security Symposium*, vol. 12. CA, USA: USENIX, 2012, pp. 285–300.

[40] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88. New York, NY, USA: ACM, 1988, pp. 11–19.

[41] L. Kamm and J. Willemson, "Secure floating point arithmetic and private satellite collision analysis," *International Journal of Information Security*, vol. 14, no. 6, pp. 531–548, 2015.

[42] I. San, N. At, I. Yakut, and H. Polat, "Efficient Paillier cryptoprocessor for privacy-preserving data mining," *Security and Communication Networks*, vol. 9, no. 11, pp. 1535–1546, 2016.

[43] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, ser. Chapman & Hall/CRC Cryptography and Network Security Series. Boca Raton, FL: CRC Press, 2014.

[44] M. Bellare and P. Rogaway, "Optimal asymmetric encryption," in *Advances in Cryptology — EUROCRYPT'94*, A. De Santis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 92–111.

[45] G. C. Goodwin, S. F. Graebe, and M. E. Salgado, *Control System Design*. NJ, USA: Prentice Hall, 2001.

[46] G. Hachez and J.-J. Quisquater, "Montgomery exponentiation with no final subtractions: Improved results," in *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 293–301.

[47] G. C. T. Chow, K. Eguro, W. Luk, and P. Leong, "A Karatsuba-based Montgomery multiplier," in *2010 International Conference on Field Programmable Logic and Applications*. NJ, USA: IEEE, Aug 2010, pp. 434–437.

[48] A. Le Masle, W. Luk, J. Eldredge, and K. Carver, "Parametric encryption hardware design," in *Reconfigurable Computing: Architectures, Tools and Applications*, P. Sirisuk, F. Morgan, T. El-Ghazawi, and H. Amano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 68–79.

[49] A. Daly and W. P. Marnane, "Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic," in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM, 01 2002, pp. 40–49.

[50] N. Mentens, "Secure and efficient coprocessor design for cryptographic applications on FPGAs," 2019, phD Thesis, https://www.esat.kuleuven.be/cosic/publications/thesis-131.pdf.

[51] G. Sassaw, C. J. Jimenez, and M. Valencia, "High radix implementation of Montgomery multipliers with CSA," in *2010 International Conference on Microelectronics*. Piscataway, NJ USA: IEEE, Dec 2010, pp. 315–318.

[52] F. Bernard, "Scalable hardware implementing high-radix montgomery multiplication algorithm," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 117–126, Feb. 2007.

[53] C. Baetoniu, "High speed true random number generators in Xilinx FPGAs," 2016, [Online]. http://forums.xilinx.com/xlnx/attachments/xlnx/EDK/27322/1/HighSpeedTrueRandomNumberGeneratorsinXilinxFPGAs.pdf.

[54] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA-based true random number generation using circuit metastability with adaptive feedback control," in *Cryptographic Hardware and Embedded Systems – CHES 2011*, B. Preneel and T. Takagi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 17–32.

[55] D. Schellekens, B. Preneel, and I. Verbauwhede, "FPGA vendor agnostic true random number generator," in *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*. Piscataway, NJ, USA: IEEE, 2006, pp. 1–6.

[56] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM J. Comput.*, vol. 15, pp. 364–383, 05 1986.

[57] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," *SIAM Journal on Computing*, vol. 13, pp. 850–864, 01 1984.

[58] C. Ding, D. Pei, and A. Salomaa, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1996.