University Library



A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Tran, J;Farokhi, F;Cantoni, M;Shames, I

Title:
Implementing homomorphic encryption based secure feedback control

Date:
2020-04

Citation:
Tran, J., Farokhi, F., Cantoni, M. & Shames, I. (2020). Implementing homomorphic encryption based secure feedback control. Control Engineering Practice, 97, pp.104350-104350. https://doi.org/10.1016/j.conengprac.2020.104350.

Persistent Link:
https://hdl.handle.net/11343/251368

Implementing Homomorphic Encryption Based Secure Feedback Control[☆]

Julian Tran[a], Farhad Farokhi[b,c], Michael Cantoni[b], Iman Shames[b]

[a] *BAE Systems, Williamstown, VIC 3016, Australia*
[b] *The University of Melbourne, Parkville, VIC 3010, Australia*
[c] *CSIRO's Data61, Docklands, VIC 3008, Australia*

Abstract

This paper is about an encryption based approach to the secure implementation of feedback controllers for physical systems. Specifically, Paillier's homomorphic encryption is used to digitally implement a class of linear dynamic controllers, which includes the commonplace static gain and PID type feedback control laws as special cases. The developed 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 an inverted pendulum controller; as this is an unstable plant, the control is necessarily fast.

Keywords: secure control, homomorphic encryption, digital design, FPGA.

1. Introduction

1.1. Motivation

Advances in communication, control, and computer engineering have enabled the 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, 2, 3, 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 spatially distributed 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 grid. The same also goes for the actuator. 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 transmit these 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 each new sample can be processed by the controller, it must process the previous one, compute the control inputs, and transmit the control inputs to the actuators. Therefore, the sampling rate of the sensors cannot be faster than the inverse of the worst-case delay/latency caused by the required computations and communications. On the other hand, in order to guarantee stability and per-

[☆] Corresponding author F. Farokhi.
Email addresses: julian.tran@unimelb.edu.au (Julian Tran), farhad.farokhi@unimelb.edu.au (Farhad Farokhi), farhad.farokhi@data61.csiro.au (Farhad Farokhi), cantoni@unimelb.edu.au (Michael Cantoni), ishames@unimelb.edu.au (Iman Shames)

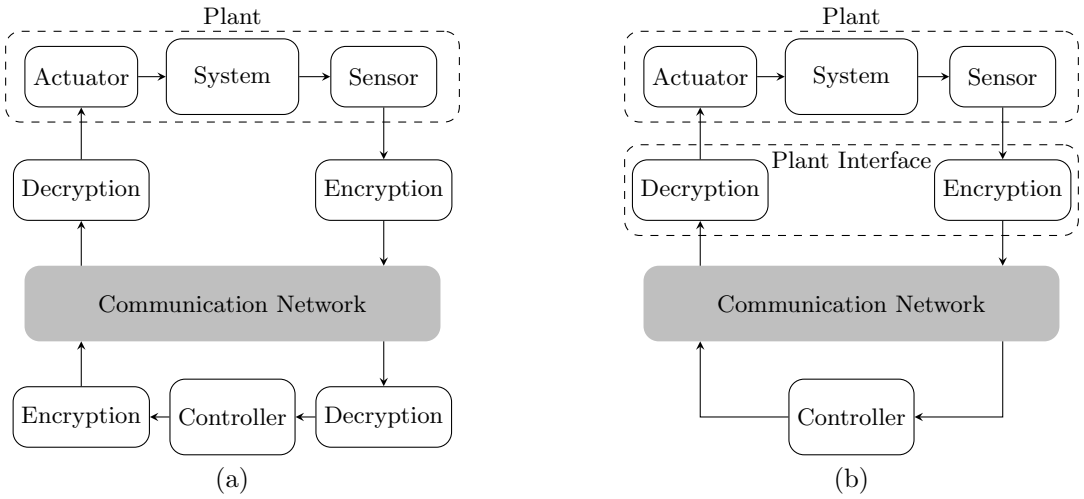


Figure 1: The schematic diagram of a networked control system with (a) normal encryption and (b) semi-homomorphic encryption-decryption units.

formance 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 controlled systems dynamics need to be) [9].

In [10], a general purpose microprocessor based system, 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 tight control on the timing is unfortunately not always possible on general purpose microprocessor based systems with operating systems because computations and their timings are subject to the operating system scheduling. Even without an operating system, the time sequential nature of software implementations for execution on a general purpose processor can be limiting from the perspective of achievable sampling rate. 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.

1.2. Contributions

In this paper, we use homomorphic encryption, specifically the Paillier encryption scheme [11], to implement linear control laws. This includes many popular control laws, such as static gain [12], proportional-integral-derivative (PID) control [13] and linear quadratic regulators (LQR) [12]. Linear control laws, such as PID controllers, have been heavily used within the industry for regulating nonlinear physical systems and are of practical relevance [14]. Although the paper presents the digital system implementation within the context of Paillier encryption, the underlying methodology is applicable, in principle, to other homomorphic encryption methods that rely on the exponentiation of large integer numbers, such as RSA and ElGamal encryption [15, 16]. After the quantization and transformation of the controllers for implementation on ciphertexts, modular multipliers and exponentiators are implemented using Montgomery multiplication [17, 18]. These modules can be used in parallel for encryption, controller computations, and decryption. We analyze the timing of an FPGA realization of such an implantation of a feedback controller, and present experimental results for the control of an unstable system, namely, an inverted pendulum.

Table 1: Summary of popular homomorphic encryption methods.

