HETEROGENEOUS COMPUTATION AND EXPRESSION OPTIMIZATION FOR REAL-TIME HOMOMORPHICALLY ENCRYPTED ROBOT CONTROL

A Dissertation
Presented to
The Academic Faculty

By

Shane Adam Kosieradzki

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in the
Georgia W. Woodruff School of Mechanical Engineering
Department of Mechanical Engineering

Georgia Institute of Technology

April 2024

HETEROGENEOUS COMPUTATION AND EXPRESSION OPTIMIZATION FOR REAL-TIME HOMOMORPHICALLY ENCRYPTED ROBOT CONTROL

Thesis committee:

Dr. Jun Ueda
Mechanical Engineering
Georgia Institute of Technology

Dr. Shreyas Kousik
Mechanical Engineering
Georgia Institute of Technology

Dr. Aldo Ferri
Mechanical Engineering
Georgia Institute of Technology

Date approved: March 18, 2024

What we know is a drop, what we don’t know is an ocean.

Issac Newton

To all the teachers, mentors, and educators who helped me along the way.

ACKNOWLEDGMENTS

I would like to thank the members of my thesis committee for their help in preparation of this work. My advisor, Jun Ueda, for taking me into his group, teaching me control theory, facilitating countless opportunities, and helping me grow as a researcher.

I am honored to have worked with such talented friends and colleagues who made this work possible – Hyuk Bin Kwon was particularly invaluable and was my right-hand man during much of this research, thank you. To the many students I mentored, I am humbled by the insights they shared that expanded my own understanding.

Special thanks to my parents who always believed in and encouraged my education, and to my loving partner Kate Noel for her endless support and encouragement throughout my studies.

The author gratefully acknowledges the support for this work offered by the United States National Science Foundation and the Georgia Institute of Technology's Biorobotics and Human Modeling Laboratories. Any views and conclusions contained herein are those of the author, and do not necessarily represent the official positions, express or implied, of the funders.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

SUMMARY

Homomorphic Encryption is a relatively new cryptographic method which, unlike traditional encryption, allows computations to be preformed on encrypted data. Robotic controllers can take advantage of these new techniques to increase system security by encrypting the entire motion control scheme including: sensor signals, model parameters, feedback gains, and perform computation in the ciphertext space to generate motion commands without a security hole. However, numerous challenges exist which have limited the wide spread adoption of homomorphically encrypted control systems. The following thesis address several of these pressing issues–cryptographic overflow and heterogenous deployment.

Cryptographic overflow is a phenomenon intrinsic to homomorphic ciphers. As encrypted data is computed on the level of 'noise' inside the ciphertext increases, until it becomes too great making decryption impossible, this is known as 'overflow'. The primary contributor to noise growth is multiplication. Thus, this thesis explores topological sorting methods to find semantically equivalent but syntactically simpler control expressions. This allows an encrypted control scheme to preform the same calculation but with fewer multiplications, thus reducing the total amount of noise injected into the system.

Furthermore, encrypted calculations impose a hefty computational burden as compared to its unencrypted counterparts. As such, heterogeneous mix of different computing technologies (i.e. CPU, GPU, FPGA) are needed to achieve real-time signal processing. As such, this thesis explores which aspects of an encrypted control system is best suited for which computing technology and describes a deployment strategy to take advantage of these differences.

CHAPTER 1
INTRODUCTION

1.1 Background

In our modern society, virtually all devices are connected to network. Industry 4.0 [1] will revolutionize factory automation by taking advantage of today's information technology, transforming the conventional automation systems to efficient cyber-physical systems (CPS). Many modern automation systems are CPS which connect to network and tightly interact with remote devices. While the benefits are many, such a network configuration with frequent information exchange introduces security concerns [2, 3]. Cybersecurity of networked industrial automation systems is an emerging field [3, 4, 5].

While protection of CPS at the communication level has been extensively studied and implemented [6, 7], there is a void in the study of protection at a lower level, such as at the motion control level [8]. Motion control systems of interest in this project are a system consisting of force-generating components (actuators), measurement devices (sensors), a power electronic circuit (or driver), and an embedded microprocessor (motion controller) that receives commands from higher-level controllers and regulates actuator efforts based on implemented control algorithms.

It should be noted that while general low-level controllers must be designed carefully to ensure stability and required performance, the size of motion control software is usually small enough to be embedded in a microprocessor. This, in turn, indicates that motion control software is vulnerable to malicious system identification attacks if not appropriately protected. Allowing cyberattacks to a motion controller would result in: a) leaking of controller architecture, gains, and models, b) interception of motor commands and monitoring signals, and c) system disruption due to falsification of the controller. Minimal falsification

of a simple control scheme could easily modify its physical behavior. For example, power transmission equipment was infected with the malware, resulting in large-scale blackout in the capital Kiev of Ukraine in 2015 and 2016 [9]. The same malware also damaged a nuclear facility by falsifying control parameters in the supervisory control and data acquisition systems.

From the motion control standpoint, a lack of established cybersecurity measures may lead to critical incidents. Unsecured motion controllers may serve as an attractive target for adversaries.

Recent work has shown hopeful statistical based attack detection [10, 11], where the authors injected known statistical noise in the so called "watermarking" processes, which is than compared with the received control signal to detect malicious signal disturbances. Using this method it was shown possible to detect both replay and deception attacks [12, 13]. While these are impressive accomplishments, they do nothing to stop information leaks and thus surveillance attacks.

Encryption is an effective technique to secure data by encapsulating sensitive information at the communication level. When encryption techniques are applied to security enhancement of motion control devices, special treatment is needed according to specific system configurations and control schemes [14, 15, 16]. In particular, encrypted control is an emerging concept that encrypts not only signals on communication lines, but also control schemes and controller gains by applying homomorphic encryption algorithms [17, 18, 19, 20].

The goal of the work presented in this thesis is to establish theoretic controls methods to enhance cyber security for networked motion control systems utilizing somewhat homomorphic encryption [9, 17, 21, 22]. The proposed approach encrypts motion control algorithms, sensor signals, model parameters, feedback and feedforward gains, and performs necessary computation in the ciphertext space to generate motion commands to servo systems without a security hole. The concept of encrypted control is shown in Fig-

ure 1.1. This method ensures that sensitive system information is always encrypted except at the plant, where information decryption and control signal calculation are performed and executed.

Among existing homomorphic encryption algorithms, fully homomorphic encryption (FHE) can perform both addition and multiplication on the ciphertext an unlimited number of times. On the other hand, partially homomorphic encryption (PHE) performs either addition or multiplication an unlimited number of times. Where real-time control of robotic systems is concerned, practically usable FHE has not yet been realized due to high computational load. PHE however has been used in the majority of the existing encrypted control studies for practical reasons; most of these studies are limited to linear and relatively-low dimensional controllers [17, 22, 23, 5]. In some cases, nonlinear plant dynamics must be evaluated in real-time for model-based compensation, which increases the complexity of the control scheme. Due to this, expansion of encrypted control methodologies to general nonlinear and/or time-varying control has not been well studied. Furthermore, even "text-level" transformation between coordinates–a simple kinematics problem in robotics–involves matrix multiplications, which cannot be performed by PHE.

A "somewhat homomorphic encryption" (SHE) algorithm proposed by Dyer et al. [24] has shown promise of online encryption upon which this study will develop new realization procedures. The SHE family of cryptographic algorithms can perform both additive and multiplicative homomorphic encryption with a limited number of operations. Teranishi et al. showed it possible to use this SHE algorithm for real-time control [21]. Note that inappropriate selections of security parameters and signal quantization levels in SHE will cause overflow and system instability.

The authors' group has applied that particular SHE algorithm to dynamic controllers with nonlinear expressions [21, 25, 26]. Known limitations of SHE require the users to choose appropriate security parameters and quantization levels to balance between the performance and numerical computation stability (i.e., overflow). The original message can-

not be recovered in case of overflow, and is a critical issue in leveraging SHE algorithms in systems.

While the use of a long key is in general preferred to improve the cybersecurity of encrypted control systems, the increased computational overhead due to homomorphic encryption would degrade the real-time control performance. To resolve this trade-off, a concept of key switching has been studied in which relatively low-length keys are generated and switched at a certain frequency, fast enough before a single key is theoretically identified by the adversary via a brute-force attack [27]. Since the computational burden to continuously generate keys is high, the use of a dedicated high-performance circuit, such as Field Programmable Gate Arrays (FPGAs), is favorable rather than a fully software-based (e.g., CPU) approach.

1.2 Cryptographic Methods

1.2.1 Homomorphic encryption

A cryptosystem is represented by the tuple $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where $\mathsf{Gen} : \mathcal{S} \to \mathcal{K}$ is a key generation algorithm, $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ is an encryption algorithm, and $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ is a decryption algorithm. The set $\mathcal{S}$ contains security parameters such as key lengths. $\mathcal{K}$ is a key space, $\mathcal{C}$ is a ciphertext space, and $\mathcal{M}$ is a plaintext space. The cryptosystem $\mathcal{E}$ is said to be homomorphic if $\mathsf{Enc}(k, m) \circ \mathsf{Enc}(k, m') = \mathsf{Enc}(k, m \star m')$, $\forall m, m' \in \mathcal{M}$ is met, where $\circ$ and $\star$ are binary operations in the ciphertext and plaintext space, respectively. A key k is a pair of a public key pk and a secret key sk in asymmetric encryption. pk and sk are used for encryption and decryption.

As an example, the Van Dijk [28] family of ciphers is defined over the integers. So the plaintextspace $\mathcal{M}$ is the integers, the cipherspace $\mathcal{C}$ is a subset of the integers, the key space $\mathcal{K}$ is the set of primes or a list of primes. Within the cipherspace the notion of "addition" or "multiplication" may be defined, which we will notate as $\oplus$ and $\otimes$ respectively. The exact nature of all these spaces as well as which operations are available, will depend heavily on

the chosen cipher.

Homomorphic encryption allows certain types of arithmetic operation in ciphertext. Multiplicative homomorphic encryption, such as RSA [29] and ElGamal [30] algorithms, can perform multiplication in ciphertext: $\mathsf{Enc}(k, m) \otimes \mathsf{Enc}(k, m') = \mathsf{Enc}(k, m \times m')$. Similarly, additive homomorphic encryption, such as Paillier, can perform addition in ciphertext: $\mathsf{Enc}(k, m) \oplus \mathsf{Enc}(k, m') = \mathsf{Enc}(k, m + m')$. Note that operations $\otimes$ and $\oplus$ are not necessarily limited to traditional multiplication and addition between ciphertexts. For example, in the ElGamal algorithm, $\otimes$ is the Hadamard product. In the following, we omit the key k in the notation of encryption and decryption if appropriate for simplicity.

1.2.2 Somewhat Homomorphic Encryption

Somewhat homomorphic encryption (SHE) is a family of algorithms that can perform both additive and multiplicative homomorphic encryption with a limited number of operations– if operations are allowed for an arbitrary time, such an algorithm is called fully homomorphic encryption (FHE). The limiting factor is the divergence of noise introduced into the ciphertext, primarily by multiplication.

Depending on the SHE scheme used, the exact nature of noise introduction varies; however, an overarching pattern of these methods is that of *multiplicative depth*. Each time a ciphertext message participates in an encrypted multiplication, the depth of the resulting product increases by one. Once the depth has grown too large, the scheme is said to "overflow" and the homomorphic property is lost. In other words, the ciphertext can no longer be decrypted back to plaintext.

Thus, it is critical to minimize the net multiplicative depth, which could be done by appropriately factoring or expanding the arithmetic expression of interest. In addition, it is desirable to develop algorithms to *automate* the depth analysis and modification to be applicable to general complex expressions. The next section will show that sophisticated associative grouping can drastically reduce the net depth.

1.2.3 Homomorphic encryption of motion controllers

Encrypted control is an emerging field of control theory [31]. Currently, several international research groups are jointly or independently working on related topics [32, 33, 34, 35, 36, 37, 38, 39, 40]. As one of the earliest attempts, Kogiso and Fujita proposed an approach to secured realization of a linear motion controller in the cloud [17]. As opposed to the conventional approach of encrypting only signals on the communication line, this concept is to encrypt both controller gains and signals by homomorphic public-key encryption as shown in Figure 1.1. This method ensures that sensitive system information is always encrypted, except at the plant where information decryption and control signal execution is performed. One important feature of this scheme is that the secret key for decrypting signals does not need to be shared with the cloud controller, a frequent target of attack. Only the end device (the plant in Figure 1.1) possesses the secret key, which is considered a safer configuration. Encrypted signals and feedback gains in the control scheme are then used to directly compute motion commands in ciphertext being sent to the actuator. Because the signals and gains inside the motion controller are in ciphertext, not plaintext, this encryption approach is suitable as proactive measures for unauthorized login and falsification.

1.3 Contributions

Homomorphic Encryption is a relatively new cryptographic method which, unlike traditional encryption, allows computations to be preformed on encrypted data. Robotic controllers can take advantage of these new techniques to increase system security by encrypting the entire motion control scheme including: sensor signals, model parameters, feedback gains, and perform computation in the ciphertext space to generate motion commands without a security hole. However, numerous challenges exist which have limited the wide spread adoption of homomorphically encrypted control systems. The following

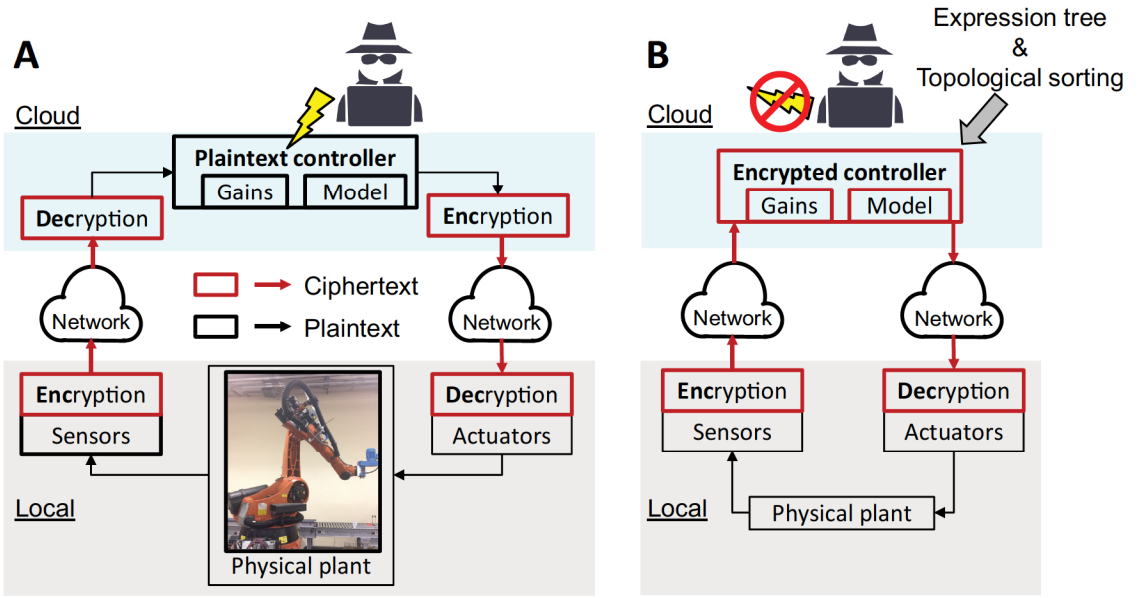


Figure 1.1: Security-enhanced networked control. A. Conventional encrypted communication (control scheme computation in plaintext), B. Encrypted control (control scheme computation in ciphertext).

thesis address several of these pressing issues–cryptographic overflow and heterogenous deployment. Cryptographic overflow is a phenomenon intrinsic to homomorphic ciphers. As encrypted data is computed on the level of 'noise' inside the ciphertext increases, until it becomes too great making decryption impossible, this is known as 'overflow'. The primary contributor to noise growth is multiplication. Thus, this thesis explores topological sorting methods to find semantically equivalent but syntactically simpler control expressions. This allows an encrypted control scheme to preform the same calculation but with fewer multiplications, thus reducing the total amount of noise injected into the system. Furthermore, encrypted calculations impose a hefty computational burden as compared to its unencrypted counterparts. As such, heterogeneous mix of different computing technologies (i.e. CPU, GPU, FPGA) are needed to achieve real-time signal processing. As such, this thesis explores which aspects of an encrypted control system is best suited for which computing technology and describes a deployment strategy to take advantage of these differences.

CHAPTER 2

CRYPTOSYTEM COMPARISON

The goal of this research is to establish control theoretic methods to enhance cyber security of networked motion control systems by utilizing somewhat homomorphic encryption. The proposed approach will encrypt the entire motion control schemes including: sensor signals, model parameters, feedback gains, and performs computation in the ciphertext space to generate motion commands to servo systems without a security hole. The thesis will discuss implementation of encrypted bilateral teleoperation control schemes with nonlinear friction compensation. The thesis will present (1) encrypted teleoperation control realization with somewhat homomorphic encryption and (2) simulation results.

2.1 Encrypted Control Concept

2.1.1 PHE for linear systems and limitations

One of the biggest challenges of homomorphic computation, is the significantly limited arithmetic operation capability in ciphertext. Early attempts such as [41] tried to implement a fully-homomorphic encryption (FHE) algorithm to perform all arithmetic operations in the ciphertext space. Note that control commands need to be updated, typically, on the order of 10 to 100 milliseconds for closed-loop dynamic control of industrial motion systems. However, computation time and finite lifespan (bootstrapping) of encrypted variables were reported to be impractical with FHE [41]. The current state-of-the-art regarding real-time encrypted motion control, adopts multiplicative partial homomorphic encryption (PHE) schemes such as RSA [29, 42] and ElGamal [30]. This method has been applied to realizing a class of linear controllers, including: PID controllers, two-degree-of-freedom controllers, disturbance observers, and model-predictive controllers (with single iteration per sampling

period) [9, 17, 22, 23]. Figure 2.1 shows an example of such implementation of state-feedback with a linear observer applied to a second order inertial system (e.g., a DC motor) [22]. Recall that only either addition or multiplication can be performed in ciphertext with PHE. As shown in Figure 2.1A, the control scheme is represented by a state-space equation,

$$\begin{bmatrix} x[k+1] \\ u[k] \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x[k] \\ y[k] \end{bmatrix} := \boldsymbol{\Phi}\boldsymbol{\xi}[k] \tag{2.1}$$

$$= f\left(\boldsymbol{\Phi}, \boldsymbol{\xi}[k]\right) = f^{\times} \circ f^{+} \tag{2.2}$$

where $\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_1 & \boldsymbol{\Phi}_2 & \dots \end{bmatrix}$ is a state matrix represented by column vectors and $\boldsymbol{\xi}[k] = \begin{bmatrix} \xi_1 & \xi_2 & \dots \end{bmatrix}^T$ is a state variable vector. To apply the multiplicative PHE algorithm ElGamal, multiplications and additions are separated into an expanded form of matrix-vector products: $\boldsymbol{\Phi}\boldsymbol{\xi}[k] = \xi_1\boldsymbol{\Phi}_1 + \xi_2\boldsymbol{\Phi}_2 + \dots = \begin{bmatrix} \boldsymbol{\Psi}_1 & \boldsymbol{\Psi}_2 & \dots \end{bmatrix} = \sum \boldsymbol{\Psi}_i$. Where multiplicative operations $f^{\times}$ occurred in ciphertext, and additive operations f^{+} occurred in plaintext, where: $f^{\times}\left(\mathsf{Enc}(\boldsymbol{\Phi}), \mathsf{Enc}(\boldsymbol{\xi}[k])\right) = \begin{bmatrix} \mathsf{Enc}(\xi_1) \otimes \mathsf{Enc}(\boldsymbol{\Phi}_1) & \mathsf{Enc}(\xi_2) \otimes \mathsf{Enc}(\boldsymbol{\Phi}_2) & \dots \end{bmatrix} = \mathsf{Enc}(\boldsymbol{\Psi})$ and $f^{+}(\boldsymbol{\Psi}) = \sum \mathsf{Dec}(\mathsf{Enc}(\boldsymbol{\Psi}_i))$ as show in Figure 2.1 B. Since addition is preformed in plaintext after decoding, this realization leaves a potential security hole in the system as shown in Figure 2.1C.

An extension from single-controller-single-plant linear systems to nonlinear systems or multi-plant systems is not trivial. Successful realization depends highly on the choice of an encryption algorithm and the structure of the control scheme.