RSA [15]	multiplicatively homomorphic
ElGamal [16]	multiplicatively homomorphic
Paillier [11]	additively homomorphic
Benaloh [19]	additively homomorphic
Gentry [20]	fully homomorphic
LWE [21]	fully homomorphic
BGN [22]	somewhat homomorphic

1.3. Related Studies

The study of homomorphic encryption, a form of encryption that enables computations to be carried out on the encrypted data, dates back to the pioneering result of [23] after observing semi-homomorphic properties in RSA [15]. Semi-homomorphic encryption only allows for a smaller number of operations to be performed on

the encrypted data in contrast with fully homomorphic encryption. For example, in the case of RSA and ElGamal encryption [16] multiplication of plaintext data corresponds to multiplication of encrypted data, and in the case of Paillier encryption [11] summation of plaintext data corresponds to multiplication of encrypted data. The Gentry encryption scheme [20] 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., [24, 25]. The computational burden of fully-homomorphic encryption methods is often much greater than that of semi-homomorphic encryption methods. A summary of some popular homomorphic encryption methods is presented in Table 1.

Homomorphic encryption has been used previously for third-party cloud-computing services [26, 27, 28, 29, 30, 31]. More recent studies [32, 10, 33, 34, 35] 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 can grow linearly with the number of iterations. This renders the memory of such control laws useless after a certain number iterations due to an overflow or an underflow[1]. We borrow theoretical results from [36, 37, 38, 39] to propose a finite-memory implementation of dynamic controllers over ciphertexts. It should be noted that this paper is distinct from all the earlier studies on the use of homomorphic encryption for control, particularly [10, 39, 32, 10, 33, 34, 35], as the focus is on issues pertaining to the practical implementation of the control laws in real-time with the aid of a customized digital engine rather than developing theoretical guarantees based on idealized computational models.

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 for secure two-party computations [40]. 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 [41] and efficiently implementable for Boolean functions [42]. 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 [43], is not trivial [44]. 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 [45, 46]. Application of secret sharing to general problems is difficult and the digital design becomes problem specific to the application.

Finally, note that the Paillier encryption scheme has been recently implemented on FPGAs in [47]; however, that paper considered the problem of privacy-preserving data mining, which has different requirements in comparison to real-time encrypted control. This difference in requirements resulted in the consideration of a different implementation architecture in this paper. In particular, the binomial expansion for the specific choice of the exponential base is exploited to achieve fast encryption in this paper. Further, there are differences between the operations required for data mining and controller computation.

1.4. Paper Outline

The rest of the paper is organized as follows. In Section 2, 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 3, the digital design for FPGA realization is described. We present the experimental results for the control of an inverted pendulum in Section 4. Finally, we conclude the paper and present avenues for future research in Section 5.

2. Secure Feedback Control

In this section, we discuss encryption, decryption, and controller blocks of the networked control systems in Figure 1 (b).

2.1. Feedback Controller

In this paper, we consider dynamic controllers of the following form:

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

$$u[k] = Cx[k], \quad (1b)$$

where $x[k] \in \mathbb{R}^{n_x}$ is the controller state, $u[k] \in \mathbb{R}^{n_u}$ is the vector of control inputs to the physical system, $y[k] \in \mathbb{R}^{n_y}$ is the vector of plant outputs, and T is the number of time steps between controller state resets. Conditions for selecting T with stability and performance guarantees are presented in [39]. The class of controllers in (1) covers static, reset integral, reset lead and lag controller. For instance, in the case of static controllers, $A = 0$, $B = I$, and

[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.

C is the static gain of the controller. Note that there is a delay of one sampling time between measurement and actuation, modelling computation and communication time associated with the networked controller. For static controllers, since the controller's state is not accumulative and only acts as a delay, we can set $T = \infty$ without concerns about state overflow or underflow. For reset proportional-integral (PI) controllers, $A = \text{diag}(1, 0)$ with $\text{diag}(a)$ denoting a diagonal matrix whose main diagonal is equal to a, $B = [\Delta t\ 1]^\top$ with $\Delta t > 0$ denoting the sampling time of the control system, and $C = [K_I\ K_p]$ with K_I and K_p denoting, respectively, the integral and proportional gains. Note that PI control laws have been heavily used within the industry for regulating/controlling nonlinear physical systems [14] and, therefore, the choice of linear dynamic controllers is of practical relevance. 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 in the plaintext domain, the numbers of the bits required for representing the fractional and integer parts of plaintext numbers continuously grow, and there is no simple way to truncate with small error when working in the encrypted domain). Resetting controllers have been previously studied in [36, 37, 38, 39].

As an alternative to the resetting controller in (1), we can decrypt the state of the encrypted controller, project it into the desired set of fixed-point rational numbers, and encrypt it again [48]. This approach creates unnecessary communication overhead and overburdens the computational units of the IoT device. In addition, the risk of a privacy or security breach increases by decrypting the state. Another approach can also be to restrict the controller parameters to integer numbers so that the state of the dynamic controller [49]. This makes the problem of designing the controller into a mixed-integer optimization problem, which can be computationally exhaustive. However, a robust control approach can be taken to ensure that converting non-integer controllers to integer ones does not compromise stability [49]. Upon following any of these two approaches, we can set $T = +\infty$ and the rest of the paper remains unchanged.

2.2. 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 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 these properties hold, the encryption scheme is called 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 Paillier [11], RSA [15], and ElGamal [16], are examples of semi-homomorphic encryption. 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., [26, 27, 28, 29, 30, 10, 39] 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 achieved. 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*

[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 [50].

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 [51]. 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, 16].

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. Paillier encryption works 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 [47]. 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.,

$$c_1 \oplus c_2 := (c_1 c_2) \bmod N^2, \tag{2a}$$

$$t \otimes c := c^t \bmod N^2. \tag{2b}$$

2.3. Secure Controller Implementation

The computations required to implement the controller in (1) are additions and multiplications. We restrict the controller input to fixed-point numbers and use the mapping from fixed point numbers to the integers from [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, 39], 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 manged by increasing the number of bits (specifically the number of integer bits) used to represent the underlying numbers [10, 39].

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

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

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

$$\bar{\mathcal{C}} : \bar{x}[k+1] = \begin{cases} \bar{A}\bar{x}[k] + \bar{B}(\bar{s}[k] - \bar{y}[k]), & \\ \qquad\qquad (k+1) \bmod T > 0, & \text{(3a)} \\ 0, \qquad\qquad (k+1) \bmod T = 0, & \end{cases}$$

$$\bar{u}[k] = \bar{C}\bar{x}[k], \tag{3b}$$

where $\bar{A}_{ij} = \mathcal{Q}(A_{ij})$, $\bar{B}_{ij} = \mathcal{Q}(B_{ij})$, $\bar{C}_{ij} = \mathcal{Q}(C_{ij})$, $\bar{s}_i[k] = \mathcal{Q}(s_i[k])$, and $\bar{y}_i[k] = \mathcal{Q}(y_i[k])$. We use the bar, e.g., $\bar{x}$, to denote the quantized version of any variable, e.g., x. The map from fixed point numbers to the integers $\mathbb{Z}_{2^{n'}}$ is borrowed from [10] to define

$$\hat{s}_i[k] = (2^m \bar{s}_i[k]) \bmod 2^{n'}, \tag{4a}$$

$$\hat{y}_i[k] = (2^m \bar{y}_i[k]) \bmod 2^{n'}, \tag{4b}$$

$$\hat{A}_{ij} = (2^m \bar{A}_{ij}) \bmod 2^{n'}, \tag{4c}$$

$$\hat{B}_{ij}[k] = (2^{(k \bmod T+1)m} \bar{B}_{ij}) \bmod 2^{n'}, \tag{4d}$$

$$\hat{C}_{ij} = (2^m \bar{C}_{ij}) \bmod 2^{n'}, \tag{4e}$$

$$\hat{x}_i[k] = (2^{(k \bmod T+1)m} \bar{x}_i[k]) \bmod 2^{n'}, \tag{4f}$$

$$\hat{u}_i[k] = (2^{(k \bmod T+2)m} \bar{u}_i[k+1]) \bmod 2^{n'}, \tag{4g}$$

where $n' = (n_x+1)T + n_u + n(T+2)$ to prevent overflows. Here, for simplicity, we assume that all scalar components of vectors use the same n and m, but these values can differ for various parts of the controller in general [10]. Then the

quantized controller can then be rewritten to operate on ciphertexts as

$$\tilde{\mathcal{C}}{:}\tilde{x}_i[k+1]=\begin{cases}\Big[\oplus_{j=1}^{n_x}(\hat{A}_{ij}\otimes\tilde{x}_j[k])\Big] & \\ \quad\oplus\Big[\oplus_{j=1}^{n_y}(\hat{B}_{ij}[k]\otimes(\tilde{s}_j[k]-\tilde{y}_j[k]))\Big], & (k+1)\bmod T>0,\\ \mathcal{E}(0,\kappa_p), & (k+1)\bmod T=0,\end{cases} \tag{5a}$$

$$\tilde{u}_i[k]=\oplus_{j=1}^{n_x}(\hat{C}_{ij}\otimes\tilde{x}_j[k]), \tag{5b}$$

where $\oplus,\otimes$ are defined in (2) and the tilde is used to denote the encrypted integers; i.e., $\tilde{u}_i[k]=\mathcal{E}(\hat{u}_i[k],\kappa_p)$, $\tilde{s}_j[k]=\mathcal{E}(\hat{s}_j[k],\kappa_p)$, $\tilde{y}_j[k]=\mathcal{E}(\hat{y}_j[k],\kappa_p)$, $\tilde{x}_j[k]=\mathcal{E}(\hat{x}_j[k],\kappa_p)$. Finally, the control signal at the actuator is computed by

$$\hat{u}_i[k]=\mathcal{D}(\tilde{u}_i[k],\kappa_p,\kappa_s)\bmod 2^{n'}, \tag{6a}$$

$$\bar{u}_i[k]=2^{-(k\bmod T+2)m}(\hat{u}_i[k]-2^{n'}\mathbb{1}_{\hat{u}_i[k]\geq 2^{n'-1}}), \tag{6b}$$

where $\mathbb{1}_p$ is equal to one if statement p holds and is equal to zero otherwise.

3. Digital Design

Timing is an important issue when implementing controllers in real-time. While the maximum computation to be performed by the controller is effectively the same in every iteration, implementations on a general purpose microprocessor based system are subject to variable timing performance dependent on operating system scheduling. Even without an operating system, the time sequential nature of software implementations for execution on a general purpose processor can be limiting from the perspective of achievable sampling rate. 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, thereby broadening the applicability of encryption based methods for securing feedback control systems. 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 [52]. 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.

Figure 2 illustrates the schematic diagram of the custom digital system for encrypted control discussed in this section. There are three major parts: encryption and decryption units, in the plant interface, and the physical system controller unit, accessed over a network. Each of these units includes a digital engine controller, which orchestrates data flow through the components of these systems, according to a corresponding algorithmic state machine. The activity of each major part is triggered by external events. Encryption is periodically triggered by the generation of samples of the plant output. Physical system controller and decryption unit activity is triggered by the arrival of data over the network.