2.1.2 Proposed Approach

PHE algorithms such as RSA (multiplicative), ElGamal (multiplicative), Paillier (additive) have been used for encryption of linear time-invariant (LTI) controllers [8] including the authors' previous work considering security holes resulting from arithmetic operations on plaintext, as mentioned above. Research to expand homomorphic encryption methodolo-

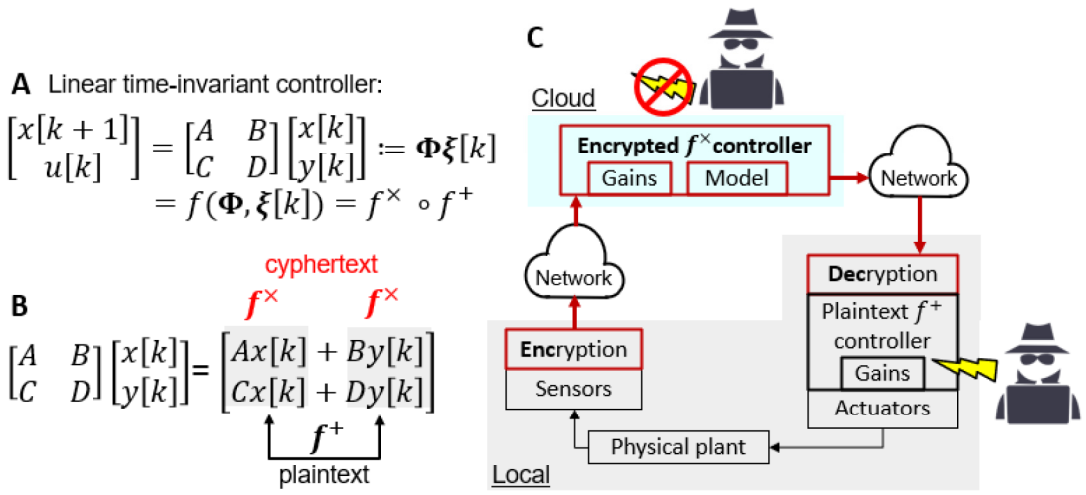


Figure 2.1: Encryption of multivariable linear controller. A.) Controller B.) Realization with PHE C.) Implementation with a potential security hole at the plant.

gies to generalized, or nonlinear time-varying, control has not been performed almost at all [5]. The main technical barrier has been a lack of an encryption algorithm capable of handling increased arithmetic operations required for realization of nonlinear controllers. In some cases, nonlinear plant dynamics must be evaluated in real time for model-based compensation, which increases the complexity of the control scheme, however Teranishi and Kogiso showed it feasible to use SHE for real time control [21]. This paper will utilize emerging somewhat homomorphic encryption (SHE) [23] to realize encrypted nonlinear controllers. SHE allows for a limited number of both multiplication and addition in ciphertext before operations overflow or lose precision. Note that SHE in general is also known to be computationally expensive and its application to real-time control is considered to be infeasible. However, a recent SHE algorithm proposed by Dyer et al. [24] has shown promise of online encryption upon which this study will develop new realization procedures.

System parameters to be protected should not be stored or operated in plaintext to avoid potential data breach. Recall that PHE-based approach [32, 30, 22] was to manipulate a linear control scheme and sort additions and multiplications separated into a product of a constant matrix and a state variable vector (i.e., LTI state-space representation). For SHE,

care must be taken regarding algebraic manipulation of high-order polynomial expressions. Not only the amount of arithmetic operations, but also the order of the operations significantly impacts the risk of overflow and loss of precision.

2.1.3 Problem formulation

We propose to manipulate the algebraic expressions including the nonlinear terms and obtain an executable form in ciphertext as shown in (Equation 2.3). The concept is to evaluate some of the products between state variables (i.e., sensor readings), given as $\boldsymbol{\varsigma}[k]$ in the sensing device in advance and perform encryption together with other linear variables. Nonlinear functions, such as sin and cos, cannot be evaluated in ciphertext, which are also evaluated and encrypted in the sensing device. Based on this concept, the realization problem of encrypted nonlinear control schemes is formulated as follows:

Problem: Determine constant matrices $\boldsymbol{\Phi}$, $\boldsymbol{\Psi}$ and nonlinear state vector $\boldsymbol{\varsigma}[k]$ for the nonlinear control scheme represented by:

$$\boldsymbol{u}[k] = \boldsymbol{\Phi}\boldsymbol{\xi}[k] + \boldsymbol{\Psi}\boldsymbol{\varsigma}[k] := f^{\text{SHE}}\left(\boldsymbol{\Phi}, \boldsymbol{\xi}[k], \boldsymbol{\Psi}, \boldsymbol{\varsigma}[k]\right) \tag{2.3}$$

such that (Equation A.3) and (Equation A.4) are simultaneously satisfied for given κ and p.

Equation A.3 and Equation A.4 are restated below:

$$\kappa > (n+1)^d M^d \tag{2.4}$$

$$p > (n+1)^d (M + \kappa^2)^d \tag{2.5}$$

where κ and p are cryptographic constants of the Dyer's cypher dictated by the choice of security parameters, M is the maximum value any intermediary encrypted computation

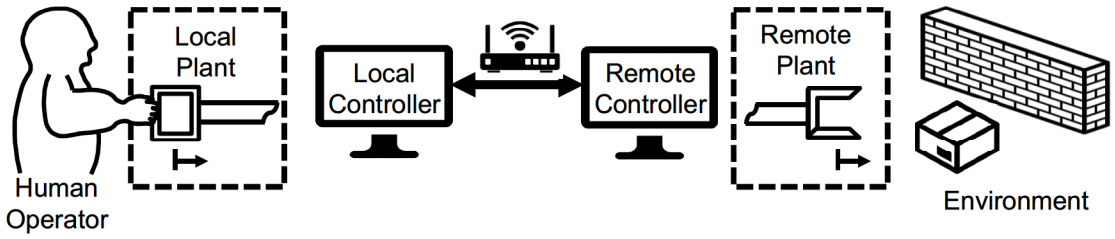


Figure 2.2: Bilateral teleoperation

can be, and n is the dimensionality of the encrypted calculation (i.e. how many variables the encrypted calculation takes as input). These two equations serve as constraints relating choice of security parameters with size of cypherspace. To summarize, if the security parameters are too weak, then the cypherspace will be too small to accommodate large encrypted values.

This thesis will, amoungst other things, address issues mentioned above and demonstrate the applicability of SHE to bilateral control of two telemanipulators.

2.2 Encrypted teleoperation

2.2.1 Representative Teleoperation Control Scheme

This section will extend the SHE approach to an encrypted teleoperation system where two control loops of the local and remote plants are intertwined.

Let the coefficients m, b, μ, τ, and f denote system mass, damping, friction coefficient, actuator force, and external force; furthermore, let the subscript m and s denote the local and remote system. Then the system is modeled by:

$$m_m\ddot{x}_m + b_m\dot{x}_m + \mu_m \operatorname{sign}(\dot{x}_m) = \tau_m + f_m \tag{2.6}$$

$$m_s\ddot{x}_s + b_s\dot{x}_s + \mu_s \operatorname{sign}(\dot{x}_s) = \tau_s - f_s \tag{2.7}$$

Evaluation of $\mathbf{\Phi\xi}[k]$ (linear) and $\mathbf{\Psi\varsigma}[k]$ (nonlinear) must be performed in an increased

number of encoding blocks. Figure 2.2 shows a possible configuration of an encrypted teleoperation system. The main concept is to encrypt shared information using a private encryption key known **only** to the plants. Both the local and remote plants are responsible for system output measurement and encryption by using the public keys. The networked controller stores encrypted system parameters, as well as encrypted output measurements received from both plants.

The general linear terms may be represented by:

$$\begin{aligned}\begin{bmatrix}\boldsymbol{\Phi}_m\boldsymbol{\xi}_m\\ \boldsymbol{\Phi}_s\boldsymbol{\xi}_s\end{bmatrix} &= \begin{bmatrix}\boldsymbol{K}_{amm} & \boldsymbol{K}_{dmm} & \boldsymbol{K}_{pmm}\\ \boldsymbol{K}_{asm} & \boldsymbol{K}_{dsm} & \boldsymbol{K}_{psm}\end{bmatrix}\begin{bmatrix}\ddot{\boldsymbol{x}}_m\\ \dot{\boldsymbol{x}}_m\\ \boldsymbol{x}_m\end{bmatrix}\\ &+ \begin{bmatrix}\boldsymbol{K}_{ams} & \boldsymbol{K}_{dms} & \boldsymbol{K}_{pms}\\ \boldsymbol{K}_{ass} & \boldsymbol{K}_{dss} & \boldsymbol{K}_{pss}\end{bmatrix}\begin{bmatrix}\ddot{\boldsymbol{x}}_s\\ \dot{\boldsymbol{x}}_s\\ \boldsymbol{x}_s\end{bmatrix}\\ &+ \begin{bmatrix}\boldsymbol{K}_{fmm} & \boldsymbol{K}_{fms}\\ \boldsymbol{K}_{fsm} & \boldsymbol{K}_{fss}\end{bmatrix}\begin{bmatrix}\boldsymbol{f}_m\\ \boldsymbol{f}_s\end{bmatrix}\end{aligned} \tag{2.8}$$

using accelerations, velocities, displacements, forces, as well as gains, to introduce intervening impedance (i.e., virtual spring and damper) between two motion plants [43]. Nonlinear terms $(\boldsymbol{\Psi}_m\boldsymbol{\varsigma}_m)$ and $(\boldsymbol{\Psi}_s\boldsymbol{\varsigma}_s)$ that compensate for friction, time-delay, and other nonlinearities in the system, will be decomposed into $\boldsymbol{\Psi}$ and $\boldsymbol{\varsigma}_{m,s}$.

While there are a variety of control schemes to realize bilateral teleportation, a representative symmetric control scheme utilizing PD feedback with inertial and friction compensation is considered in this paper:

$$\tau_m = (m_m - m_{ms})\ddot{x}_m + k_p(x_s - x_m) + k_d(\dot{x}_s - \dot{x}_m) + 0.9\mu_m \operatorname{sign}(\dot{x}_m) \tag{2.9}$$

$$\tau_s = (m_s - m_{ms})\ddot{x}_s + k_p(x_m - x_s) + k_d(\dot{x}_m - \dot{x}_s) + 0.9\mu_s \operatorname{sign}(\dot{x}_s) \tag{2.10}$$

where $\mathbf{\Psi} = \text{diag}[0.9\mu_m, 0.9\mu_s], \boldsymbol{\varsigma} = [\text{sign}(\dot{x}_m), \text{sign}(\dot{x}_s)]^T$. Other linear terms are expressed in $\mathbf{\Phi}\boldsymbol{\xi}$.

2.3 Realization of Encrypted Teleoperation

2.3.1 Choice of Security Parameters

BFV parameters:

Primarily, we focused on the computation time for `poly_modulus` about the BFV cryptosystem. `poly_modulus` affects the range of signals that are encryptable. Increasing the value of `poly_modulus` makes encryption of a wider range of signals possible.

Dyer's parameters:

Dyer's SHE method requires very large integers to represent ciphertext. The bit-width of these ciphertexts are determined by the security parameters λ, η (Equation A.2). These parameters determine the size of primes p, q which define the public modulus N. Since all homomorphic operations are performed modulo the public modulus, a bit-width of

$$\gamma = \lfloor \log_2(N) \rfloor + 1 \tag{2.11}$$

is required to represent all of cipherspace.

We refer to γ as the *bit requirement* of the cipherspace. If an integer's bit requirement exceeds the system's word size w, then the integer will have to be processed piece-wise in

segments of length w. This results in γ/w additional operations being required to operate on big integers.

The `x64` architecture is a popular choice today, and has a $w = 64$. Virtually all choices of security parameters result in $\gamma > 64$. Given that the C++ standard library cannot represent integers larger than the system's w, a large integer library is required. Performance of large integer arithmetic is implementation dependent.

2.3.2 Implementation Used

BFV:

BFV Encryption was realized via Microsoft's *Simple Encrypted Arithmetic Library* (SEAL). SEAL is a highly optimized library which can provide hardware specific speedups when using supported processor architecture [44].

SEAL allows users to set plaintext, ciphertext, and rng parameters, however typically only the `poly_modulus`, `coeff_modulus` are changed, as they directly dictate security level. We will restrict our attention to only these aspects of the BFV cryptosystem.

Dyer's:

Dyer's encryption was implemented in-house against C++17 feature set, using MSVC version `19.29.30140` targeting `x64` architecture. [45] was used for large integer arithmetic. The minimal residue

$$a \text{ Mod } m = \begin{cases} b, & b < |b - m|, \\ b - m, & \text{otherwise}, \end{cases} \tag{2.12}$$

was employed for correct decryption of negative integers as described by [46].

2.3.3 Encrypted Arithmetic Comparison

An encrypted controller is constructed by relegating the evaluation of all special functions (e.g. $\sin$, $\exp$, etc.) to the plant such that only additive and multiplicative operations

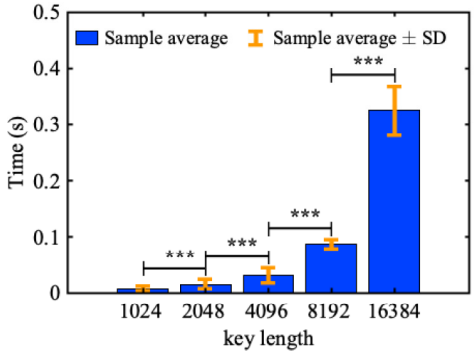
remain. These operations are performed homomorphically in cipherspace, therefore the speed at which a cryptographic can evaluate these operations has a direct impact on the feasibility of real-time control.

This section analyzes execution time of homommorphic arithmetic parameterized by the systems' security parameters. To do this we measured the time to compute the polynomial $ax + by$, where $a, b, x, y \leftarrow \{0, \cdots, 9\}$, 1000 times on these cryptosystems for each security parameter, described in the following sections. The range of variables is chosen not to violate homomorphic operation on the ciphertext. The specification of the CPU is Intel Core i5-8250U at 1.6 GHz.

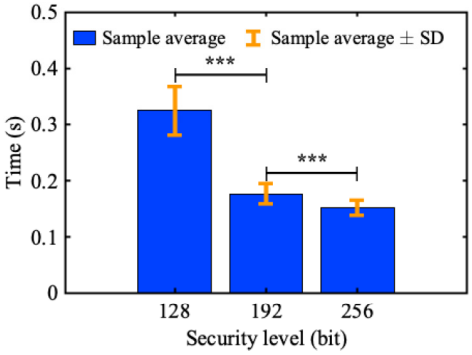
BFV cryptosystem:

`poly_modulus` and `coeff_modulus` are key parameters that determine computational load. Figure 2.3 shows the execution time to compute a polynomial $ax + by$. Figure 2.3 (a) shows the mean $\pm$ standard deviation (SD) calculated from 1000 samples for each bar. Figure 2.3 (b) shows a log-log breakdown of average computation time for homomorphic operations of the BFV scheme. In Figure 2.3 (a), the mean for each polynomial modulus is significantly larger than that of the bar on the left at a 0.1 % significance level. It was confirmed using a pairwise t-test. The figure shows a polynomial growth of the computation time with respect to `poly_modulus`. This trend held for the other operations in the cryptosystem.

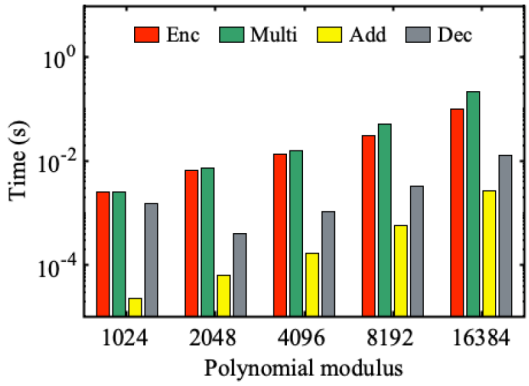
`coeff_modulus` determines the security Level in Microsoft SEAL. 128, 192, and 256 bit security levels are defined in the library (the default is 128 bit). Parameters of `coeff_modulus` associated with the security level are given as default values. Higher security requires a smaller `coeff_modulus`. Figure 2.3 (c) shows that the execution time for each security level. The average is significantly less than that of the bar on the left at a 0.1 % significance level.



(a) Comparison of average computation time in BFV scheme for `plain_modulus` = 1024. The mean and standard deviation derived from 1000 samples. No horizontal bar between neighboring bars indicate that the left-side bar is not statistically larger than the right at a 5% significance level.

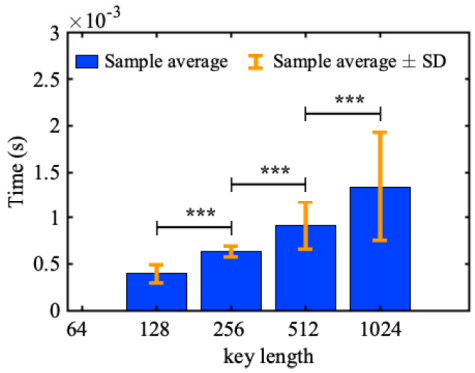


(b) Comparison of security level between `poly_modulus` = 16384 and `plain_modulus`. No horizontal bar between neighboring bars indicates that the left-side bar is not statistically small than the right at a 5% significance level.

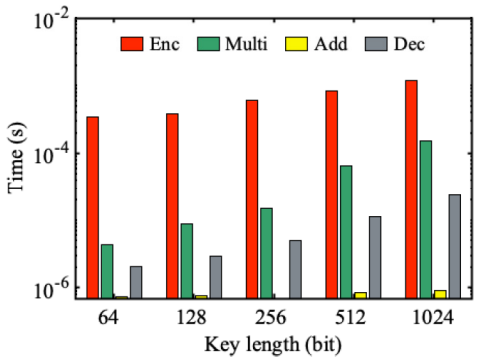


(c) Breakdown of average computation times. Shown are comparison of encryption; $\mathsf{Enc}(a)$, $\cdots$, $\mathsf{Enc}(y)$, Multiplication; $\mathsf{Enc}(a) \otimes \mathsf{Enc}(x)$ and $\mathsf{Enc}(b) \otimes \mathsf{Enc}(y)$, Addition; $\mathsf{Enc}(ax) \oplus \mathsf{Enc}(by)$, Decryption; $\mathsf{Dec}(\mathsf{Enc}(ax + by))$.

Figure 2.3: Computation time analysis of BFV. P-values are indicated as ***, $p \leq 0.001$; **, $p \leq 0.01$; *, $p \leq 0.05$.



(a) Comparison of average computation time of Dyer's method for different polynomial moduli. No horizontal bar between neighboring bars indicate that the left-side bar is not statistically larger than the right at the 5% significance level.



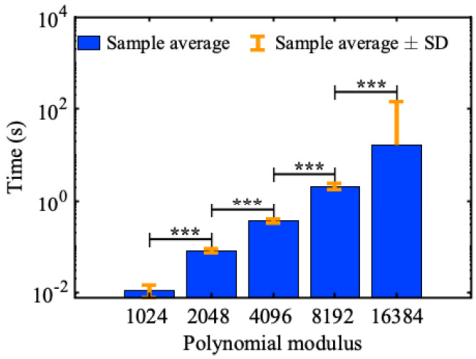
(b) Breakdown of average computation times of SHE operations.

Figure 2.4: Computation time analysis of Dyer's SHE. P-values are indicated as stars described in Figure 2.3.

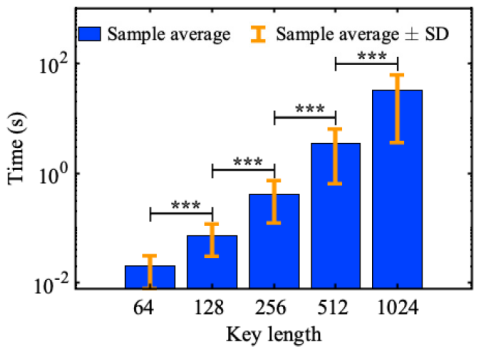
Dyer's cryptosystem:

Dyer's cryptosystem has several security parameters: key length λ bit, ρ, and ρ'. In this cryptosystem, the key length has a significant impact on the computational load. Figure 2.4 shows the execution time to compute a simple polynomial: $ax+by$. Figure 2.4 (a) shows the mean $\pm$ standard deviation (SD) calculated from 1000 samples for each bar. Figure 2.4 (b) shows the breakdown of average computation times of homomorphic operations in a log-log plot. All of the mean values shown in Figure 2.4 (a) are larger than that of the left at a 0.1% significance level. While in general the computation time increases as the key length increases, especially ,that of homomorphic multiplication on cipher texts grow up faster than the other element. On the other hand, the computation time for additions is negligibly small in this cryptosystem.

While BFV and Dyer's SHE schemes exhibit different characteristics in terms of key generation, encryption, addition, multiplication, and decryption, overall, Dyer's scheme completes a simple polynomial approximately two orders of magnitude faster than BFV. Dyer's cryptosystem may be more suitable for real-time motion control applications than



(a) Comparison of `poly_modulus` with `plain_modulus` = 1024.



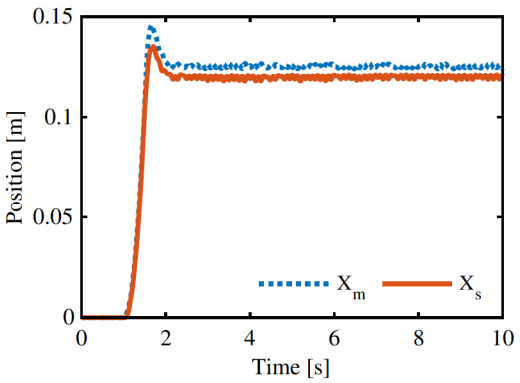
(b) Comparison of key length λ with $\rho = 1$, $\rho' = \lambda/4$.

Figure 2.5: Computation time of key generation. P-values are indicated as stars described in Figure 2.3. No horizontal bar between neighboring bars indicate that the left-side bar is not statistically larger than the right at the 5% significance level.

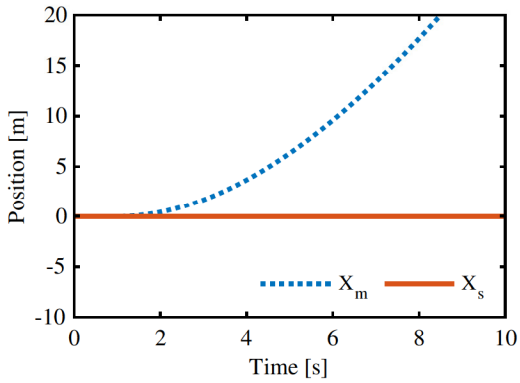
BFV. On the other hand, it should be mentioned that the computation load is high for generating a key in Dyer's encryption.