In this section, we describe plant interface (encryption and decryption) and physical system controller blocks in Figure 2. Modular multiplication and modular exponentiation are important recurring elements in all of these blocks. Therefore, we start by describing these elemental building blocks in Subsection 3.1. We then describe the controller in Subsection 3.2 and the plant interface in Subsection 3.3.

Algorithm 1 [53] Modified Coarsely Integrated Operand Scanning (CIOS) method variant of Montgomery multiplication using 16 bits per word and without the final conditional subtraction.

Inputs
  M Odd modulus
  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 $R=2^{16(w+1)}$, $RR^{-1}\bmod M=1$
function MontMult(M,X,Y)
  Set w such that $M<2^{16w}$
  Set M' such that $MM'\bmod 2^{16}=2^{16}-1$
  $T\leftarrow 0$
  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

3.1. Modular Multiplication and Exponentiation

In many homomorphic encryption schemes, including Paillier encryption, efficient implementation of modular exponentiation is essential for fast encryption, decryption, and homomorphic operations; see Subsection 2.2. Within the context of secure feedback control implementation, the time it takes to perform encryption, decryption and homomorphic operations on cyphertexts, is a lower bound on the control loop sample 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

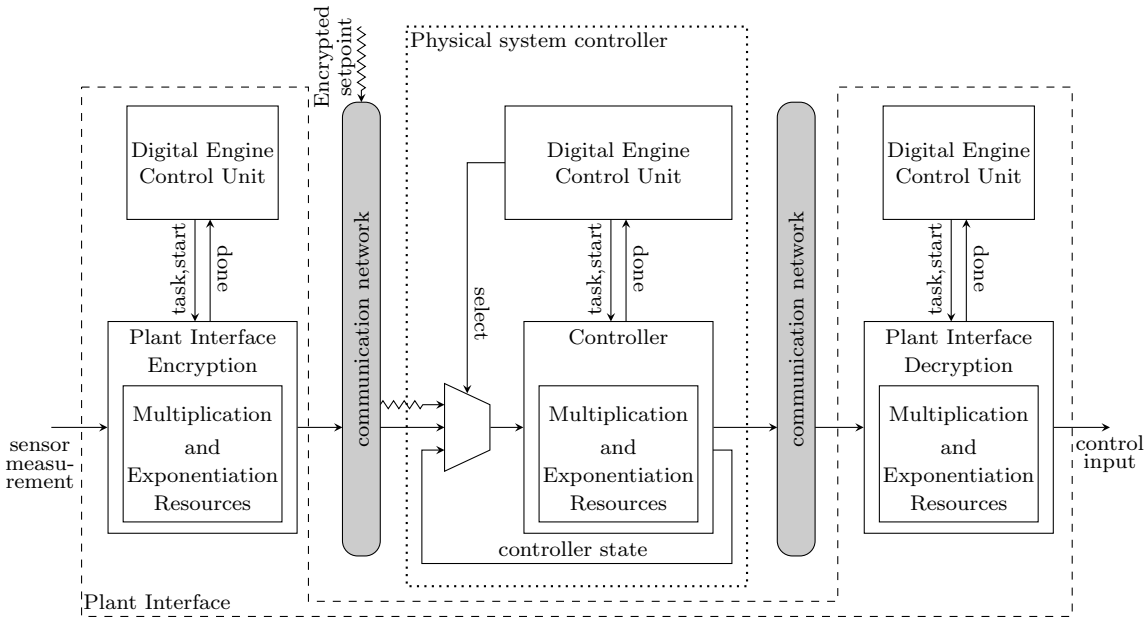


Figure 2: Schematic diagram of the custom digital system for encrypted control.

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

1: **Parameters**
2: N Paillier public key
3: R Montgomery radix
4: **Inputs**
5: B Base in Montgomery form $B = bR \bmod N^2$
6: E Exponent with l bits
7: **Outputs**
8: P Power in Montgomery form $P = b^E R \bmod N^2$
9: **function** MontExp(B, E)
10: $P \leftarrow R \bmod N^2$
11: **for** $i = 1, ..., l$ **do**
12: **if** $E \bmod 2 = 1$ **then**
13: $P \leftarrow$ MontMult(N^2, P, B)
14: **end if**
15: $E \leftarrow \lfloor E/2 \rfloor$
16: $B \leftarrow$ MontMult(N^2, B, B)
17: **end for**
18: **return** P
19: **end function**

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 Algorithm 2. 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 modular multiplication in line 13 in Algorithm 2 is performed in parallel to the modular multiplication that must be always performed in each iteration in line 16 in Algorithm 2. The right-to-left binary method for exponentiation involves calculating many sequential modular multiplications. The algorithm best suited for this purpose is Montgomery multiplication [17]. 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 Algorithm 1. 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, larger than M. In the right-to-left binary implementation of 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 [17]. The block diagram for a realization of the Montgomery exponentiator is illustrated in Figure 3.