On the other hand, it should be noted that Dyer's scheme has a notable computational load for key generation. The computational load for key generation becomes a problem when we consider improving the security level of controllers such as dynamic key schemes. Dynamic key generation is one of the approaches to improve security by switching between multiple keys to increase the cost of attacks. Since this method requires constant key generation, the computational cost of key generation should be reduced. Figure 2.5 shows the time required to generate keys for the Dyer and BFV.

The computation times for a simple polynomial are close for the BFV with `poly_modulus` = 1024 (6.668 ms) and the Dyer with $\lambda = 1024$ (1.3 ms). In Figure 2.5, the key generation times in each scheme are 10.9 ms for the BFV and 32.5 s for the Dyer. The BFV computation time to generate a key is less than the Dyer. This suggests that the BFV cipher is more suitable for building a more secure control system using dynamic keys.



(a) Dyer's encrypted control signal ($\lambda = 256, \rho = 1, \rho' = 32, \Delta = 0.01$). Small oscillations are due to encoder.



(b) Dyer's encrypted control signal ($\lambda = 512, \rho = 1, \rho' = 32, \Delta = 0.001$). The security parameters are not appropriate to permit correct computations.

Figure 2.6: Simulink control signals

2.4 Simulation

2.4.1 Simulink/C++ Interoperations

Simulink (Mathworks, Natick, MA) is a graphical programming environment designed to model dynamic systems by wiring together computational blocks. The system dynamics, and control loop were implemented in this fashion. The controller was implemented in C++17 via matlab's mex-api. The mex toolchain works by invoking the system's compiler on C++ source written against the mex interface; then linking with MATLAB provided static libraries, which provide the interface's definitions. The result is either a `.mexa64`, `.mexmaci64`, or `.mexw64` file for linux, mac, or windows systems respectively.

These mex files are essentially metadata bundled with a shared object which the MATLAB interpreter loads at runtime. This architecture provides several benefits. First, it allows different implementations of the controller to be used in a "plug-and-play" fashion. Second, lower-level languages such as C++ gives more precise control over the resources used and representation of encrypted data. Three different implementations of the teleoperated controller describes by (Equation 2.9) and (Equation 2.10) were tested: *plaintext-control*, *bfv-control*, and *dyer-control*.

2.4.2 Simulation Results

Plaintext-control:

The plaintext controller does not incur any of the computational overhead that the encrypted methods do. Therefore, it serves as a good baseline with which to compare the other methods. Simulations were run on an AMD Ryzen 9 4900HS 3.00 GHz processor running Windows 11. Using this system, the plaintext implementation was able to achieve a 16.3 kHz refresh rate. This will serve as a baseline to compare the encrypted implementation against.

BFV-control:

While BFV does provide homomorphic operations, its execution time is far too slow for real time operation. We varied the `poly_modulus`, and found that the BFV encrypted controller refresh rate remained relatively constant see Table 2.2. This is far bellow what is required for real time operations, generally considered to be >1kHz.

Dyer-control:

Dyer's encryption was faster than BFV for all security parameters tested. Results show that as the security parameters of the encryption increases the performance decreases, See Table 2.1.

Encrypted controllers may be used in real-time systems if an appropriate encryption scheme is used. Furthermore, improper choice of security parameters can result in unstable behavior, as shown in Figure 2.6b.

λ	f (Hz)
500	485
400	666
300	956
200	1206
100	1539

Table 2.1: Dyer's encryption controller performance.

$Poly\ Modulus\ Degree$	f (Hz)
4000	2.80
6000	2.80
8000	2.53
10000	3.02
12000	2.71

Table 2.2: BFV encryption controller performance.

CHAPTER 3

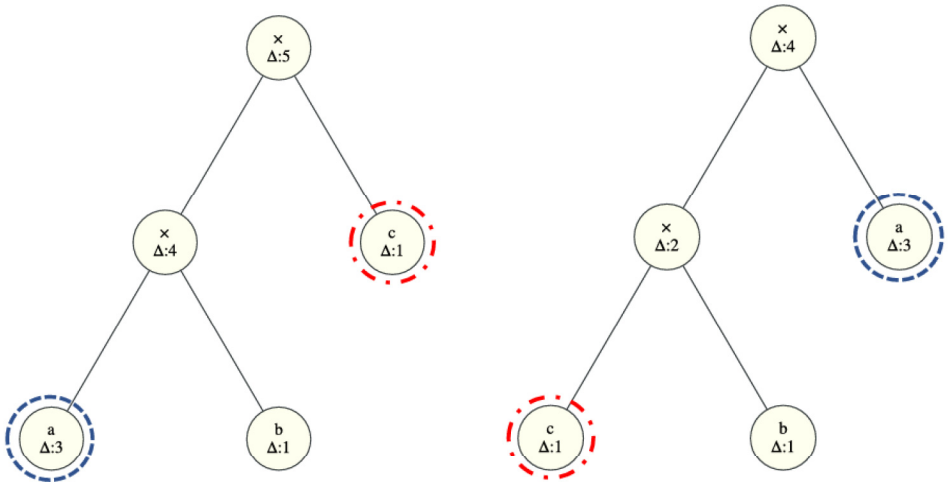
TOPOLOGICAL EXPRESSION SORTING

This thesis presents topological sorting methods to minimize the multiplicative depth of encrypted arithmetic expressions. The research aims to increase compatibility between nonlinear dynamic control schemes and homomorphic encryption methods, which are known to be limited by the quantity of multiplicative operations. The proposed method adapts rewrite rules originally developed for encrypted binary circuits to depth manipulation of arithmetic circuits. The thesis further introduces methods to normalize circuit paths that have incompatible depth. Finally, the thesis provides benchmarks demonstrating the improved depth in encrypted computed torque control of a dynamic manipulator and discusses how achieved improvements translate to increased cybersecurity.

3.1 Expression Manipulations

3.1.1 Tree Representation

To analyze an arithmetic expression in a convenient manner, the binary-tree data structure is adopted where each operation is represented by an internal node with its children as the operands. SHE methods are often limited to additive and multiplicative operations only [46]. As such, the paper imposes the same limitation on an operation-limited syntax tree called an *arithmetic circuit* [47, 48]. It is observed that the leafs are always variables of the expression, while interior nodes are allowed binary operations. By disallowing unary operations (e.g., $\sin$, $\exp$, etc.), any arithmetic circuit can be represented as a *full* binary tree at all times.



(a) Original (b) Rewritten

Figure 3.1: The left expression has a node of greater depth, a, deeper in the tree. By swapping with a shallower node, c, we lift node a up thus reducing total depth by one.

3.1.2 Rewrite Rules

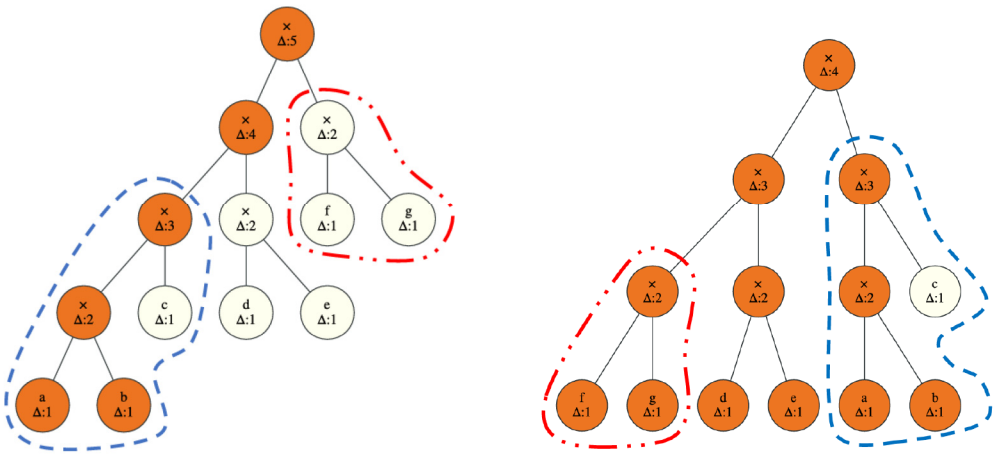
The arithmetic circuit representation is convenient when attempting to analyze and optimize the depth of the circuit. Two perturbations, adapted from [48], are introduced to be compatible with arithmetic circuits. These perturbations are presented as "rewrite rules," whereby certain detectable patterns are replaced with syntactically different, but semantically equivalent representations.

Associative Rewrite

represents a regrouping of parameters as in $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, and is only applicable to two adjacent `MUL` gates. For example, if a has a greater depth than both b and c, i.e.,

$$\mathsf{depth}(b),\ \mathsf{depth}(c) < \mathsf{depth}(a). \tag{3.1}$$

then, a rewrite will decrease the overall depth of the circuit, If the above expression holds, then the algorithm can *associatively rewrite* the expression by swapping node a with node c as shown in Figure 3.1 and described in Algorithm 1.



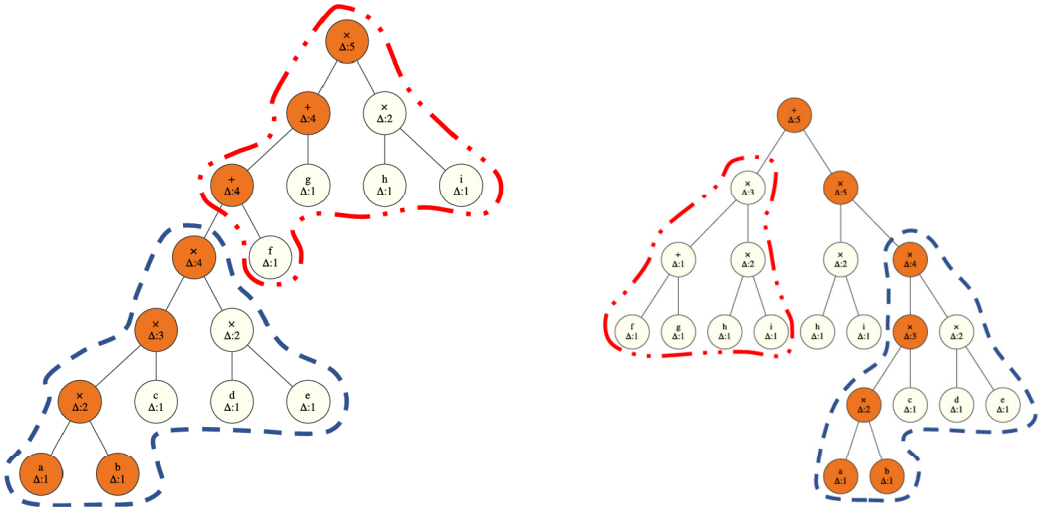
(a) Original Expression (b) Rewritten Expression

Figure 3.2: The original expression on the left. On the right is the output of the associative rewrite. Note the circled regions have been swapped with each other in the rewritten circuit.

Since depth(a) is the greatest of all nodes considered, it is counterproductive to have node a deep in the graph. By swapping node a and c, we reduce the number of future `MUL` gates that a will have to pass through by one. Furthermore, because of (Equation 3.1), a has the greatest depth, and it follows that a's depth would dominate this subcircuit. Thus, by reducing future gates a must pass through, the total depth contributed by this subcircuit can be reduced. As a result, this rewrite rule, when applied to sub-expressions satisfying (Equation 3.1), reduces the depth by one. Another example is shown in Figure 3.2 where the associative rewrite has the tendency of convert graph depth to graph width, thus steering the graph towards a more *balanced* configuration. The path shown in orange is the "critical path" that results in the maximum depth of the circuit.

Distributive Rewrite

is not able to lower circuit depth, unlike the associative rewrite, and in fact adds an additional `MUL` gate into the circuit. In spite of this, the distributive rewrite is advantageous by moving a `MUL` gates higher up in the circuit to be adjacent to a different `MUL` gate. When these newly neighbored gates satisfy the pattern described by (Equation 3.1), the distributive rewrite will facilitate a future associative rewrite with depth reducing ability at the cost



(a) Original Expression (b) Rewritten Expression

Figure 3.3: The circled regions show sub-graphs that were rearranged, while the uncircled represent newly created gates.

of net one additional `MUL` gate as shown in Figure 3.3.

A larger number of `MUL` gates negatively impacts the execution time of the resulting circuit; however, if the distributive rewrite is able to facilitate a future associative rewrite– and thus facilitate future depth reduction–then it is worth the few additional `MUL` gates for the depth reduction gained. The process is depicted in Figure 3.4.

The distributive rewrite represents the distribution of an outside factor to a term in a sum, such as $(((x + y_1) + ...) + y_k) \cdot z$. The naïve approch would be to distribute the z to every addend, such as: $(xz) + (y_1 z) + ... + (y_k z)$. This however, will add a `MUL` gate for every `ADD` gate on the path, which is often more than is necessary. Instead, one can selectively distribute z in a form such as: $(x \cdot z) + (y_1 + ... + y_k) \cdot z$.

3.2 Depth analyzer software

The proposed depth analysis and rewrite rules have been implemented in a CPython based application. The application revolves around a binary tree class called `ExpressionTree`, which holds references to the encompassing `ExpressionNode` objects. Sympy, a popular computer algebra system in the python ecosystem, was used to parse expressions-string

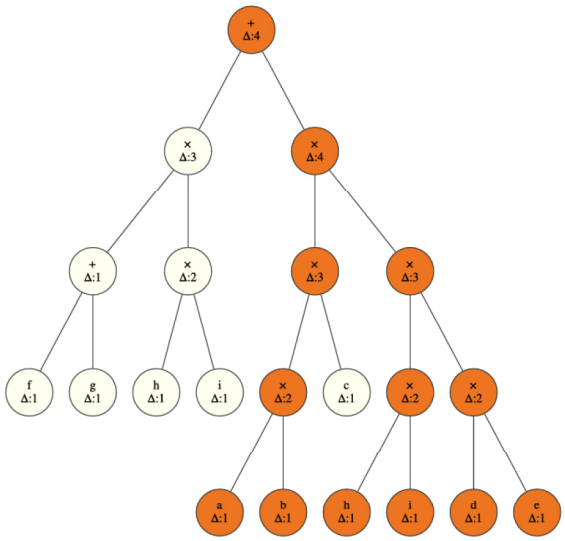


Figure 3.4: Outcome of first apply distributive-rewrite, followed by associative-rewrite. Net depth decreases.

input into `MUL` and `ADD` nodes respectively. After parsing, the critical path is calculated by recursively crawling the `ExpressionTree` from the top-down. Once the critical-path(s) is known, the algorithm iteratively checks for possible rewrite.

3.2.1 Associative Rewrite Procedure

The associative rewrite looks for any adjacent `MUL` gates on the critical path. If found the adjacent gates are checked against (Equation 3.1), to determine if a rewrite will produce a more optimal depth. This check is carried out by Algorithm 3, which checks if the depth of either of c's children is greater than p. If the condition (Equation 3.1) is met, the associative rewrite is processed.

Algorithm 1 Associative Rewrite Attempt

Input Let π be the critical path

procedure ASSOCIATIVEREWRITE(π)
  $c \leftarrow$ `pop` (π) ▷ *`pop` removes and returns next node*
  while $\pi \neq \emptyset$ **do**
    ▷ *"Cycling through pairs* $(a, b) \rightarrow (b, c) \rightarrow (c, d)$*"*
    $p \leftarrow c$
    $c \leftarrow$ `pop` (π)
    if ISMUL(p) **and** ISMUL(c) **then**
      $s \leftarrow$ GETSIBLING(p, c)
      switch CHECKCHILDREN(s, c) **do**
        case `LEFT`
          SWAP(s, c_ℓ)
          return `True`
        case `RIGHT`
          SWAP(s, c_r)
          return `True`
        case `NONE`
          continue
    end if
  end while
  return `False` ▷ *No swaps were possible*
end procedure

3.2.2 Distributive Rewrite Procedure

The distributive rewrite viability check is similar to that of the associative check. The main difference is that where the associative rewrite acts on two adjacent `MUL` gates, the distributive rewrite acts on `MUL` gates separated by an arbitrary amount of `ADD` gates, which we call *quasi-adjacent*.

The distributive rewrite iterates over every pair of quasi-adjacent `MUL` gates in the critical path, if any exist. If two quasi-adjacent `MUL` gates are found, Algorithm 3 checks if the associative rewrite on said quasi-adjacent gates would be beneficial. While the associative rewrite is by itself not possible for quasi-adjacent gates, the distributive rewrite will rearrange the tree such that the quasi-adjacent `MUL` gates are now next to each other. Algorithm 2 implements this procedure.

Algorithm 2 Distributive Rewrite Attempt

```
Input Let π be the critical path
procedure DistributiveRewrite(π)
    c ← nextMul(π)                          ▷remove and return next Mul gate
    while π ≠ ∅ do
        ▷ Cycling through pairs (a, b) → (b, c) → (c, d)
        p ← c
        c ← nextMul(π)
        u ← getUncle(p, c)
        if CheckChildren(u, c) then
            s ← GetSibling(p, c)
            Orphan(c)
            ReplaceParent(s)
            overwrite(p, c · u + p)
            break
        end if
    end while
end procedure
```

3.2.3 Helpers

The rewrite procedures depend on a handful of trivial as well as non-trivial subroutines. Some non-trivial routines are defined as shown bellow.

Check Children

The `CheckChildren` procedure is used to see if an associative rewrite will yield a net reduction in depth. It does this by comparing the depths of children c against the depth of node s, where node s is either the *sibling* or *uncle* of c.

Algorithm 3 Check if swapping children will reduce depth

```
procedure CHECKCHILDREN(s, c)                ▷We compare children of c against s
    if getDepth(c_ℓ) > getDepth(s) then
        return LEFT
    else if getDepth(c_r) > getDepth(s) then
        return RIGHT
    else
        return NONE
    end if
end procedure
```

Get Sibling

In order to assess the viability of an associative rewrite, we need to run `CheckChildren` against the child of c's parent that is not c, i.e., c's sibling. Algorithm 4 will compute c's sibling for us.

Algorithm 4 Return child of p that is not c

```
procedure GETSIBLING(p, c)
    if c = p_r then
        return p_r
    else
        return p_ℓ
    end if
end procedure
```

Get Uncle

The objective of the distributive rewrite is to move two `MUL` gates next to each other so they can participate in a future associative rewrite. A criterion is needed to judge if a distributive rewrite is beneficial.

This criterion centers around the future associative rewrite that this distributive rewrite facilitates, which is called the *would be* associative rewrite. If the *would be* associative rewrite can reduce the depth of the critical path, satisfying (Equation 3.1), it is beneficial to bring this rewrite to fruition. This can be done by performing the said distributive rewrite. Therefore, the merit of a distributive rewrite can be judged by that of the depth reducing ability of the corresponding *would be* associative rewrite.

To achieve the evaluation mentioned above, a comparison of the depth of c's children against the sibling of c using `CheckChildren` must be performed. However, since there could be arbitrarily many `Add` gates between c and its ancestor p, it is not possible to simply identify c's sibling for this comparison. Instead, the generalized notion of *sibling*, namely c's uncle, must be used. Here, the uncle of c is defined to be the child of p that is not a direct ancestor of c, i.e., the node that would be the sibling of c if p were c's parent. The procedure to find the uncle of c relative to p is given by Algorithm 5.

Algorithm 5 Return child of p that is not a direct ancestor of c

```
procedure GETUNCLE(p, c)
    a' ← GETPARENT(c)                          ▷a' is equal to p at this time
    while True do
        a ← a'
        a' ← GETPARENT(a)
        if a' = p then
            ▷ We have traced ancestry from c to p
            if a = p_ℓ then                    ▷c is a left ancestor of p
                return p_r                     ▷Uncle is p_r
            else                               ▷c is a right ancestor of p
                return p_ℓ
            end if
        end if
    end while
end procedure
```

3.3 Illustrative Example

Consider computed torque control (or feedback linearization) of a planar RP manipulator shown in Figure 3.5. Encrypted control of the same system was considered to reduce the number of multiplications by inspection [21]. Note that in the SHE scheme, the net depth is the direct metric related to overflow, not necessarily the number of multiplications. In contrast, this paper will minimize the depth of the expression by applying the rewrite rules. Nonlinear actuator effort to linearize the dynamics of the manipulator is given by:

$$\begin{bmatrix} \tau_1 \\ f_2 \end{bmatrix} - \begin{bmatrix} m_1 {l_1}^2 + I_1 + m_2 {d_2}^2 + I_2 & 0 \\ 0 & m_2 \end{bmatrix} (\boldsymbol{K}_p \boldsymbol{e} + \boldsymbol{K}_v \dot{\boldsymbol{e}}) + \begin{bmatrix} 2m_2 d_2 \dot{d}_2 \dot{\theta}_1 \\ -m_2 d_2 \dot{\theta}_1^2 \end{bmatrix} + \begin{bmatrix} (m_1 l_1 g + m_2 d_2 g)\cos\theta_1 \\ m_2 g \sin\theta_1 \end{bmatrix} \tag{3.2}$$

where $\boldsymbol{K}_p$= diag(K_{p1}, K_{p2}) is a proportional gain, $\boldsymbol{K}_v$ = diag(K_{v1}, K_{v2}) is a differential gain, $\boldsymbol{e} = [e_1, e_2]^T = [r_1, r_2]^T - [\theta_1, d_2]^T$ is the error between a reference vector $\boldsymbol{r}$ and the output. θ_1 is the angle of Joint 1, d_2 is the displacement of Joint 2, m is the mass, I is the inertia, g is the gravitational acceleration, and l is the location of the center of mass, where the subscripts indicate link numbers.

Define $\alpha = 2g_{pd}/(2 + T_s g_{pd})$, $\beta = (2 - T_s g_{pd})/(2 + T_s g_{pd})$ where T_s is the sampling period and g_{pd} is the bandwidth. Equation (Equation 3.2) is discretized by using bilinear transformation: $e_1[k] = r_1[k] - \theta_1[k]$, $\dot{e}_1[k] = \alpha(e_1[k] - e_1[k-1]) + \beta \dot{e}_1[k-1]$,$e_2[k] = r_2[k] - d_2[k]$, $\dot{e}_2[k] = \alpha(e_2[k] - e_2[k-1]) + \beta \dot{e}_2[k-1]$ where k the time step.