Many hardware designs for computing Montgomery multiplications exist. A design involving the Karatsuba multiplication algorithm can be used to evaluate very large multiplications [54]. While this proved to be computationally effective in [54], 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

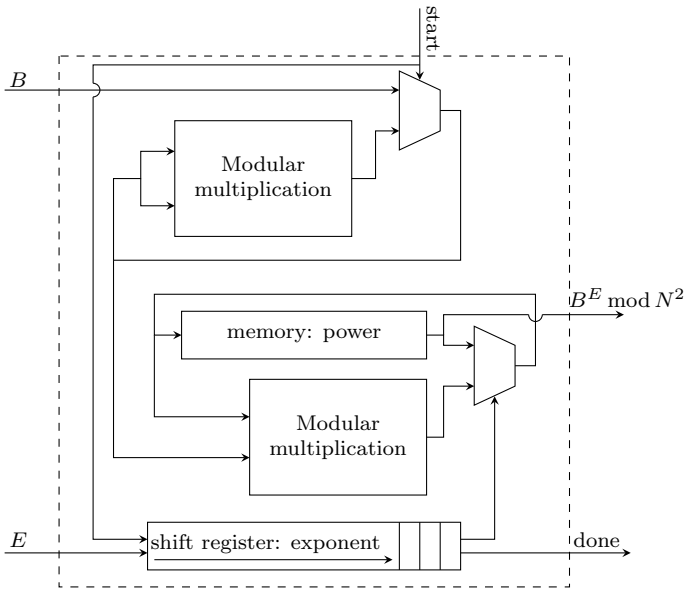


Figure 3: Block diagram of the Montgomery exponentiator using two modular Montgomery multipliers.

Coarsely Integrated Operand Scanning (CIOS) variant [18] with a word size of a single bit. Implementations of this algorithm are described in [55, 56]. The bitwise approach greatly simplifies the architecture of the Montgomery multiplier, as it is only required to perform additions and right shifts. However, the bitwise design cannot make use of the multi-bit word 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 [57, 58, 59]. 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 Algorithm 1, we borrow the modified CIOS method [53] 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. Algorithm 1 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 $\mathbb{Z}_M$. Such an output 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 [53].

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. 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.

Algorithm 3 Computing the control input using the Montgomery multiplication and 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
Inputs
  $\tilde{x}$ Encrypted controller state in Montgomery form $\tilde{x}_1, ..., \tilde{x}_{n_x}$
Outputs
  $\tilde{u}$ Encrypted control inputs in Montgomery form $\tilde{u}_1, ..., \tilde{u}_{n_u}$
function GENERATECONTROL($\tilde{y}, \tilde{s}, \tilde{x}$)
  for $i = 1, ..., n_u$ **do** ▷ Generate encrypted scalar products
    for $j = 1, ..., n_x$ **do**
      $\mathrm{v}_{ij} \leftarrow \textsc{MontExp}(\tilde{x}_j, \hat{C}_{ij})$
    end for
  end for
  for $i = 1, ..., n_u$ **do** ▷ Homomorphically sum up encrypted scalar products
    for $j = 2, ..., n_x$ **do**
      $\mathrm{v}_{i1} \leftarrow \textsc{MontMult}(N^2, \mathrm{v}_{i1}, \mathrm{v}_{ij})$
    end for
    $\tilde{u}_i \leftarrow \mathrm{v}_{i1}$
  end for
  return $\tilde{u}$
end function

In what follows, using the custom digital implementations of the Montgomery multipliers and the Montgomery exponentiators as the underlying arithmetic blocks, we design plant interface and physical system 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.

Parallelization is possible within the building blocks of the Montgomery multiplier and the Montgomery exponentiator, and also in the designs of the plant interface and controller. Adding parallelization increases the resource consumption of the hardware design, which is a limiting

Algorithm 4 Updating state of the dynamic controller using the Montgomery multiplication and 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
  $\tilde{s}$ Encrypted setpoints in Montgomery form $\tilde{s}_1, ..., \tilde{s}_{n_y}$
  $\tilde{x}$ Encrypted controller state in Montgomery form $\tilde{x}_1, ..., \tilde{x}_{n_x}$
Outputs
  $\tilde{x}'$ Encrypted controller state in Montgomery form $\tilde{x}'_1, ..., \tilde{x}'_{n_x}$
function UpdateState($\tilde{x}, \tilde{e}, k$)
  if $k+1 \bmod T = 0$ **then** ▷ Controller reset
    for $i = 1, ..., n_x$ **do**
      $\tilde{x}'_i \leftarrow R \bmod N^2$ ▷ Encrypted value of 0, in Montgomery form
    end for
  else
    for $i = 1, ..., n_y$ **do** ▷ Generate encrypted error values
      $\tilde{e}_i \leftarrow$ MontMult(N^2, MontExp($\tilde{y}, 2^{n'} - 1$), $\tilde{s}$)
    end for
    for $i = 1, ..., n_x$ **do** ▷ Generate encrypted scalar products
      for $j = 1, ..., n_x$ **do**
        $\mathrm{v}_{ij} \leftarrow$ MontExp($\tilde{x}'_j, \hat{A}_{ij}$)
      end for
      for $j = 1, ..., n_y$ **do**
        $\mathrm{v}_{i(j+n_x)} \leftarrow$ MontExp($\tilde{e}, \hat{B}[k]_{ij}$)
      end for
    end for
    for $i = 1, ..., n_x$ **do** ▷ Homomorphically sum up encrypted scalar products
      for $j = 2, ..., n_x + n_y$ **do**
        $\mathrm{v}_{i1} \leftarrow$ MontMult($N^2, \mathrm{v}_{i1}, \mathrm{v}_{ij}$)
      end for
      $\tilde{x}'_i \leftarrow \mathrm{v}_{i1}$
    end for
  end if
  return $\tilde{x}'$
end function

Algorithm 5 Encryption of the system outputs in the plant interface (or of the setpoints elsewhere) using the Montgomery multiplication and 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 $z_i = r_i^N \bmod N^2$
Outputs
  $\tilde{y}$ Encrypted system outputs in Montgomery form $\tilde{y}_1, ..., \tilde{y}_{n_y}$
function Encrypt($\hat{y}, z$)
  for $i = 1, ..., n_y$ **do**
    $\mathrm{v1} \leftarrow$ MontMult($N^2, NR \bmod N^2, \hat{y}_i$)
    $\mathrm{v2} \leftarrow$ MontMult($N^2, \mathrm{v1} + 1, R^2 \bmod N^2$)
    $\tilde{y}_i \leftarrow$ MontMult($N^2, z_i, \mathrm{v2}$)
  end for
  return $\tilde{y}$
end function