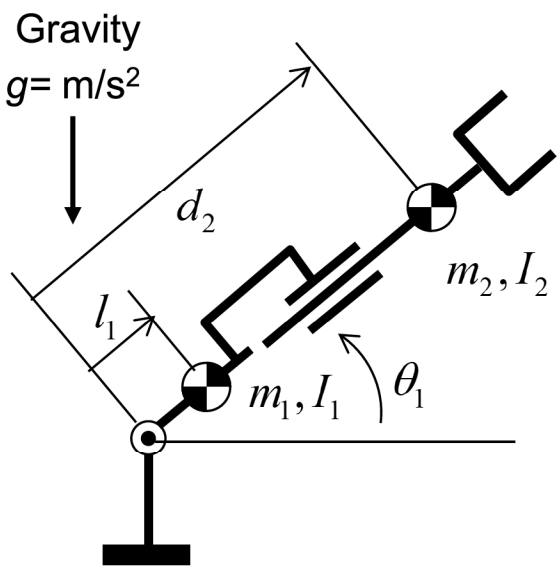


Figure 3.5: Planar RP Manipulator

$$
\begin{aligned}
\tau_1[k] = {} & (m_1 l_1^2 + I_1 + I_2)(K_{p1} + \alpha K_{v1}) r_1[k] \\
& - (m_1 l_1^2 + I_1 + I_2)(K_{p1} + \alpha K_{v1}) \theta_1[k] \\
& - (m_1 l_1^2 + I_1 + I_2) \alpha K_{v1} e_1[k-1] \\
& + (m_1 l_1^2 + I_1 + I_2) \beta K_{v1} \dot{e}_1[k-1] \\
& + m_2 (K_{p1} + \alpha K_{v1}) r_1[k] d_2^2[k] \\
& - m_2 (K_{p1} + \alpha K_{v1}) \theta_1[k] d_2^2[k] \\
& - m_2 \alpha K_{v1} e_1[k-1] d_2^2[k] \\
& + m_2 \beta K_{v1} \dot{e}_1[k-1] d_2^2[k] \\
& + 2 m_2 d_2[k] \dot{\theta}_1[k] \dot{d}_2[k] \\
& + m_1 l_1 g \cos \theta_1[k] \\
& + m_2 g \cos \theta_1[k] d_2[k]
\end{aligned}
\tag{3.3}
$$

This expression of $\tau_1[k]$ has a total depth of 6, and is represented by the arithmetic circuit in Figure 3.6. The proposed rewrite methods rearrange (Equation 3.3) so as to be of the

form:

$$
\begin{aligned}
\tilde{\tau}_1[k] = {} & d_2[k](\cos\theta_1[k]m_2g) \\
& + \cos\theta_1[k](gl_1m_1) \\
& + (2d_2[k]m_2)(\dot{d}_2[k]\dot{\theta}_1[k]) \\
& + (K_{v1}\beta m_2)(\dot{e}_1[k-1](d_2[k]d_2[k])) \\
& - (K_{v1}\alpha m_2)((d_2[k]d_2[k])(r_1[k-1]-\theta_1[k-1])) \\
& - (m_2(K_{p1}+K_{v1}\alpha))(\theta_1[k](d_2[k]d_2[k])) \\
& + (m_2(K_{p1}+K_{v1}\alpha))(r_1[k](d_2[k]d_2[k])) \\
& + (\beta(K_{v1}\dot{e}_1[k-1]))(I_2+I_1+l_1(l_1m_1)) \\
& - (K_{v1}(r_1[k-1]-\theta_1[k-1]))(\alpha(I_2+I_1+l_1(l_1m_1))) \\
& + r_1[k](K_{p1}+K_{v1}\alpha)(I_2+I_1+l_1(l_1m_1)) \\
& - \theta_1[k](K_{p1}+K_{v1}\alpha)(I_2+I_1+l_1(l_1m_1))
\end{aligned}
\tag{3.4}
$$

which has a depth of 4 as shown in Figure 3.7. Note that this rewrite was done fully automatically by applying the proposed algorithms described in the previous section.

Similarly, the other input f_2 is also discretized as:

$$
\begin{aligned}
f_2[k] = {} & m_2(K_{p1}+\alpha K_{v1})r_2[k] - m_2(K_{p1}+\alpha K_{v1})d_2[k] \\
& - m_2\alpha K_{v1}e_2[k-1] + m_2\beta K_{v1}\dot{e}_2[k-1] \\
& - m_2d_2[k]\dot{\theta}_1^2[k] + m_2g\sin\theta_1[k]
\end{aligned}
\tag{3.5}
$$

and is rewritten reducing the depth from 4 in the unsorted expression to 3 in the sorted expression. Tree expressions of f_2 are omitted.

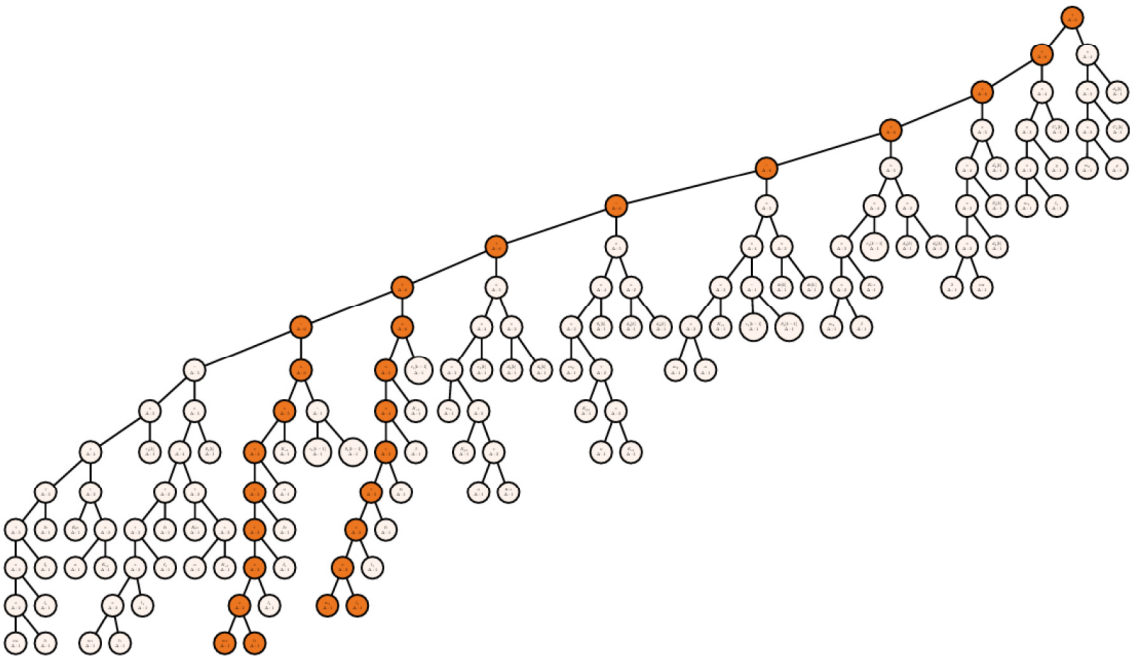


Figure 3.6: Arithmetic Circuit of (Equation 3.3). There are two problematic paths shown as the long orange legs with a depth of 6.

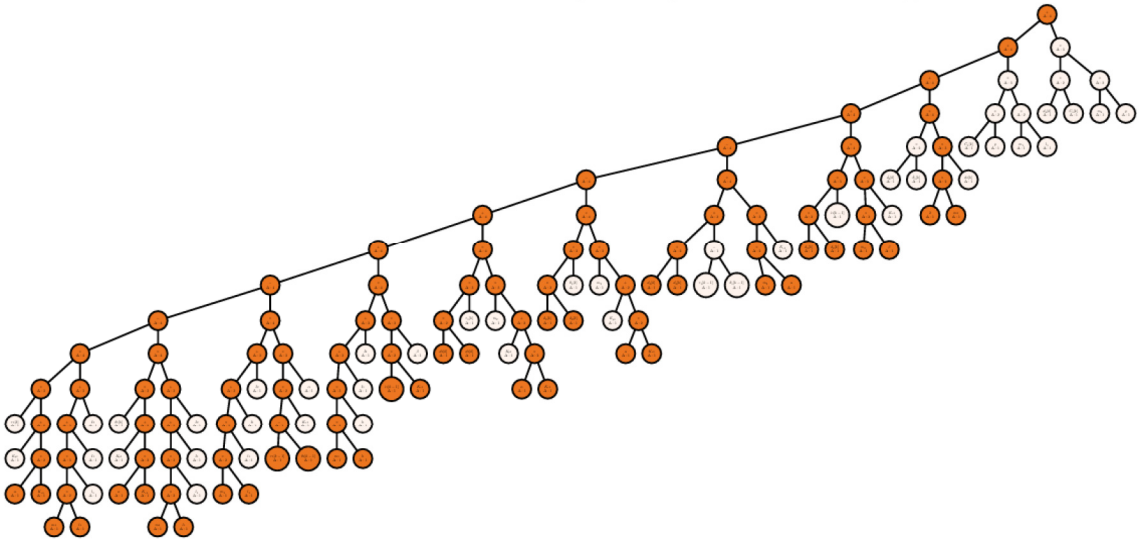


Figure 3.7: Arithmetic Circuit of (Equation 3.4) after successful rewrites. The longest paths with a depth of 4 are seen in orange.

CHAPTER 4

ENCRYPTED DYNAMICS SIMULATION PLATFORM

The research presented in this thesis aims to establish functional mockup units (FMU) co-simulation methods to simulate and evaluate encrypted dynamic systems using somewhat homomorphic encryption (SHE). The proposed approach encrypts the entire dynamic system expressions, including: model parameters, state variables, feedback gains, and sensor signals, and perform computation in the ciphertext space to simulate dynamic behaviors or generate motion commands to servo systems. The developed FMU co-simulation helps analyze the relationship between security parameters and performance. Two illustrative examples are presented and analyzed: 1) encrypted Duffing oscillator and 2) encrypted teleoperation. How the time delay due to FMU co-simulation affects the refresh rate is also reported.

4.1 Simulation Environment

4.1.1 Functional Mock-up Interface

Functional Mock-up Interface (FMI) is a tool independent standard for the Model Exchange (ME) and for Co-Simulation (CS) between different tools in a standardized format [49]. In the FMI nomenclature, a Functional Mock-up Unit (FMU) model implements one or two of FMIs. FMU is a promising candidate to become the industry standard and cross-company collaboration as it allows co-simulation of various FMUs components generated from different tools to develop complex cyber-physical systems [50]. Compared to Simulink (MATLAB, Mathworks) that is designed to model dynamic systems, FMU has the advantage of supporting more data types and language features. FMI also addresses the disadvantage of the Simulink S-function as it is easier to integrate with other simulators and takes less

memory overhead [51].

FMU is essentially an archive (i.e. a .zip file), containing a `modelDescription.xml` file in the root, and either binary or source files. In model exchange, FMU does not come with its numerical solver. FMU only provides functions to set the states and inputs and compute the state derivatives. It requires the solver in the host environment/import tool to query the derivatives and update the states of the FMU. In co-simulation, a specialized numerical solver is embedded in the FMU. The host environment only sets the inputs and time steps, and reads the outputs [52].

Since encrypted control usually requires a complex model, the authors adopted the co-simulation option. FMU can be run with `FMPy`, a free Python library, to execute FMUs that support Co-Simulation and Model Exchange and run on Windows and Linux [53]. Interested readers can visit the authors' GitHub to find relevant implementations [54].

4.1.2 FMU/Cypher Interface

Cypher Independent Interface

A system has been created by which critical parts of the FMU's calculations can be encrypted via external codes. To achieve this, we defined an interface to access cryptographic methods. FMUs can be created from vendor tools such as Simulink. Co-simulation standalone FMUs can be exported from Simulink after setting the system target file and fixed-step. By using the `S-Function` block, we can insert calls to our interface methods via function pointers and system functions (`LoadLibrary` on Windows, `dlopen` on Linux).

By separating the cryptographic routines into a standalone `.dll`, the implementation of the interface is isolated from the FMU's internal operation. This architecture enables the user to exchange a particular `cypher.dll` for a `cypher.dll` that implements the same interface for testing different cryptographic systems. The interface serves as a convenient test bed for evaluating the feasibility of different encrypted cyber-physical systems.

FMU plug-and-play setup

The modular structure of FMU may be used in conjunction with the aforementioned cypher-interface to create an encrypted cyber-physical test-bed. A particular FMU system is paired with different implementations of `cypher.dll` to test which cypher/security-parameters are best compatible with the particular system. An example of this workflow can be seen in Figure 4.1.

Data type constraints

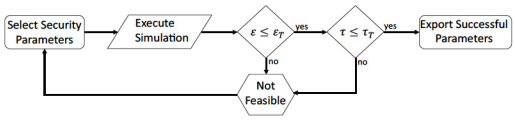
FMI was designed to be widely compatible with as many hardware architectures and operating systems as possible; as such, data is constrained to be "standard C types" [55]. This poses challenges when trying to pass encrypted values to FMUs, as cipher-text values usually cannot be represented by standard types alone. To work around these constraints, in the following case studies, all cryptographic methods have been moved into `controller.fmu`. It is usually not ideal to allow encryption and decryption to occur in a single controller. This particular configuration has been chosen to be acceptable as the same computations are being performed, just by different owning processes. As such, timing results are unlikely to be noticeably affected.

4.2 Case Studies

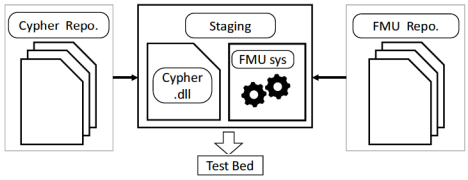
4.2.1 Limiting Equation

The primary limiting factor of Dyer's SHE is the divergence of noise introduced into the ciphertext, primarily by homomorphic multiplication. As shown in Appendix section B.2, the largest possible value among all encrypted signals, parameters, and products should be smaller than:

$$M(n, d, \lambda, \nu) := \left\lfloor \min \left\{ \frac{\sqrt[d]{\kappa}}{n+1}, \frac{\sqrt[d]{p}}{n+1} - \kappa^2 \right\} \right\rfloor \tag{4.1}$$



(a) Testbed Flow: Evaluation scheme to find successful security parameters, where ε is the measured error, ε_T the error tolerance, τ the measured simulation-cycle time, and τ_T the simulation-cycle time tolerance.



(b) Testbed staging: Construct system from FMU composition, and select the cypher to be tested. Here the "Test Bed" block refers to Figure 4.1a.

Figure 4.1: FMU/Cypher test-bed: System simulation is constructed by linking FMUs chosen from a remote repository. A cypher is then selected to be tested for compatibility with the given FMU system.

where d is the degrees of polynomial, p is the λ-bit prime, κ is the ν-bit prime–the scheme's security parameters [46, 21]. Note the left-handed ($\mathcal{L}$) and right-handed ($\mathcal{R}$) arguments of (Equation B.3) are given by: $\mathcal{L} = \frac{\sqrt[d]{\kappa}}{n+1}$, $\mathcal{R} = \frac{\sqrt[d]{p}}{n+1} - \kappa^2$.

4.2.2 Objectives

By running the simulations in FMUs, this study simulates the systems presented in the case studies similar to a real-world scenario. The simulations will validate the relationship of the security parameters in (Equation B.3) for different systems. Also, the simulation will investigate the quantization and overflow patterns as well as the choices of security parameters when $\mathcal{L}$ or $\mathcal{R}$ of (Equation B.3) dominates.

4.2.3 Duffing Oscillator

This paper will apply the SHE approach to the Duffing oscillator that includes a third degree of polynomials term. The Duffing equation is regarded as one of the prototypes for systems

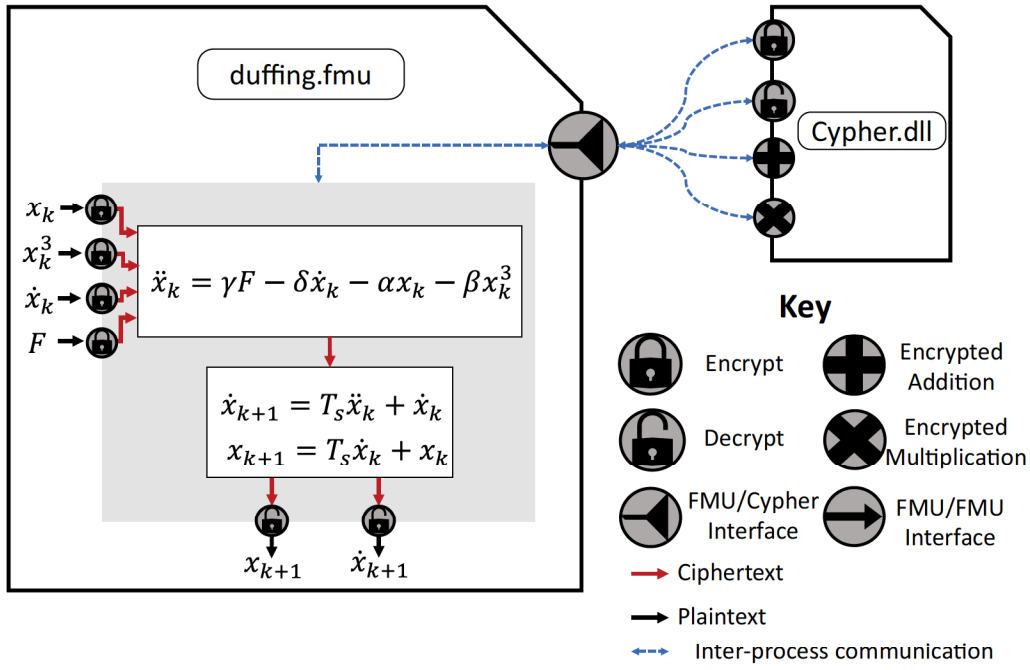


Figure 4.2: All parameters in the duffing equations were encrypted and ran in FMU, where $F = \cos(\omega t)$ is the forcing function, and $x_k = x(kT_s)$.

of nonlinear dynamics. In mechanics, it represents a class of single-degree-of-freedom systems with nonlinear stiffness.

Let the coefficients $\delta, \alpha, \beta, \gamma, \omega$ denote system damping, linear stiffness, amount of nonlinearity in the restoring force, amplitude of the driving force, and angular frequency of the force:

$$\ddot{x} + \delta\dot{x} + \alpha x + \beta x^3 = \gamma\cos(\omega t). \tag{4.2}$$

Equation (Equation 4.2) is discretized with a step time T_s for the current time $t = kT_s(k = 0, 1, 2, \cdots)$ and encrypted by using SHE, as shown in Figure 4.2. The main purpose of the encryption is to treat the parameters, $\alpha, \beta, \gamma, \delta$, as well as $\ddot{x}$, in ciphertext. On the other hand, we assume that the state variables, x and $\dot{x}$, are of interest of the user

and observed in plaintext.

$$\begin{aligned}\mathsf{Enc}(\ddot{x}_k) = \mathsf{Enc}(\gamma) \otimes \mathsf{Enc}(\cos(\omega t)) \\ \oplus \mathsf{Enc}(-\delta) \otimes \mathsf{Enc}(\dot{x}_k) \\ \oplus \mathsf{Enc}(-\alpha) \otimes \mathsf{Enc}(x_k) \\ \oplus \mathsf{Enc}(-\beta) \otimes \mathsf{Enc}(x_k^3)\end{aligned} \tag{4.3}$$

$$\mathsf{Enc}(\dot{x}_{k+1}) = \mathsf{Enc}(T_s) \otimes \mathsf{Enc}(\ddot{x}_k) \oplus \mathsf{Enc}(\dot{x}_k) \tag{4.4}$$

$$\mathsf{Enc}(x_{k+1}) = \mathsf{Enc}(T_s) \otimes \mathsf{Enc}(\dot{x}_k) \oplus \mathsf{Enc}(x_k) \tag{4.5}$$

The simulation model will then be exported into FMU and run in FMPy by setting the time step and initial conditions. Note that if the hardware resources permit in terms of multiplicative depth [25], $\mathsf{Enc}(x_k^3)$ may be replaced with $\mathsf{Enc}(x_k) \otimes \mathsf{Enc}(x_k) \otimes \mathsf{Enc}(x_k)$.

4.2.4 Teleoperation System

The SHE approach is also applied to an encrypted teleoperation system [26] where two control loops of the local and remote plants are intertwined. Let the coefficients m, b, μ, τ, and f denote system mass, damping, friction coefficient, actuator force, and external force; furthermore, let the subscript m and s denote the local and remote system. The plant dynamics is modeled as:

$$m_m\ddot{x}_m + b_m\dot{x}_m + \mu_m \operatorname{sign}(\dot{x}_m) = \tau_m + f_m \tag{4.6}$$

$$m_s\ddot{x}_s + b_s\dot{x}_s + \mu_s \operatorname{sign}(\dot{x}_s) = \tau_s - f_s \tag{4.7}$$

Evaluation of both linear and nonlinear terms must be performed in an increased number of encoding blocks. Figure 4.3 shows a possible configuration of an encrypted teleoperation system. The main concept is to encrypt shared information using a private encryption

key known *only* to the plants. Both the local and remote plants are responsible for system output measurements and encryption by using the keys. The networked controller stores encrypted system parameters, as well as encrypted output measurements received from both plants.

While there are a variety of control schemes to realize bilateral teleportation, a representative symmetric control scheme utilizing PD feedback with inertial and friction compensation is considered in this case study:

$$\begin{aligned} \tau_m = \quad & (m_m - m_{ms})\ddot{x}_m + k_p(x_s - x_m) + \\ & k_d(\dot{x}_s - \dot{x}_m) + 0.9\mu_m \operatorname{sign}(\dot{x}_m) \end{aligned} \tag{4.8}$$

$$\begin{aligned} \tau_s = \quad & (m_s - m_{ms})\ddot{x}_s + k_p(x_m - x_s) + \\ & k_d(\dot{x}_m - \dot{x}_s) + 0.9\mu_s \operatorname{sign}(\dot{x}_s) \end{aligned} \tag{4.9}$$

Inside the controller, the SHE algorithm encrypts the dynamics outputs from the local and remote plants including: accelerations, velocities, displacements, as well as gains. Then, the controller will do the computation in encryption and output the decrypted forces back to the plants.

The teleoperation system consists of three separate FMUs: local plant, controller, and remote plant. Each component is exported from the Simulink model and run in `FMPy`. During the FMU simulation, the components in the co-simulation establish communications with each other. A component publishes a specific output variable that is subscribed by other components as input. Two cycles exist in the simulations: feedback from the controller to the local plant and from the remote plant to the controller. Therefore, both plants and controllers have the same priority but cannot run in parallel.

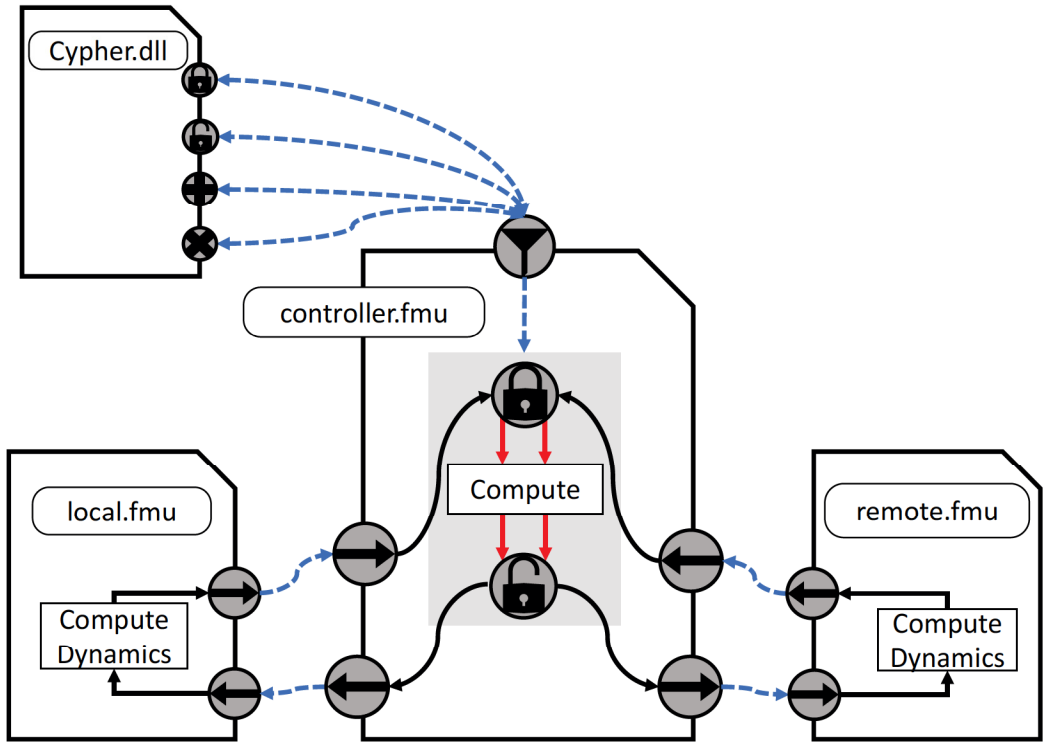


Figure 4.3: The teleoperation system consists of three separate FMU: local plant, controller, and remote plant. The controller makes calls to `cypher.dll` to perform cryptographic operations.

4.3 Results and Discussion

4.3.1 Duffing Oscillator

The nonlinear phenomenon is evident in hysteresis, which is induced by x^3 in the Duffing equation. When α and β have the same signs, the stiffness characteristic is hardening. For a hardening spring oscillator, the frequency response overhangs to the high-frequency side, as shown in Figure 4.4.

For encrypted signals, the time step (T_s) could not be smaller than 10 ms since quantization with a higher resolution is needed for a smaller T_s. Therefore, there was a balance between Δ and T_s to capture accurate data while not overflowing. When T_s is too large, the system fails to capture accurate data points. When T_s is too small, Δ needs to be smaller, resulting in numerical overflow. As shown in Figure 4.5, encryption impacted the perfor-

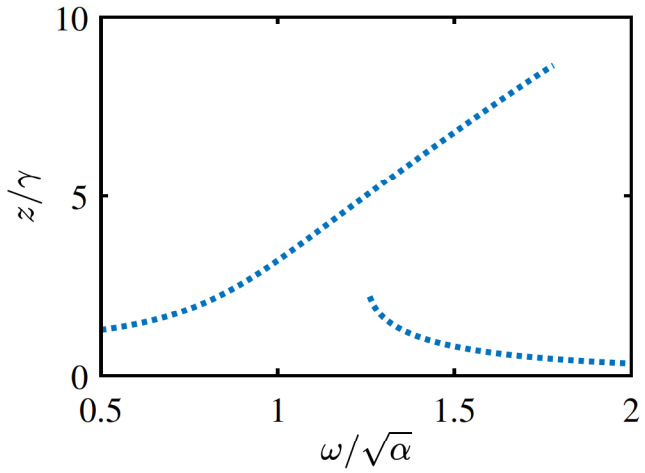


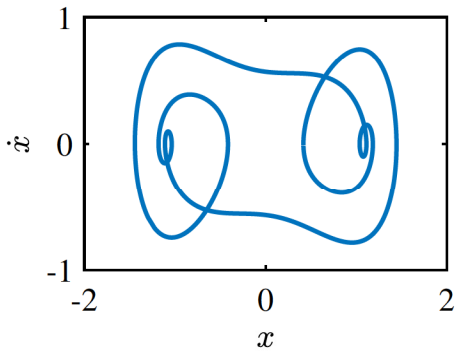
Figure 4.4: Duffing's parameter $(\alpha = \delta = 1, \gamma = 0.1, \beta = 0.04)$. Dyer's encrypted signal $(\lambda = 256, \rho = 1, \nu = 35, \Delta = 0.005)$. Frequency response overhangs to the high-frequency side in a hardening spring oscillator.

mance compared to the plaintext equation due to the resolution of the Duffing equation. The percentage of error by comparing the L2-norm of the variables was 6.93%. $\nu = 32$ where $\lambda = 192$ was the minimum security parameter to prevent numerical overflow, as shown in Figure 4.5c, which had a maximum velocity threshold of 0.45.

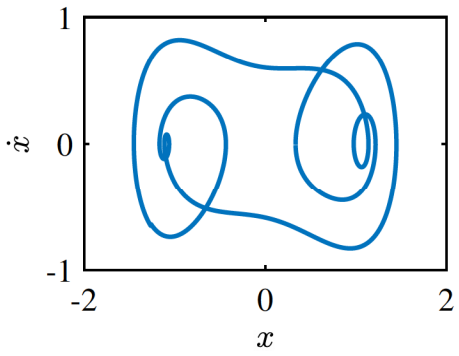
The success of the simulation result is determined by the largest encrypted number M. To investigate the $\mathcal{R}$ argument of (Equation B.3), the system was parameterised by the security parameter λ and ν. Pass/fail analysis was performed by comparing the L2-norm of the encrypted data. As shown in Figure 4.6, the $\mathcal{L}$ argument dominates and $\mathcal{R} = M$. As for a constant λ, increasing ν results in numerical overflow and failure. d, the degree of polynomial, equals to 3 because of the cubic power term results from quantization. At the boundary of pass/fail, there is a positive linear relationship between the two parameters, and λ is about as six times as large as ν.

4.3.2 Teleoperation System

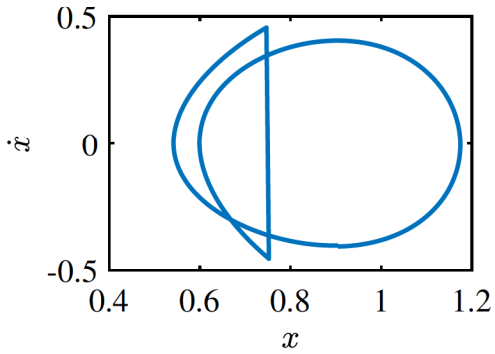
Figure 4.7 shows simulated displacements of the teleoperation system. Compared to the plaintext simulation shown in Figure 4.7a, oscillations in displacements were observed in



(a) Plaintext trajectory. Success phase plot (T_s =10 ms).



(b) SHE trajectory with security parameters: $(\lambda = 256, \rho = 1, \nu = 35, \Delta = 0.005)$. Success phase plot ($T_s = 10$ ms).



(c) SHE trajectory with security parameters: $(\lambda = 256, \rho = 1, \nu = 35, \Delta = 0.004)$. Failure phase plot ($T_s = 10$ ms).

Figure 4.5: FMU trajectory of duffing oscillator (unencrypted, encrypted-success, encrypted-fail).

the encrypted simulation due to the encoder as shown in Figure 4.7b. $\nu = 31$ and $\lambda = 125$ are the smallest security parameters to prevent numerical overflow as shown in Figure 4.7c.

Compared to the plaintext Simulink simulation that could run at a maximum time step of 20 ms for this particular system, running in FMU required a time step of at most 5 ms (Figure 4.7b). This is primarily caused by the delay in the co-simulation architecture. The local solver takes one extra time step to transfer the output of one FMU to the input of the other FMU. For example, FMU A transfers data to FMU B. FMU B takes the output from FMU A in the last time step as its input, which represents a unit of time delay between them. There is a communication delay equivalent to four time steps between the local plant's input

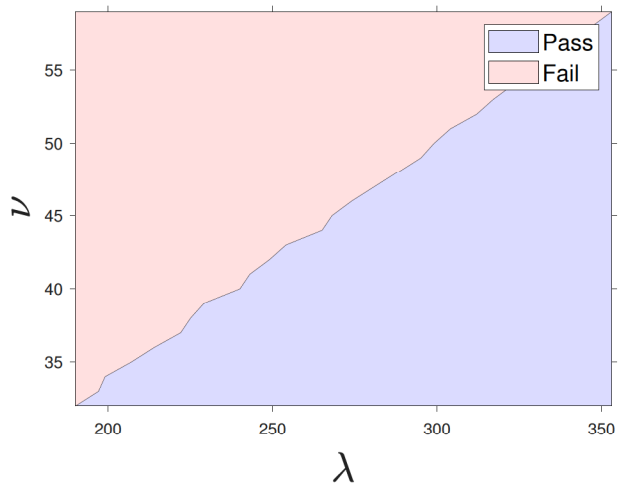
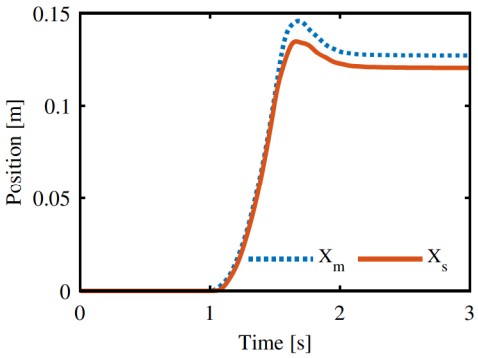


Figure 4.6: Pass/fail results of Duffing system simulations using Dyer's SHE. The system was parameterised by the security parameter λ and the encoder resolution ν. Remaining security parameters well held constant at $\rho = 1$, $\Delta = 0.005$. The system starts working from $\nu = 32$ and $\lambda = 192$.

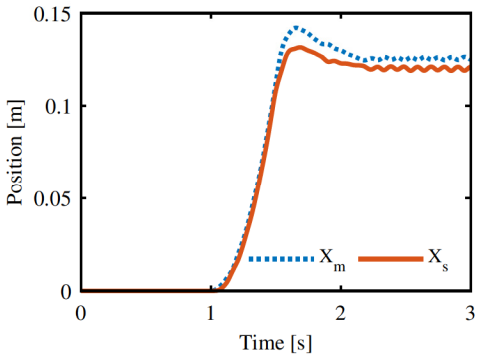
and output for the signals through the remote plant. The time delays include: one from the local plan to the controller, from the controller to the remote plant, from the remote plant back to the controller, and from the controller back to the local plant. Nevertheless, Dyer's SHE scheme has much faster refresh rate than 200 Hz to realize real-time control system.

Figure 4.8 depicts that the $\mathcal{L}$ argument dominates and $\mathcal{R} = M$. Compared to the duffing system, d decreases from 3 to 2 in the teleoperation system. The minimum start value of $\nu = 32$ is decreased by one, and the minimum value of $\lambda = 125$ is decreased by about 33%. So, the decrease of ν is due to the increase of Δ and the decrease of λ is mainly due to the decrease of d. There is a linear relationship between λ and ν. λ is about four times as large as ν, which is the product of the number of multiplication and d.

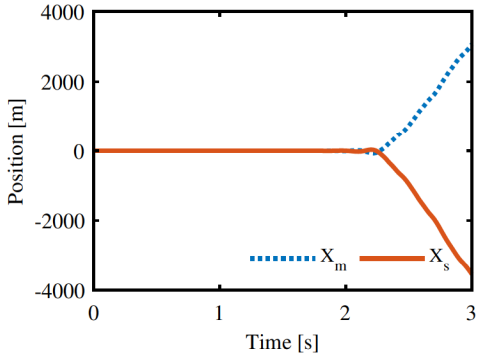
To investigate the $\mathcal{L}$ argument of (Equation B.3), the system was parameterised by the security parameter λ and the encoder resolution Δ, as shown in Figure 4.9. When Δ becomes smaller, the size of the encoded number and M increases. As shown in Figure 4.9, increases ν can increase M while ν is smaller than 36. Increasing ν can increase the $\mathcal{L}$ argument of (Equation B.3) while significantly decreasing $\mathcal{R}$. Therefore, for $\lambda = 256$, the



(a) Plaintext control ($T_s = 20$ ms)



(b) SHE control ($\lambda = 128, \rho = 1, \rho' = 32, \Delta = 0.005, T_s = 4$ ms). Oscillations are shown due to encoder.



(c) SHE control ($\lambda = 128, \rho = 1, \rho' = 32, \Delta = 0.005, T_s = 20$ ms). The step size is not small enough to permit correct computations.

Figure 4.7: FMU teleoperation results (unencrypted, encrypted-success, encrypted-fail).

$\mathcal{R}$ argument dominates and $\mathcal{L} = M$ while ν is smaller than 36. Otherwise, the $\mathcal{L}$ argument dominates and $\mathcal{R} = M$.

To sum up, the FMU simulations find the quantization and overflow pattern of encrypted control using Dyer's SHE scheme: if $\nu \geq \frac{\lambda}{2d}$, the system fails; if $\frac{\lambda}{3d} \leq \nu \leq \frac{\lambda}{2d}$, the $\mathcal{L}$ argument dominates and $\mathcal{R} = M$; if $\nu \leq \frac{\lambda}{3d}$, then the $\mathcal{R}$ argument dominates and $\mathcal{L} = M$. These inequality relationships are observed patterns found from the simulations and may not hold in general.

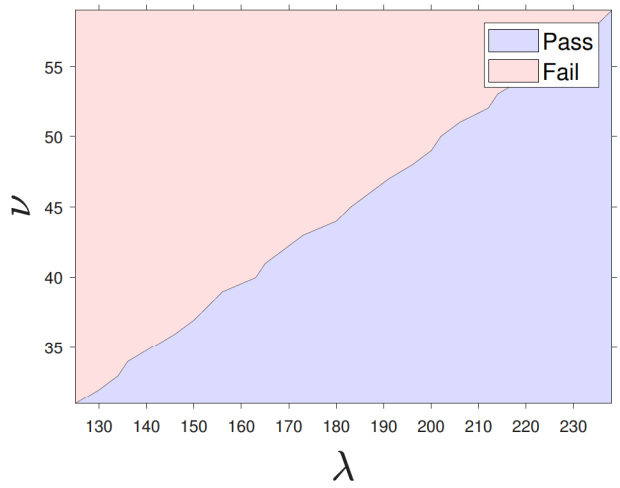


Figure 4.8: Pass/fail results of teleoperated control simulations using Dyer's SHE. The system was parameterised by the security parameter λ and the encoder resolution ν. Remaining security parameters well held constant at $\rho = 1$, $\delta = 0.01$. The system starts working from $\nu = 31$ and $\lambda = 125$.

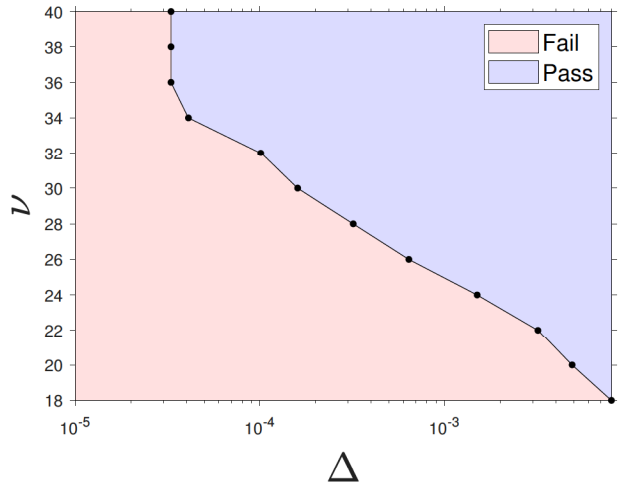


Figure 4.9: Pass/fail results of teleoperated control simulations using Dyer's SHE. The system was parameterised by the security parameter ν and the encoder resolution Δ. Remaining security parameters well held constant at $\rho = 1$, $\lambda = 256$.

CHAPTER 5

HARDWARE ACCELERATED CRYPTOGRAPHIC ARCHITECTURE

This thesis presents an investigation into the improvement of security and operation time in homomorphically encrypted systems using Field Programmable Gate Array (FPGA) technology. The primary objective is to generate keys efficiently, minimizing key sizes while maintaining security. By leveraging FPGA capabilities for key generation and key switching, smaller ciphertext sizes can be achieved, ultimately improving operation time. The thesis focuses on the development of a sensor data encryption system implemented on an FPGA board. The proposed approach enables simultaneous key generation and encryption of incoming sensor data using generated keys. The developed system implemented fixed-size random number generation and prime number checking in hardware, subsequently expanding these capabilities to produce arbitrarily sized prime numbers.

5.1 Proposed FPGA transducer node with fast key generation

The system consists of a dedicated encryption circuit `Enc` attached to sensor equipment which immediately encrypts the sensor signal and exposes only the encrypted signal. The augmented sensor module is equipped with a communication channel (e.g. a pin) over which the module recieves the key with which it is to use to preform encryption. The encrypted signal can then be sent to the controller for encrypted calculation of the control signal. A dedicated decryption circuit `Dec` is embedded into the actuator node with a communication channel in the same way as the sensor node. The decryption circuit then decrypts the control signal, using the key it received externally.

Both the sensor and actuator nodes receive their key from the same *key register*. This register is periodically overwritten by the output of a dedicated key-generation circuit `KeyGen`. Once the register is updated the `Enc`, `Dec` modules will use this updated value

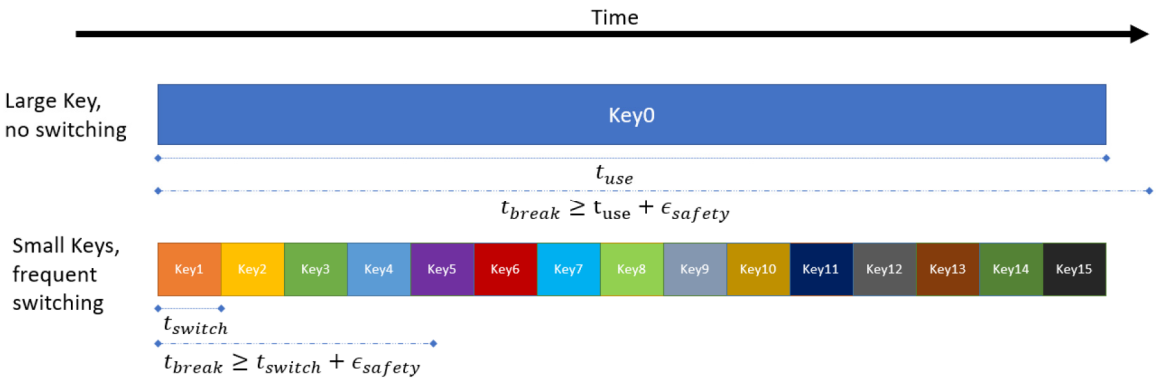


Figure 5.1: Key switching motivation: By using small keys we get faster computation time, but they can be "hacked" easier, so we need to switch keys frequently.

as their keys. This design layout can be seen in Figure 5.2

By adjusting the period T_{gen} of key generation we can compensate for weaker keys. Some care is needed during the transition from one key to the next, since the signal still in the controller will become stale when the key register is overwritten. Metrics for how often a key should be switched to ensure security of a real-time encrypted control system have been explored by [56].

Homomorphic cyphers over the integers rely on strong primes to produce secure keys [57], [58]. Therefore in order to generate a key a `PrimeGen` circuit must be constructed. Primes are generated by combination of a *Random Number Generator* (RNG) and a *Primality Test*. There are several different techniques to achieve RNG in hardware [59], [60], the presented work uses a collection of Linear Feedback Shift Registers (LFSR) which each contribute a different bit to the output and are independently seeded. The RNG output is constructed by concatenation of sufficient LFSR outputs to achieve the desired bit-length.