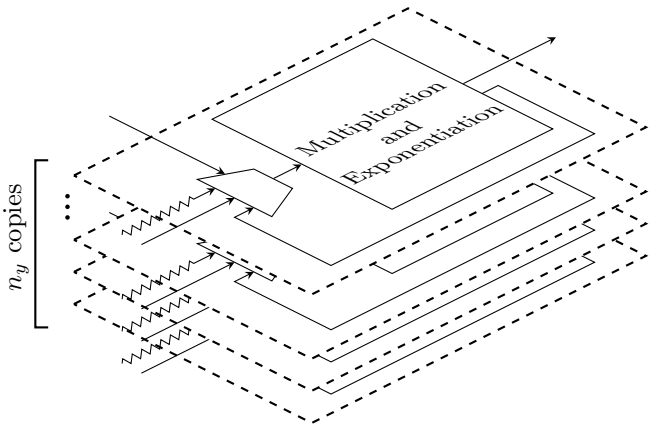


Figure 4: Block diagram of control computation using the Montgomery multiplication and exponentiation. These parallel blocks sit within the controller block in Figure 2.

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 for this purpose.

3.2. Controller Module Design

Consider dynamic controllers in (5). There are computations for incorporating the state of the controller into the generated control inputs, described in Algorithm 3, and for updating the state of the controller, described in Algorithm 4. The update of the controller state can be performed independently of the generation of the control inputs. Figure 4 illustrates the block diagram for a possible realization of Algorithms 3 and 4. Because calculating the control inputs are independent of each other, individual computations can all be performed in parallel using n_u copies of the multiplier and exponentiator. These parallelizations allow physical systems with more inputs and outputs to be controlled, without increasing the time re-

Algorithm 6 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 $z_i = r_i^N \bmod N^2$
function CalculateRandom(r)
  for $i = 1, ..., n_y$ **do**
    $z_i \leftarrow$ MontExp(r_i, N)
  end for
  return z
end function

quired 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 controller would require only one Montgomery exponentiator module. The matrix multiplications for updating the state of the controller can also be parallelized for each row by utilizing n_x copies of the multiplier and exponentiator. 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.

3.3. Plant Interface Module Design

The plant interface's role in the encrypted control system is to encrypt the plant 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 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.

Paillier encryption algorithm in Algorithm 5 requires values for $r^N \bmod N^2$ as inputs, which is independent of the plaintext being encrypted. The steps required for generating $r^N \bmod N^2$ are described in Algorithm 6. A block diagram similar to Figure 4 can be employed for parallel realization of the steps in Algorithm 6. 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.

Algorithm 7 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
  $\tilde{u}$ Encrypted control inputs in Montgomery form $\tilde{u}_1, ..., \tilde{u}_{n_u}$
Outputs
  $\hat{u}$ Control inputs $\hat{u}_1, ..., \hat{u}_{n_u}$
function Decrypt($\tilde{u}$)
  for $i = 1, ..., n_u$ **do**
    v1 $\leftarrow$ MontExp($\tilde{u}_i, \lambda$)
    v2 $\leftarrow$ MontMult(N^2, v1, 1)
    v3 $\leftarrow$ MontMult($N^2 + 2$, v2 $- 1, N^{-1}R^2 \bmod (N^2 + 2)$)
    v4 $\leftarrow$ MontMult($N^2 + 2$, v3, 1)
    v5 $\leftarrow$ MontMult(N, v4, $\mu R^2 \bmod N$)
    $\hat{u}_i \leftarrow$ MontMult(N, v5, 1) $\bmod 2^{n'}$
  end for
  return $\hat{u}$
end function

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 [60, 61, 62]. 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 [63, 64]. 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 Algorithm 6, 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 ex-

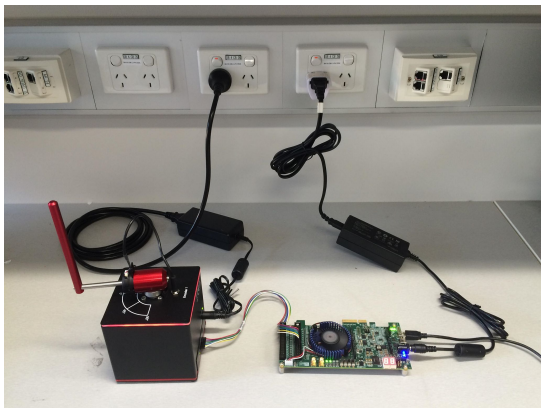


Figure 5: Inverted pendulum balance control experimental setup.