Prime generation can be done in one of two ways, either a computationally expensive but deterministic operation can be dispatched to produce a *provable prime*, or a more computationally feasible but probabilistic method can be dispatched to produce a *probable prime* [61]. Deterministic primality tests require significant resources, in the case of prime sieves a list of all integers up to some limit is constructed and composite numbers are subsequently removed [62]. For large primes, such as those suitable for cryptographic purposes,

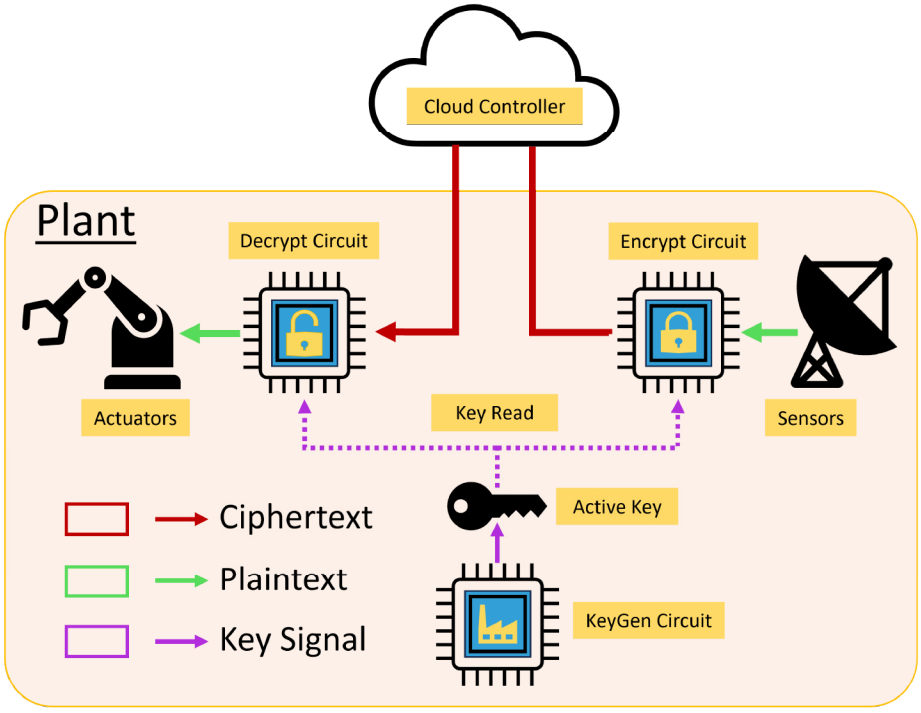


Figure 5.2: Autonomous system employing dedicated cryptographic circuits with periodic key re-generation. By embedding a `Enc` circuit into the sensor node, the system can conceal the sensor signal from the cloud. The embedded `Dec` circuit in the actuator node then retrieves the control signal calculated by the cloud. The `KeyGen` circuit creates a new *active key* periodically replacing the key used by the other circuits.

the memory requirement of such methods become infeasible. Because of the difficulties in obtaining a provable prime, we instead use probable primes with a design parameter ε which can be adjusted to control the primality probability tolerance.

The probable primes are generated by a "guess and check" method where random numbers of appropriate bit-length are constructed from the RNG module, which is sent to the Probabilistic Primality Test (PPT) module. The PPT evaluates the *candidate prime* $p_?$ and if it concludes $p_?$ is prime than successive iterations of the PPT are applied to $p_?$ until primality confidence can be brought within tolerance.

5.2 RNG using Feedback Shift Register

While true randomness is not achievable within deterministic devices such as Field Programmable Gate Arrays (FPGAs), pseudorandom number generators can be implemented utilizing a starting seed to produce random numbers. One such example is the Linear Feedback Shift Register (LFSR).

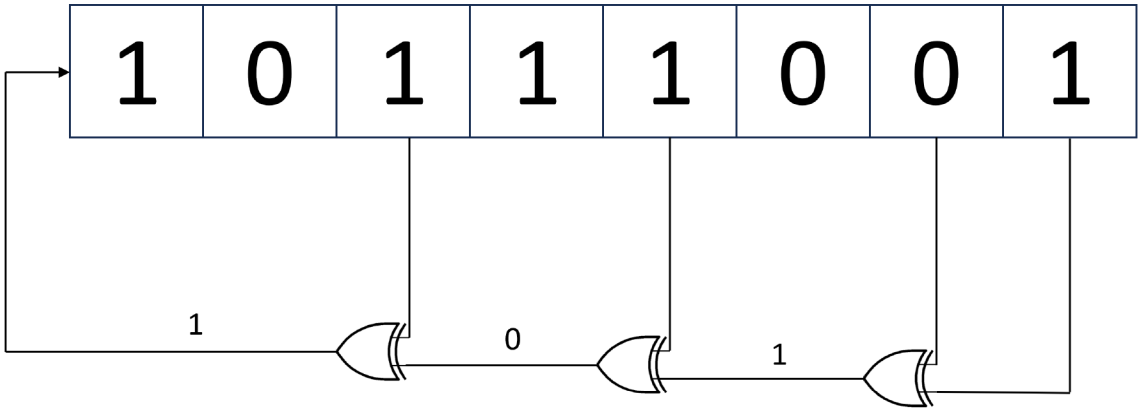


Figure 5.3: Linear Feedback Shift Register with taps at its 0th, 1st, 3rd, and 5th bits

An LFSR consists of a clocked shift register with feedback from its constituent bits, often referred to as "taps". By applying the exclusive or (XOR) operation between bits within the shift register, a new pseudorandom bit value can be introduced into the register at every clock cycle. This feedback loop is depicted in Figure 5.3.

LFSRs were chosen as the source of random numbers due to their low resource utilization and ease of implementation in hardware–as they consist of only a few gates and registers. However, they are completely deterministic designs–if an attacker knew the starting seed and the design, they would be able to predict future random number outputs, which would undermine the security of the system. Multiple instances of LFSRs (See Figure 5.4) may be employed to make this attack difficult, with only specific bits from each LFSR selected for the output random numbers. Furthermore, nondeterminism could be introduced by artificially introducing race conditions in the insertion of new bits into the LFSR.

Initially, a 128-bit LFSR generating 16-bit random numbers was chosen. For larger

random number generation requirements, multiple instances of the LFSR module (with distinct seeds) can be combined to produce arbitrarily sized pseudorandom numbers in hardware. The resulting signal through a LFSR can be seen in Figure 5.5.

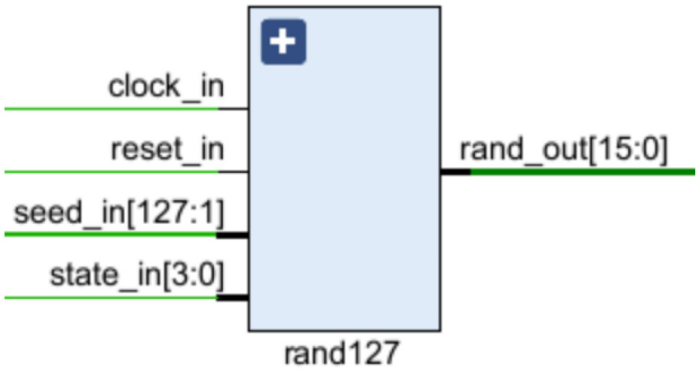


Figure 5.4: Linear Feedback Shift Register Inputs and Outputs

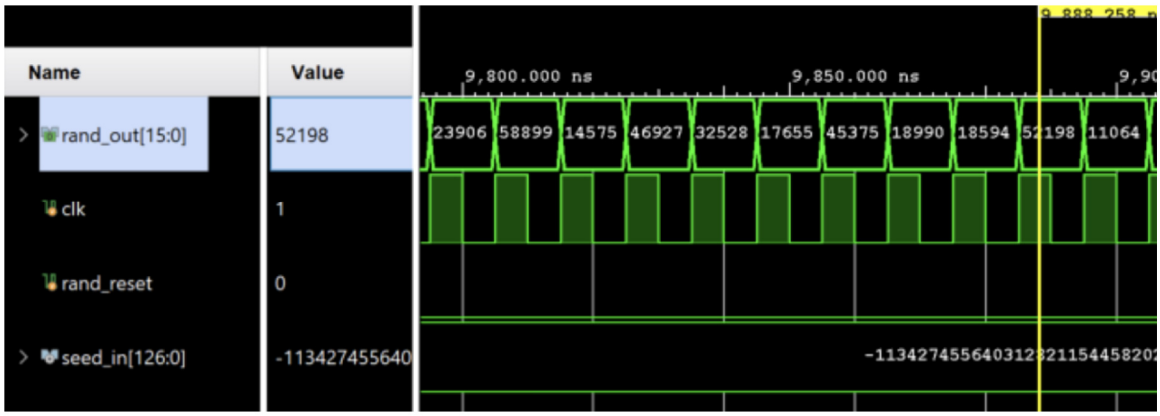


Figure 5.5: Simulated Linear Feedback Shift Register random number output signal.

5.3 Miller-Rabin Primality Check

The Miller-Rabin primality test is a probabilistic primality test based on Fermat's Little Theorem. It "checks" the primality of a number n by attempting to prove it to be composite. As opposed to deterministic method such as the Sieve of Eratosthenes, the Miller-Rabin test is much faster with significantly less memory consumption [62]. This is evidenced by their respective runtime complexities–$O(\text{sqrt}(n))$ for the Sieve of Eratosthenes and $O(k * \log^3(n))$ for the Miller-Rabin test, where k is the number of tests run [63].

Algorithm 6 Miller-Rabin Test

```
1: procedure MR(n, s)
2:     for j ← 1 to s do
3:         a ← RandomInteger(1, n − 1)
4:         if Witness(a, n) then
5:             return COMPOSITE
6:         else if
7:             thenreturn PRIME
8:         end if
9:     end for
10: end procedure
```

The `Witness` procedure of Algorithm 6 can be described as follows. Let a be a random number which is said to be a witness of n being composite, if

$$a^{n-1} \not\equiv 1 \bmod n \tag{5.1}$$

These relations are precisely the negation of the equivalence relations of Fermat's Little Theorem, giving strong evidence that n is composite [64]. However, even if n passes the above test, there is a chance that it is a strong pseudoprime, for which the corresponding value of a would be a strong liar. To remedy this, the Miller-Rabin test is performed several times on a potential prime, reducing the chances that it is a strong pseudoprime with so many strong liars [65], [66].

Let ε be the probability that n is a strong pseudoprime after k checks. Then, an upper bound can be placed on the probability that n is a strong pseudoprime as follows [63]:

$$\varepsilon < \left(\frac{1}{4}\right)^k. \tag{5.2}$$

Therefore, a corresponding lower bound can be placed on the probability that n is prime after k checks

$$P(n \text{ is prime after } k \text{ checks}) \approx 1 - \left(\frac{1}{4}\right)^k. \tag{5.3}$$

5.3.1 State Machine

The FPGA based Miller-Rabin circuit is implemented as a finite state machine with 6 states that systematically checks an input 32-bit prime number up to a specified number of checks (k, determined by setting system parameter ε) and determines whether it is prime or not.

The state machine begins with the transition to the **start** state, which takes in a potential prime n and a number of checks to do k, and resets other intermediate registers. It then transitions to the **get exp** stage, which prepares the exponent that will go into the miller rabin check. Next, in the **get random** state the random number a is retrieved. The **modular exponentiation** step then follows, finding the value of the base to the prepared exponent, and either continuing, or going to the finish state if n is composite. Next the exponent is continually squared and reduced in the **squaring** step - going to the finish state if n is composite, and only going to the check state if the congruence holds. The **check** state keeps track of the number of loops (new values of a) that have been checked, and if it is under the specified k value, the number of checks is incremented and the loop is restarted. If the number of checks is equal to k, n has gone through enough Miller Rabin checks to be deemed prime, and the transition to the **finish** state is made.

A modular exponentiation module was designed to compute $a^n \mod p$ for arbitrary $a, n, p \in \mathbb{Z}$. The authors identified some similar projects conducted in research labs; however, to our best knowledge, there is a limited number of this type of system integration work reported in the literature.

It repeatedly bit shifts n to the right (divides by 2) while squaring a and taking its modulus ($a_{new} = a_{old}^2 \mod p$) – storing any additional a terms in an intermediate register ($intermediate_{new} = a_{old} \times intermediate_{old} \mod p$ if n is odd). This process is repeated until n is 0 or 1, after which the result is calculated directly from the values of a and $intermediate$. With this modular exponentiation module, 32-bit primes could be tested – but to generate arbitrary size prime numbers, further modification was required to ensure the functionality of all steps of the module for arbitrary input sizes. To allow for this, the

module registers and wires were parametrized to a desired prime size (size of n), and the logic for getting random numbers (a in the above equations) for each iteration of Miller Rabin was changed from being fixed (to 16 bits) to building a random number from 16-bit random numbers up to the specified prime size. This logic was kept clocked/sequential – to limit the resource utilization of the miller rabin modules – as the speed increase would not have been worth the extra utilization.

The Miller-Rabin circuit behavior can be seen in Figure 5.6

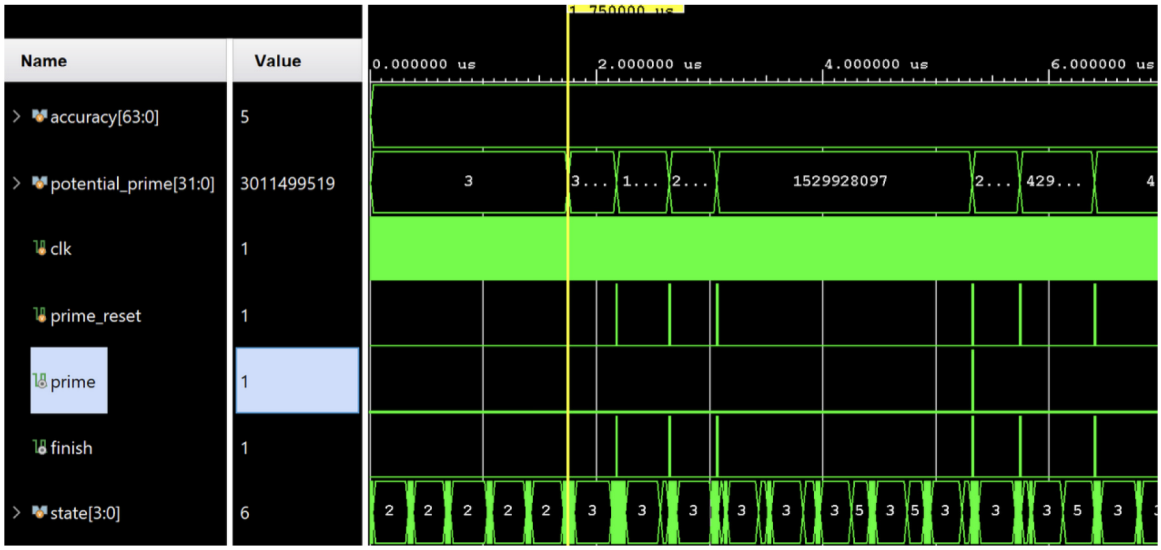


Figure 5.6: Simulated Miller Rabin primality outputs

5.4 Constructing Prime Generation Module From RNG and Miller-Rabin Modules

With these building blocks, higher level modules for the generation of keys could be generated for Dyer's HE1N scheme. The random number generation and miller rabin modules were next combined to create a prime gen module – again parametrized to allow for arbitrary prime size generation. For this module, a potential prime number is built up from 16-bit prime numbers similar to how values for a were generated in the miller rabin module. These potential primes are then fed into the miller rabin module, where the number is checked. This continues until a prime number is found, after which the module holds until it is reset. Tricks can be used to speed up the rate of prime number generation - for

example, ensuring that no potential primes are even (which can be easily implemented by ensuring that their LSB is not 0).

5.5 Cryptographic key generation

Using the prime gen module, a key gen module was then created – which syncs up 3 prime gen modules to generate a key and public modulus (κ, p, and q). This is where the strengths of the FPGA show – as while a CPU based design would have to generate κ, p, and q in series, an FPGA generates them in parallel, and is only limited by the largest prime among them (See Figure 5.7).

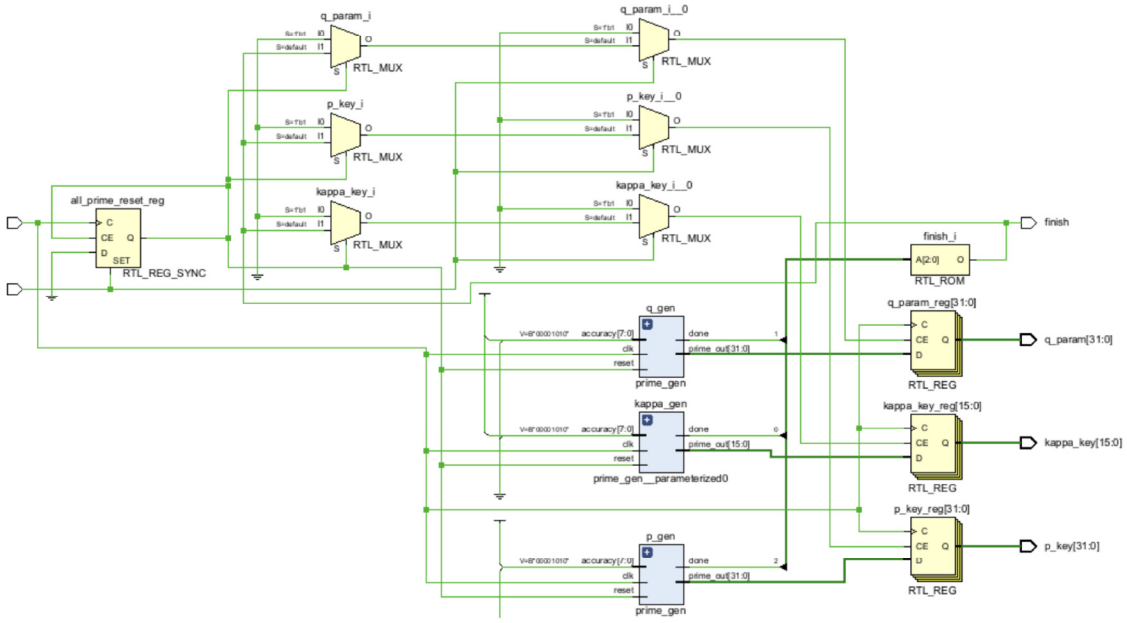


Figure 5.7: Key Gen Inputs and Outputs - including 3 Prime Gen modules

5.6 Encrypt module

An encrypt module was created to take keys from the key gen module and generate appropriately sized noise terms for r and s (See Figure 5.8). For this module, speed was crucial – much more so than everything involved with the key gen – as new data to encrypt will be coming in each clock cycle, so if the encryption process takes multiple clock cycles,

incoming data will be missed, and the analog signal will have more discretization error. To prevent this, this module was designed to be completely combinatorial – so the encryption process for a given plaintext value only takes 1 clock cycle and no data is missed. This was a time/space trade off done by instantiating as many LFSRs were needed based on the desired noise term size rather than building up the noise terms (which would take multiple clock cycles). This entails the random numbers being generated combinatorically using multiple rand modules as opposed to using a single rand module and building random numbers over multiple clock cycles as was done in the prime gen and miller rabin modules. Furthermore, to ensure the appropriate size of the generated noise terms without bit shifting that would take multiple clock cycles, a bitwise and is taken between the keys and rand module outputs to ensure that the noise terms are smaller than the keys.

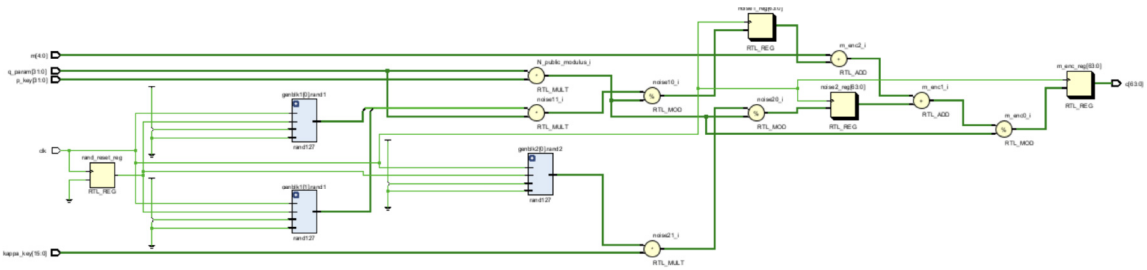


Figure 5.8: Encrypt module inputs and outputs – the random number generation modules are instantiated parametrically – using a Verilog generate block

5.7 Final Block Design

The final experimental setup consisted of an the Zynq-7000 development board with a function generator plugged to its analog input, see Figure 5.10. The Internal Logic Analyzer (ILA) IP was used to view outputs connected to internal wires in the FPGA. Keys were continuously generated, and information continuously flowed to the analog input to the encryption module, and was encrypted in real time by it. The final block design can be seen in Figure 5.9. Though a decryption module was not designed yet, using the ILA to view the values of the keys and ciphertext showed that the FPGA was correctly encrypting the

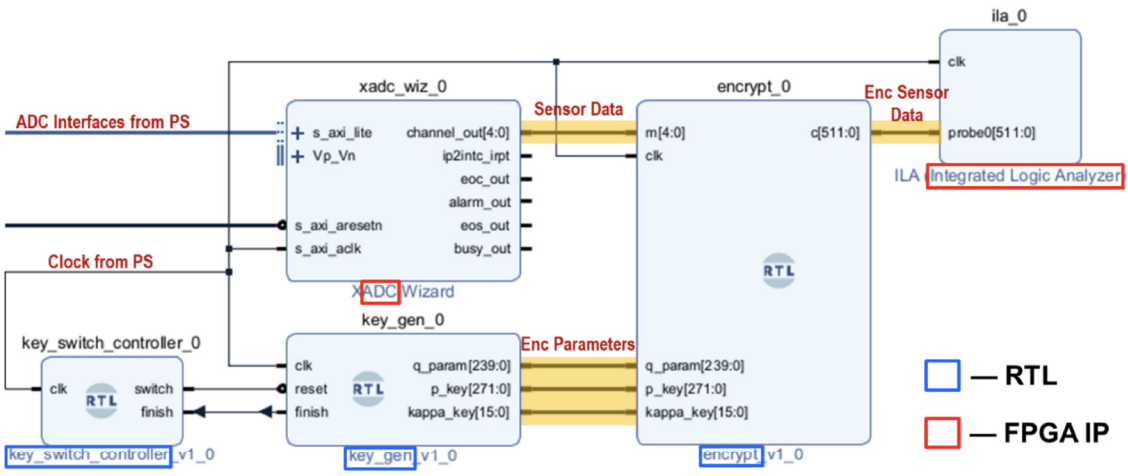


Figure 5.9: Final block diagram with all of the top level modules – security parameters are $\lambda = \eta = 32, \nu = 16$

input discretized numbers.