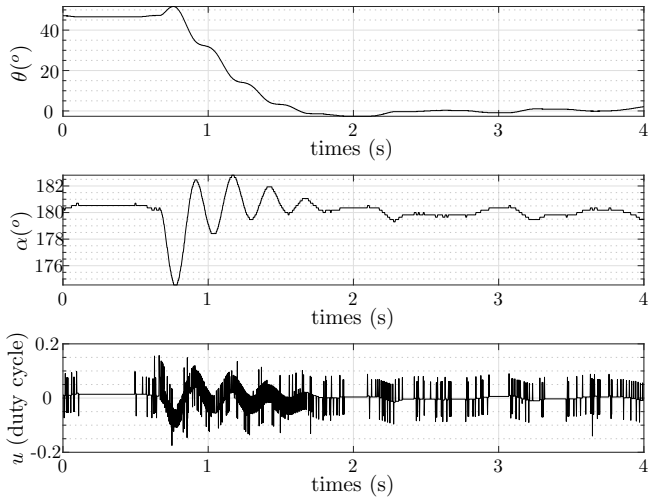


Figure 6: The inverted pendulum system stabilizing to its setpoint. Note that the control input duty cycle is signed to specify the direction of rotation for the motor.

ponentiation 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 Algorithms 5, 6, and 7, expressed as a collection 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. Every loop in Algorithms 5, 6, and 7 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. 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.

4. 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 5.

We use the following dynamic controller with a control sampling frequency of 500 Hz to stabilize the inverted pendulum:

$$\mathcal{C}: x[k+1] = \begin{bmatrix} 0_{3\times3} & 0_{3\times1} \\ \frac{125\pi}{3072}\begin{bmatrix}500 & 0 & 625\end{bmatrix} & 0 \end{bmatrix} x[k] + \begin{bmatrix} I_{3\times3} \\ 0_{1\times3} \end{bmatrix} (s[k] - y[k]) \tag{7a}$$

$$u[k] = \begin{bmatrix} \frac{125\pi}{3072}\begin{bmatrix}-500 & -2 & -655\end{bmatrix} & 1 \end{bmatrix} x[k], \tag{7b}$$

$$s[k] = \begin{bmatrix} 0 \\ \theta_s[k] \\ 1024 \end{bmatrix}, \quad y[k] = \begin{bmatrix} \theta[k] \\ \theta[k] \\ \alpha[k] \end{bmatrix}, \tag{7c}$$

where $\theta[k]$ is the measured rotational arm angle, $\theta_s[k]$ is the rotational arm angle setpoint, $\alpha[k]$ is the measured pendulum angle, all in encoder counts (with 2048 encoder counts measured per revolution), $0_{n\times m}$ is a matrix of zeros with n rows and m columns, and $I_{n\times n}$ is an identity matrix of size n. 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 2, 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:

$$\hat{s}_i[k] = \bar{s}_i[k] \bmod 2^{32} \tag{8a}$$

$$\hat{y}_i[k] = \bar{y}_i[k] \bmod 2^{32} \tag{8b}$$

$$\hat{B}_{ij} = 2^7 \bar{B}_{ij} \bmod 2^{32} \tag{8c}$$

$$\hat{C}_{ij} = \bar{C}_{ij} \bmod 2^{32} \tag{8d}$$

$$\hat{x}_i[k] = 2^7 \bar{x}_i[k] \bmod 2^{32} \tag{8e}$$

$$\hat{u}_i[k] = 2^7 \bar{u}_i[k] \bmod 2^{32} \tag{8f}$$

Since there is no state evolution (i.e., the state is a simple two steps delay to calculate velocities from position

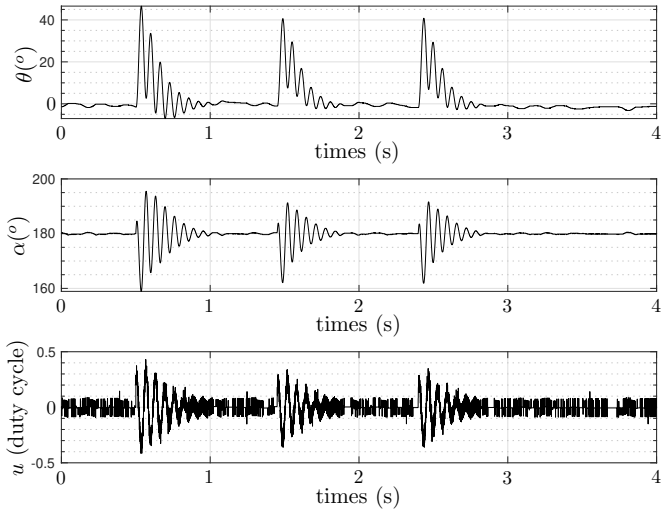


Figure 7: The inverted pendulum system with disturbances introduced at the tip of the pendulum. Note that the control input duty cycle is signed to specify the direction of rotation for the motor.

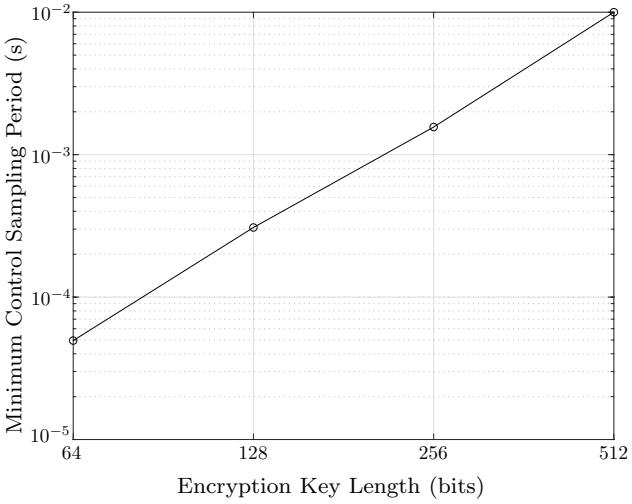


Figure 8: Graph depicting how the minimum control sampling period increases with greater security.