5.8 Results

Using these modules, an experiment was conducted to compare the prime generation time between a CPU and an FPGA. Keys from 16 bits to 2048 bits were generated on an FPGA and a CPU. Initially, only time to the first prime is recorded, which resulted in a much larger spread of results due to randomness in prime number distribution. To better characterize steady state performance of both implementations, multiple samples were taken for each bit size (5 for CPU, 30-60 for FPGA – CPU tests took very long, so more than 5 samples for each size would have significantly increased experiment time).

The times were then averaged and plotted, with their standard deviations being used for error bars. A log-log scale was used to show the trends of bit size and time over larger magnitudes. These performance behaviors can be seen in Figure 5.11. The FPGA based architecture is shown to have a consistently lower key generation time in all tested key size ranges–with key generation times between 10-100$\times$ faster–allowing for much stronger key production for the same amount of time.

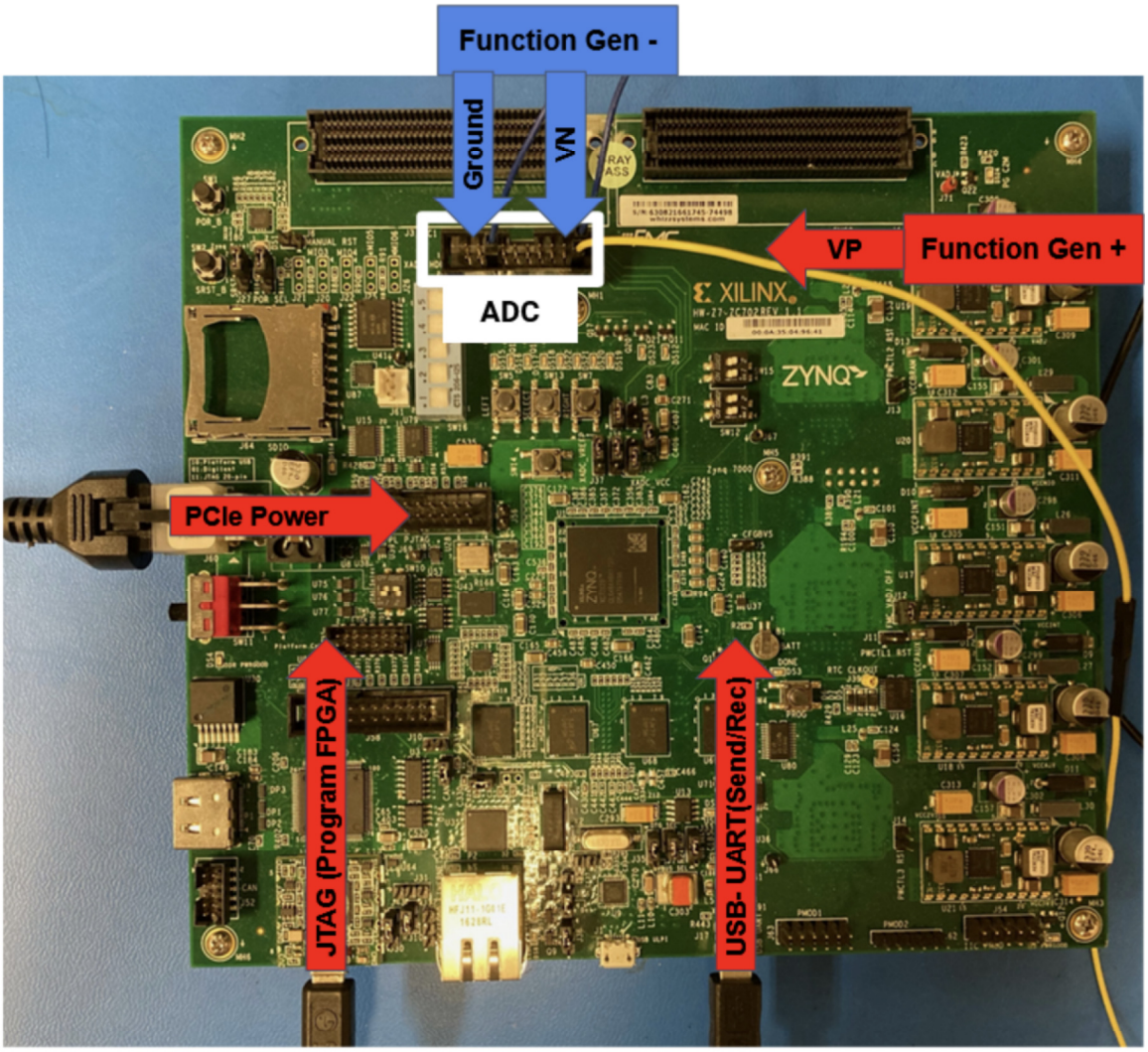


Figure 5.10: Experimental setup with function generator emulating sensor data

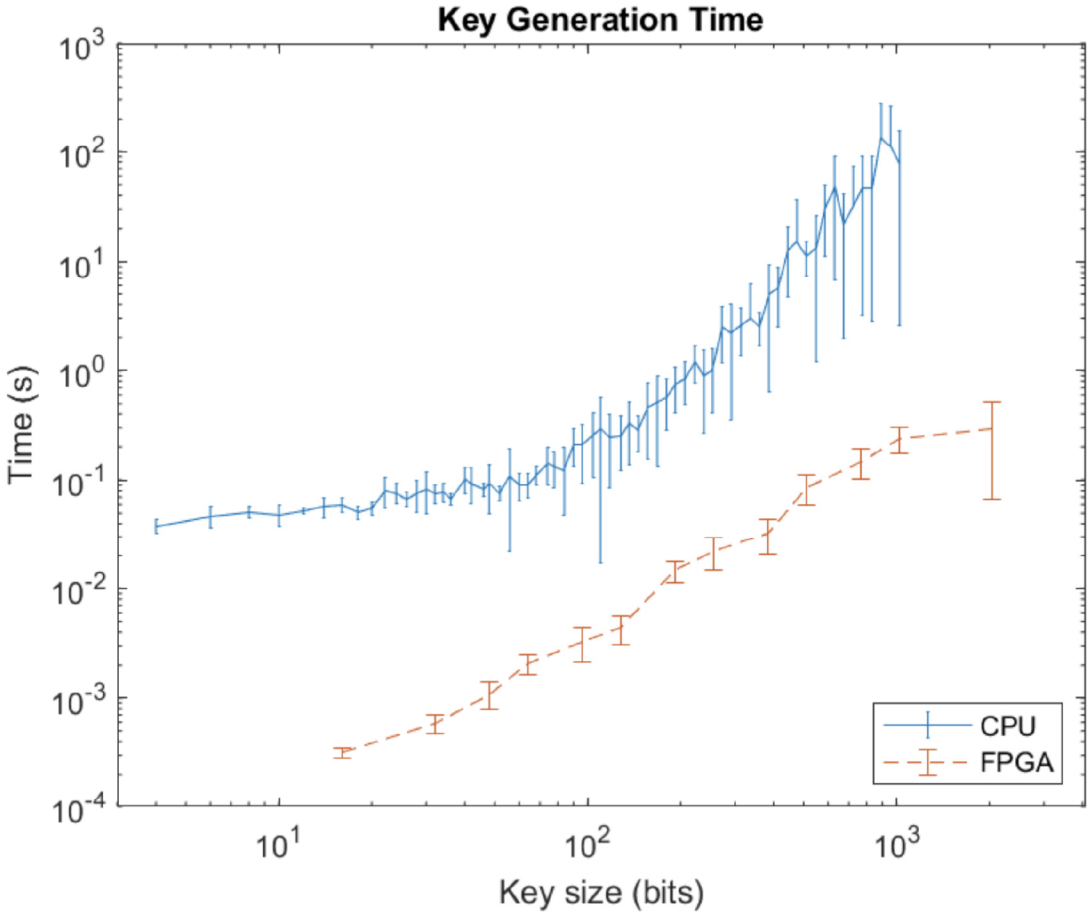


Figure 5.11: Key Generation Time on a CPU and FPGA – FPGA performance is approximately 2 orders of magnitude better at minimum

CHAPTER 6

REALIZED REAL-TIME ENCRYPTED CONTROL

This thesis seeks to understand the viability of encrypted robot control. Controllers are susceptible to malicious attacks unless controller parameters are encrypted; however, homomorphic encryption is necessary in order to allow controller mathematical operations on encrypted text, but is limited due to heavy computational overhead. Encrypted control is accomplished via the implementation of Dyer's somewhat homomorphic encryption scheme on multi and single threaded matrix transformations in order to telecommunicate movement commands between a virtual-reality joystick and a robot arm. Results find that encrypted teleoperation via the user interface is a viable encrypted controller technique, and is optimally produced on multi-threaded systems.

6.1 Methodology

6.1.1 Geometric representation and its encryption

The positions of the user control device (a hand-held virtual reality (VR) interface) and the robot end-point are represented by the coordinate frames shown in Figure 6.2. For a discretized time $t = k\Delta_t$ where Δ_t is a sampling time and k is a counter ($k = 0, 1, \cdots$), the transformation matrix ${}^{C0}\boldsymbol{T}_C$ transforming the initial user interface state ${}^{0}\boldsymbol{T}_{C0}$ to the current user interface state ${}^{0}\boldsymbol{T}_C(k)$ is given as ${}^{C0}\boldsymbol{T}_C(k) = {}^{0}\boldsymbol{T}_{C0}^{-1}\ {}^{0}\boldsymbol{T}_C(k)$. Frames $C0$ and $R0$ represent the initial states of the user interface and robot, respectively, and C and R are the current states. This transformation is applied to the initial robot state to move the robot's end-point as the desired state, i.e.,

$$ {}^{0}\boldsymbol{T}_R(k) = {}^{0}\boldsymbol{T}_{R0}\ {}^{C0}\boldsymbol{T}_C(k) = {}^{0}\boldsymbol{T}_R\ {}^{0}\boldsymbol{T}_{C0}^{-1}\ {}^{0}\boldsymbol{T}_C(k). \tag{6.1} $$

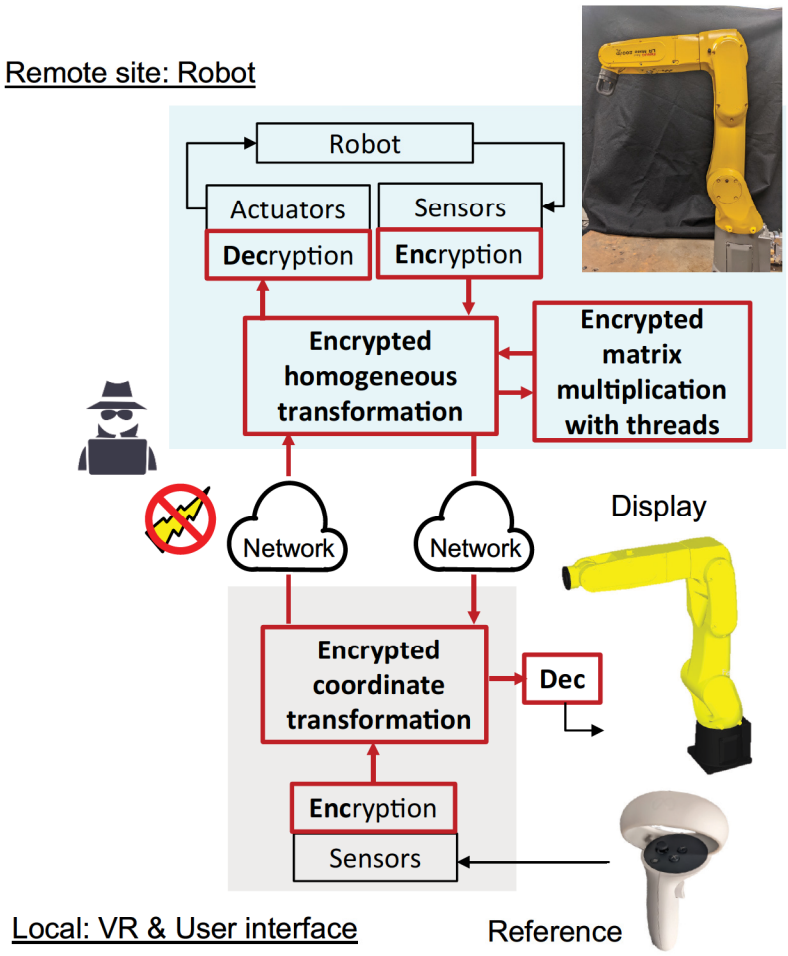


Figure 6.1: Encrypted teleoperation of a robot manipulator.

A mathematical representation to evaluate the matrix multiplicative depth (mmd) is introduced. Reducing multiplicative depth optimizes matrix multiplication time and prevents overflow of the encryption scheme. A multiplicative depth of a square $n \times n$ matrix is given by the maximum among all element-wise multiplicative depth, $l^{max}(*)$:

$$\mathrm{mmd}(\mathsf{Enc}(\boldsymbol{T})) := \max_{1 \le i,j \le n} l^{max}(\mathsf{Enc}(T_{ij})) \tag{6.2}$$

For example, multiplication between two encrypted matrices in ciphertext results in a multiplicative depth of one:

$$\mathrm{mmd}(\mathsf{Enc}({}^{0}\boldsymbol{T}_1) \circledast \mathsf{Enc}({}^{1}\boldsymbol{T}_2)) = 1 \tag{6.3}$$

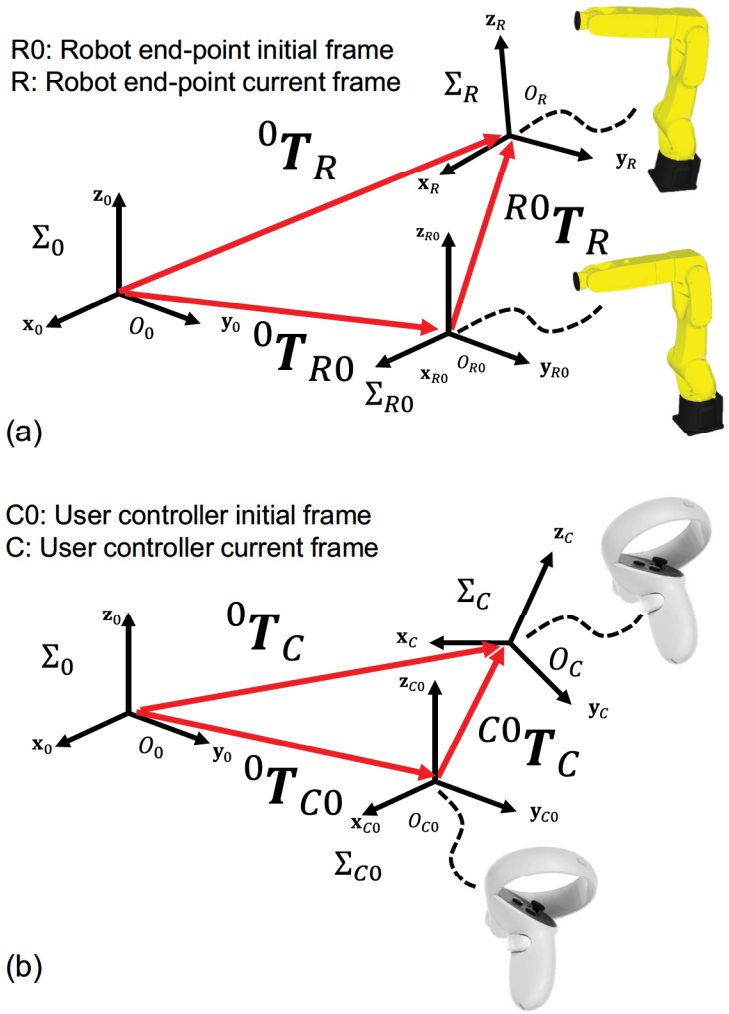


Figure 6.2: Coordinate frames; (a) Initial and current robot position, (b) Initial and current input position.

Consider the encryption of (Equation 6.1) with SHE, $\mathsf{Enc}({}^0\boldsymbol{T}_{Ed}(k))$. Note that only the relative displacement from the VR interface's initial state to the current state is used for robot control. The initial states of the robot and VR interface are fixed and stored in the system as constants. For improved security, ideally, the constant matrices, ${}^0\boldsymbol{T}_{C0}$ and ${}^0\boldsymbol{T}_{R0}$, are encrypted at the beginning (on the robot side and on the user side, respectively) and stored as in ciphertext, not in plaintext:

$$\mathsf{Enc}({}^0\boldsymbol{T}_R(k)) = \mathsf{Enc}({}^0\boldsymbol{T}_{R0}) \circledast \mathsf{Enc}({}^0\boldsymbol{T}_{C0}^{-1}) \circledast \mathsf{Enc}({}^0\boldsymbol{T}_C(k)) \tag{6.4}$$

As a result, $\mathrm{mmd}(\mathsf{Enc}({}^0\boldsymbol{T}_R(k)) = 2$. If a user wishes to reduce the multiplicative depth by one, bootstrapping can be applied on the robot side to replace the first two terms with $\mathsf{Enc}(\mathsf{Dec}(\mathsf{Enc}({}^0\boldsymbol{T}_{R0}) \circledast \mathsf{Enc}({}^0\boldsymbol{T}_{C0}^{-1})))$ without risking revealing either ${}^0\boldsymbol{T}_{R0}$ or ${}^0\boldsymbol{T}_{C0}^{-1}$ as it is performed only once.

6.1.2 Encrypted matrix multiplication with threads

Consider the matrices

$$\begin{aligned} \mathbf{A} &= \Big[\alpha \times \gamma\Big] \\ \mathbf{B} &= \Big[\gamma \times \beta\Big] \\ \mathbf{\Gamma} = \mathbf{AB} &= \Big[\alpha \times \beta\Big] \end{aligned} \tag{6.5}$$

To calculate $\mathbf{\Gamma}$, $\alpha\beta$ calculations of the form:

$$\begin{aligned} &\mathsf{Enc}\left(\boldsymbol{AB}\right)_{ij} = \tilde{\mathbf{\Gamma}}_{ij} = \\ &\left(\tilde{A}_{i1} \otimes \tilde{B}_{j1}\right) \oplus \left(\tilde{A}_{i2} \otimes \tilde{B}_{j2}\right) \oplus ... \oplus \left(\tilde{A}_{i\gamma} \otimes \tilde{B}_{j\gamma}\right) \end{aligned} \tag{6.6}$$

must be performed, where $\tilde{x} = \mathsf{Enc}(x)$. Each calculation considers γ encrypted multiplications with average time μ, and $\gamma - 1$ encrypted additions with average time σ. Then the computation time for a single entry Γ_{ij} is given by $\xi_\gamma = \gamma\mu + (\gamma - 1)\sigma$. However, in general, homomorphic multiplicative operations take significantly longer than homomorphic additive operations, i.e., $\mu \gg \sigma$. As such we will make the following simplification $\xi_\gamma \approx \gamma\mu$.