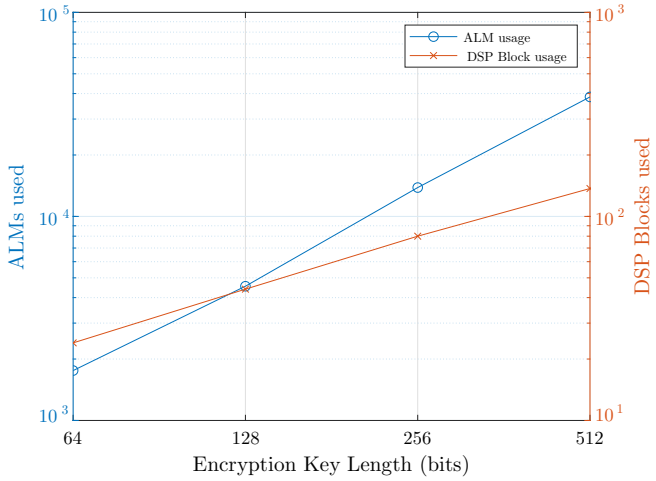


Figure 9: Graph depicting how the usage of hardware resources in the plant interface increases with greater security.

measurements by first order difference), a 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 Algorithm 1, 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 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 9 shows the hardware resource usage of the plant interface module as the encryption key length increases, for our implementation. Figure 8 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 6 shows the system behaviour converging to its setpoint. Figure 7 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 [65] 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, 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`.

Although computational times (in seconds) for microprocessor based implementations of homomorphic encryption are available in [31, 10], we have avoided presenting a such comparison with the FPGA implementation (which is faster than both [31, 10]). Comparing computational time is *ad hoc* and specific to hardware platform, e.g., clock frequency of the computer. The advantage of an FPGA implementation relates to the scope for parallelization and the composition of function across space, whereas microprocessor based implementation are limited to achieving complex functions sequentially across time. This of course can come at the expense of additional hardware resources. However, FPGA implementations also offer the ability to meet hard deadlines that are difficult to guarantee with micro-processor based systems governed by an operating system.

5. 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.

Acknowledgement

The work of I. Shames was, in part, supported by NATO Science for Peace and Security (SPS) Programme SPS.SFP G5479. The work of F. Farokhi was supported by the McKenzie Fellowship from the University of Melbourne. The work of J. Tran, M. Cantoni, and I. Shames was also supported by an ARC Linkage Grant LP160100666.

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*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 3rd ed., 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), pp. 223–238, Springer Berlin Heidelberg, 1999.

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

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

[14] K. J. Åström and T. Hägglund, "The future of pid control," *Control Engineering Practice*, vol. 9, no. 11, pp. 1163–1175, 2001.

[15] 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.

[16] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, pp. 469–472, July 1985.

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

[18] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, pp. 26–33, June 1996.

[19] J. Benaloh, "Dense probabilistic encryption," in *Proceedings of the Workshop on Selected Areas of Cryptography*, pp. 120–128, 1994.

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

[21] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Annual Cryptology Conference*, pp. 75–92, Springer, 2013.

[22] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Theory of Cryptography Conference*, pp. 325–341, Springer, 2005.

[23] 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.

[24] 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), pp. 24–43, Springer, 2010.

[25] 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.

[26] J. Yu, K. Wang, D. Zeng, C. Zhu, and S. Guo, "Privacy-preserving data aggregation computing in cyber-physical so-

cial systems," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 1, p. 8, 2018.

[27] 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*, STOC '12, (New York, NY, USA), pp. 1219–1234, ACM, 2012.

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

[29] 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*, CODASPY '16, (New York, NY, USA), pp. 142–144, ACM, 2016.

[30] 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), pp. 327–332, IEEE, 2010.

[31] 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.

[32] 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), pp. 5053–5058, IEEE, 2016.

[33] 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), pp. 6836–6843, IEEE, 2015.

[34] 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.

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

[36] 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, 1974.

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

[38] C. Prieur, I. Queinnec, S. Tarbouriech, and L. Zaccarian, "Analysis and synthesis of reset control systems," *Foundations and Trends in Systems and Control*, vol. 6, no. 2-3, pp. 117–338, 2018.

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

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

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

[42] Y. Lindell and B. Pinkas, "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), pp. 52–78, Springer, 2007.

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

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

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

[46] 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.

[47] 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.

[48] M. Zamani, L. Sadeghikhorrami, A. A. Safavi, and F. Farokhi, "Secure state estimation using semi-homomporphic encryption," in *Proceedings of the 23rd International Symposium on Mathematical Theory of Networks and Systems (MTNS)*, 2018.

[49] J. H. Cheon, K. Han, H. Kim, J. Kim, and H. Shim, "Need for controllers having integer coefficients in homomorphically encrypted dynamic system," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 5020–5025, 2018.

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

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

[52] S. A. Ward and R. H. Halstead, *Computation Structures*. MIT Engineering & Computer Science Series, New York, NY, USA: McGraw-Hill, 1990.

[53] 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), pp. 293–301, Springer Berlin Heidelberg, 2000.

[54] 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), pp. 434–437, IEEE, Aug 2010.

[55] 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), pp. 68–79, Springer Berlin Heidelberg, 2010.

[56] 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), pp. 40–49, ACM, 01 2002.

[57] 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`.

[58] 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), pp. 315–318, IEEE, Dec 2010.

[59] F. Bernard, "Scalable hardware implementing high-radix montgomery multiplication algorithm," *Journal of Systems Architecture*, vol. 53, pp. 117–126, Feb. 2007.

[60] 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`.

[61] 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), pp. 17–32, Springer Berlin Heidelberg, 2011.

[62] 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), pp. 1–6, IEEE, 2006.

[63] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM Journal on Computing*, vol. 15, pp. 364–383, 05 1986.
[64] 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.
[65] 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.