Using ξ_γ we can construct the time complexity for a single-threaded and multi-threaded implementation of the matrix product. For a single threaded implementation, each Γ_{ij} must be sequentially processed, thus $\mathcal{O}\left(\alpha\beta\xi_\gamma\right)$. While it is not typical to consider the runtime of "elementary operations," in big O analysis, it is justified in this setting as homomorphic

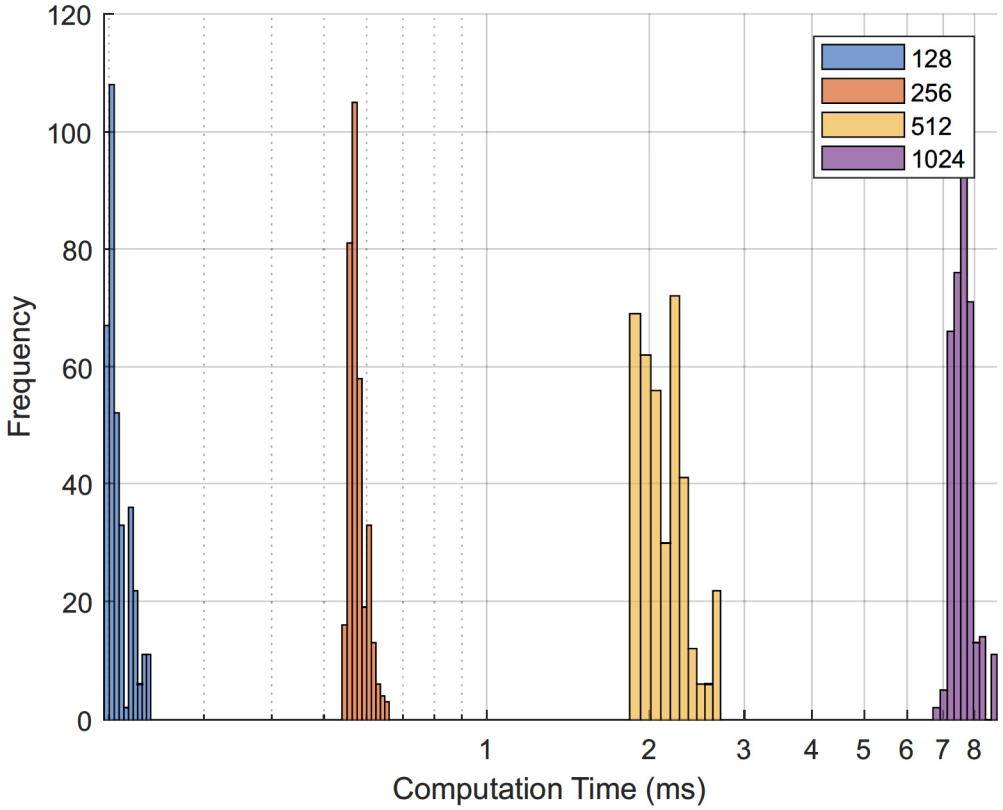


Figure 6.3: Computation time histogram for Method 1 (series), showing distribution of computation times for for security parameter λ of varying bit lengths with semi log scale. Each set is represented with 10 bins across its range.

multiplication significantly impact performance [67].

Speedup can be achieved by delegating the computation of each $\mathbf{\Gamma}_{ij}$ to its own thread. In the limit that the number of system cores N, approaches the number of elements to compute i.e. $N \to \alpha\beta$, then the complexity reduces to $\mathcal{O}(\xi_\gamma)$.

The effectiveness of parallelism is tested by implementing two methods, each tested separately and analyzed for efficiency in timing. Figure 6.6 shows the following methods: Method 1 (series) executes on a single thread, carrying out standard matrix multiplication in which each dot product awaits a preceding operation to complete before performing its operation. Method 2 (parallel) separates each dot product of a matrix multiplication into its own thread, allowing each dot product to be calculated in parallel with the others.

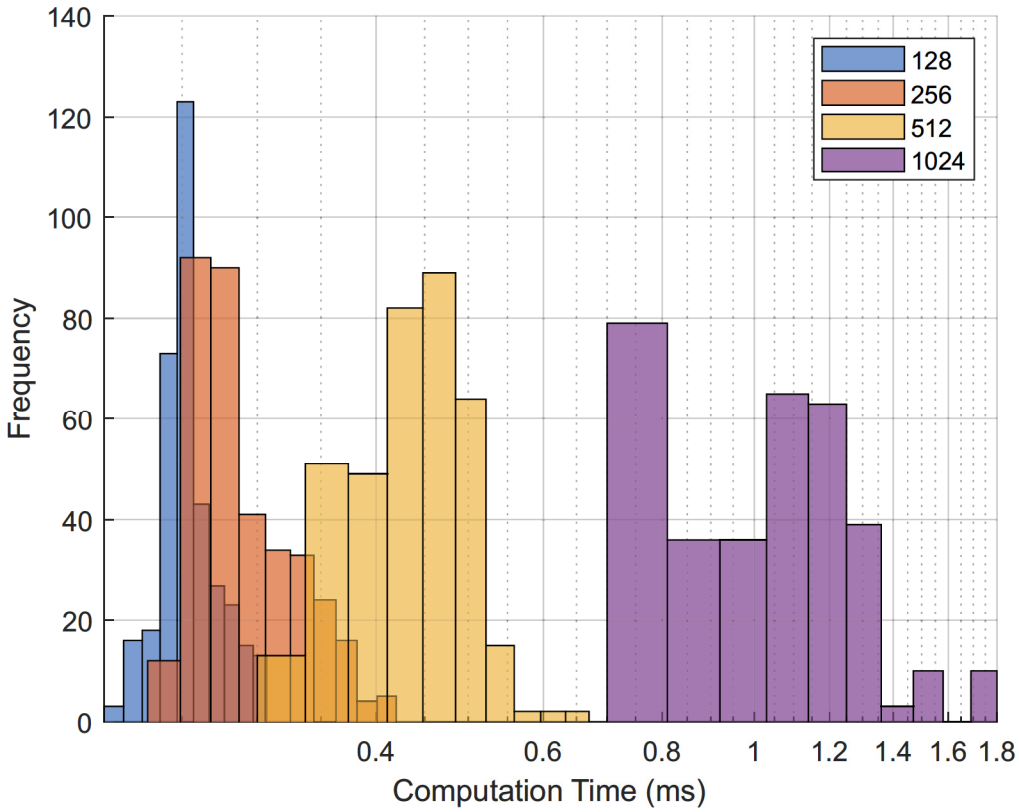


Figure 6.4: Computation time histogram for Method 2 (parallel), showing distribution of computation times for security parameter λ of varying bit lengths with semi log scale. Each set is represented with 10 bins across its range.

The performance of each method was evaluated during system operation with a timer. A total of 500 samples were taken from each operation, and its distribution was plotted as a histogram for varying λ between 128 to 1024. The histogram in Figure 6.4 shows that across an increasing range of λ, the average computation time only increased to 1.8 ms. In contrast the histogram of Method 1 in Figure 6.3 shows the computation time without parallelism. Here we see clearly that the distribution increases exponentially with increasing λ.

6.2 Encrypted Teleoperation System

6.2.1 Implementation

Point-To-Point (PTP) direct control of a robot manipulator (FANUC LR Mate 200iD/7L) requires homogeneous matrix transformations. These transformations are implemented with SHE and threading at various levels to enhance performance.

A virtual reality (VR) headset (Meta Quest 2) is the user input device in this system. The VR joystick was attached to a controllable Universal Robots 6R Manipulator in order to repeat the experimental path consistently as shown in Figure 6.5. The position of the controller was acquired on the local computer. OpenVR API (Valve Corporation) was used in this implementation. The current pose of the user interface ${}^{0}\boldsymbol{T}_{C}(k)$ is acquired, and the homogeneous transformation matrix ${}^{C0}\boldsymbol{T}_{C}$ from the initial pose of the controller to the current pose is processed, and then encrypted to be processed in the operator module. This portion of the system is considered local, with full access to all keys as in Figure 6.7.

The operator module was implemented with C#, The first part is the cipher, which implements the Dyer's SHE algorithm. The second part of the operator module implemented matrix multiplication in cipherspace. This part is representative of a "cloud" controller shown in Figure 6.7 , and will not access any information needed for encryption and decryption. To that end, the two matrices ${}^{0}\boldsymbol{T}_{R0}$ and ${}^{C0}\boldsymbol{T}_{C}(k)$ are encrypted when it is received by the thread on which operation are being carried out. A client on the remote side will receive the encrypted message from the cloud. The final command is decrypted and sent to the robot controller (RoboDK). Once RoboDK receives this command, a trajectory is sent to the FANUC Manipulator, resulting in a translation and rotation to the desired robot pose as shown in Figure 6.7. It is noted that this implementation uses threads in favor of processes for homomorphic operations. An inter-process model would be a closer representation of an ideal teleoperation system, as there is extra communication overhead. While there is no inter-process communication, threads are being executed in an asynchronous

manner as would be expected of a multi-process system. For the purposes of this study, a multi-threaded architecture can sufficiently represent a successful physical system that only operates on encrypted information.

6.2.2 Threading

As established in subsection 6.1.2, encrypted matrix operations can be sped up by distributing computational overhead across multiple processors. Multi-threading has been chosen to implement the parallelism proposed in Method 2 in Figure 6.6. Performance of a multi-threaded program is dependent on the size and scheduling of the tasks put on each thread. The size of the task should not be smaller than the overhead to start threads. If this occurs, implementing parallelism could lead to performance degradation. Computation time of homomorphic operations has been evaluated in detail in [26], including that of Dyer's SHE algorithm. Based on these findings, it is expected that for low security messages with lighter computational load, gains from parallelism will be comparable to the threading scheduling overhead. However, with increase in message length, parallel computing gains as discussed in will dominate as discussed in subsection 6.1.2.

6.2.3 Simulation

To motivate our choice of security parameters we ran simulation of single-threaded vs multi-threaded execution. To simplify this choice we define all security parameters to be in terms of one parameter λ by $\rho = 10\log_2(\lambda)$
and $\rho' = 2\rho$. This parameterization ensures that the cypher has sufficient entropy to prevent a cyphertext attack [68]. Furthermore, by parameterizing ρ and ρ' by λ, we have collapsed the parameter space to a single dimension, thus making a sweep of parameter space far less computationally burdensome. Ultimately, the security of the cyphertext against brute force attacks will be determined by the bitlength of the encrypted values. With the above parameterization we can see that bitlength is related to λ as shown in Figure 6.8. The

computation time to complete one matrix multiplication with and without threading was simulated on a 11th Gen Intel(R) Core(TM) i7-1165G7 CPU. Computation was averaged over 10 runs for each choice of λ. The results of this simulation can be seen in Figure 6.9.

Notice in Figure 6.9 that for low security parameters the series method performs better than the multi-threaded method. This is due to the overhead of creating and managing multiple threads, and indicates the threaded tasks require so little computational effort that the threading overhead is actually detrimental. Typically, we want our security parameters to be as large as possible, so such a situation is unlikely to occur in a production system.

6.3 Experimental Results & Analysis

In order to verify encrypted operation, experimental data was collected with security parameter λ chosen from the set $\{128, 256, 512, 1024\}$. Figure 6.10 presents translation data of the end effector for a test case with λ = 1024. These results show that the robot manipulator closely tracked the VR remote while command calculations were performed in cipherspace. For example, at the lower left corner, the robot tracked better with Method 2 (parallel) since the reference command was generated faster than Method 1 (series). This observation is expected to hold for complex path tracking.

Table 6.1 shows the median deviation of the robot end effector from the ideal path in millimeters at varying security parameters. The deviation showed non-normal distribution for all trials tested with the Shapiro-Wilks normality test with 95% confidence. One tailed Mann Whitney U test was performed at the significance level of 0.0125 to determine if deviation of the system was significantly smaller when method 2 is used over method 1. Results showed that there was a significant difference in deviation between methods for cases $\lambda = 512$ and 1024.

All λ values show similar results, confirming the viability of encrypted robot control through the VR system. Additionally tested was computation time again with varying λ chosen from the same set $\{128, 256, 512, 1024\}$. Results of this test can be seen in Ta-

Table 6.1: Median Robot - VR path deviation in mm at varying security parameter λ for Method 1 and Method 2, and Mann Whitney U test score U by λ ($^*U < 0.0125$)

λ	Method 1: Series	Method 2: Parallel	U
128	16.41	16.16	.342
256	21.64	23.26	.953
512	31.60	18.48	$< .001^*$
1024	42.27	18.87	$< .001^*$

ble 6.2. Observe that Method 2 gains benefit as the λ parameter increases, approximately seven times faster than that of the series implementation in the $\lambda = 1024$ case.

One tailed Mann Whitney U test was performed to determine if the positive shift in the overall distribution was statistically significant. It was found that reduction in computational time is significant from $\lambda = 256$ and above. Although there is significant time increase for $\lambda = 256$ case, amount of reduction is suspected to be too small to result in performance increase. For $\lambda = 512$ and 1024 case, time reduction and performance increase were consistent.

Table 6.2: Median and standard deviation, computation time of reference command generation (ms), and Mann Whitney U test score U by λ ($^*U < 0.0125$) .

λ	Method 1: Series		Method 2: Parallel		U
	Mdn	σ	Mdn	σ	
128	0.204	0.001	0.252	0.023	1
256	0.574	0.026	0.285	0.042	$< .001^*$
512	2.098	0.193	0.441	0.056	$< .001^*$
1024	7.661	0.383	1.068	0.236	$< .001^*$

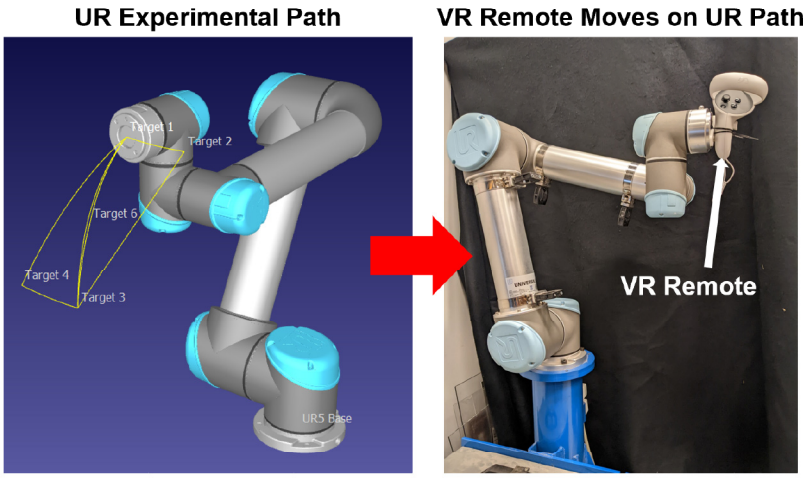
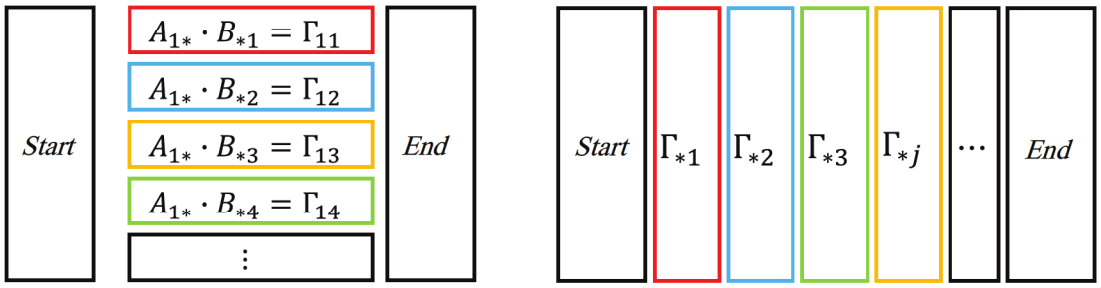


Figure 6.5: VR experimental path for UR-VR Mount.



(a) Method 1: Series Matrix Operations (b) Method 2: Parallel Matrix Operations

Figure 6.6: Threading methods

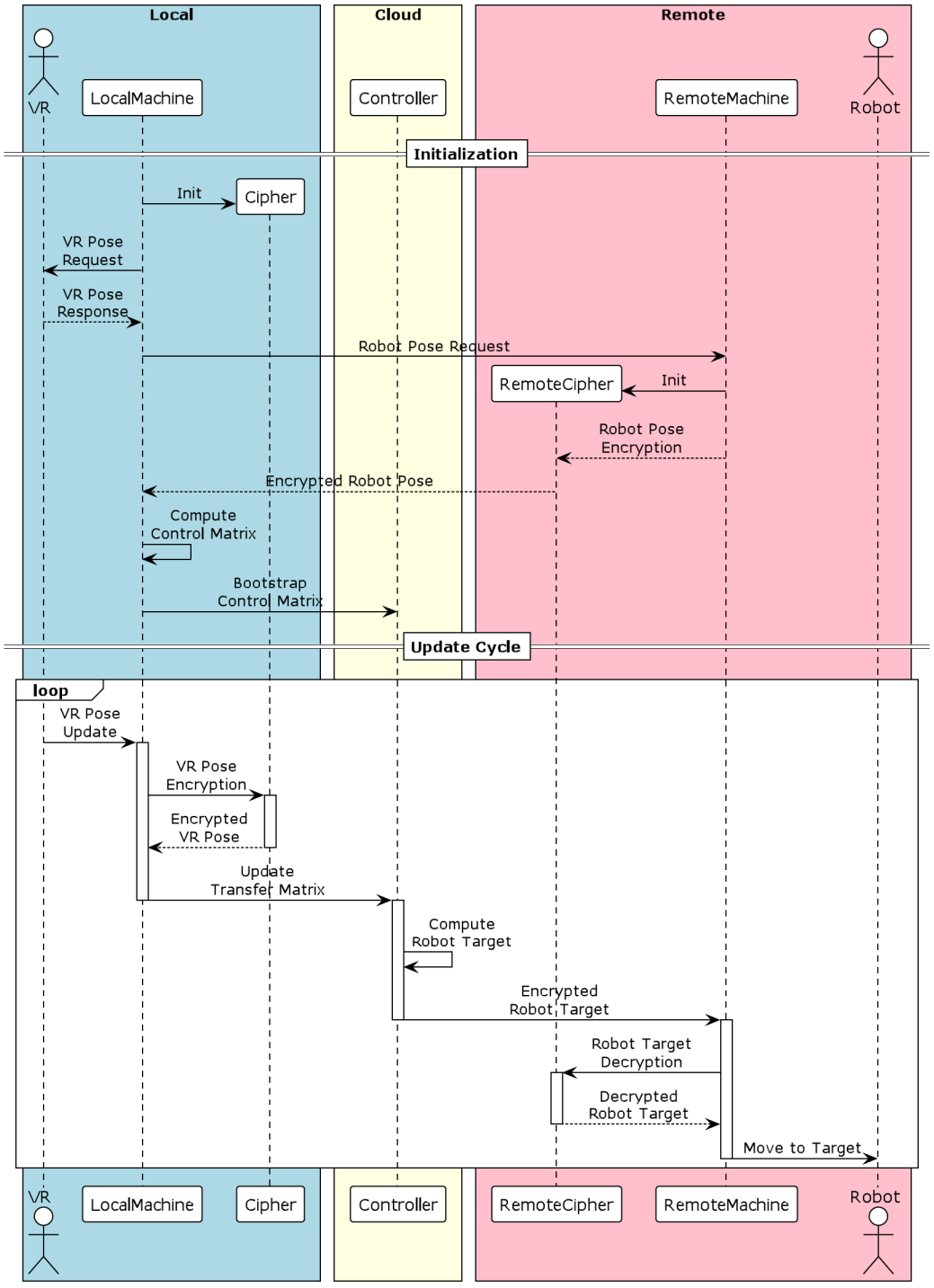


Figure 6.7: Sequential order of implementation

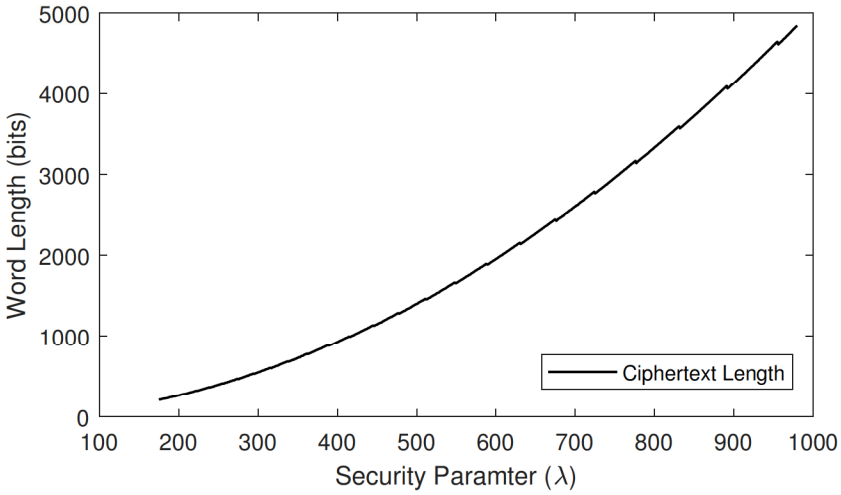


Figure 6.8: Ciphertext bitlength with respect to lambda

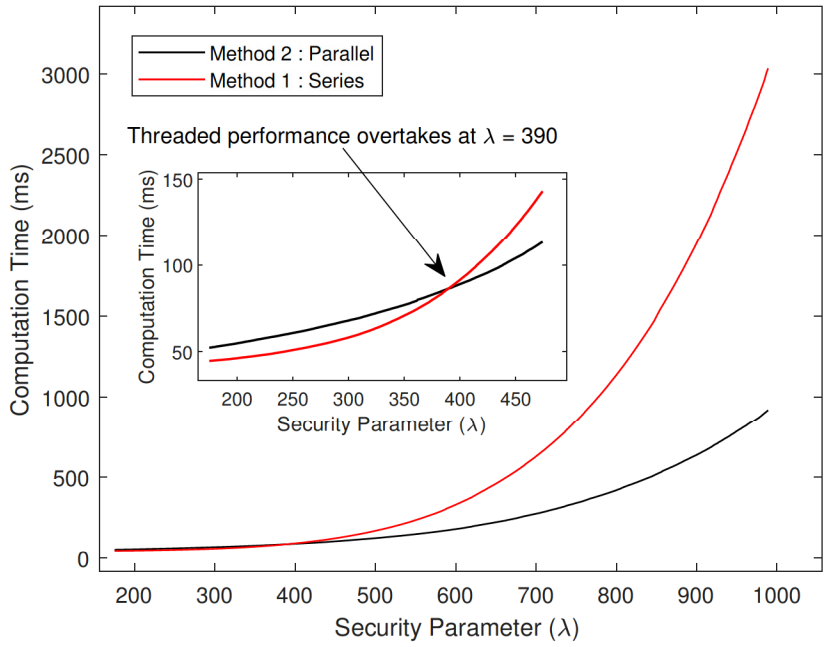


Figure 6.9: Computation time for given lambda

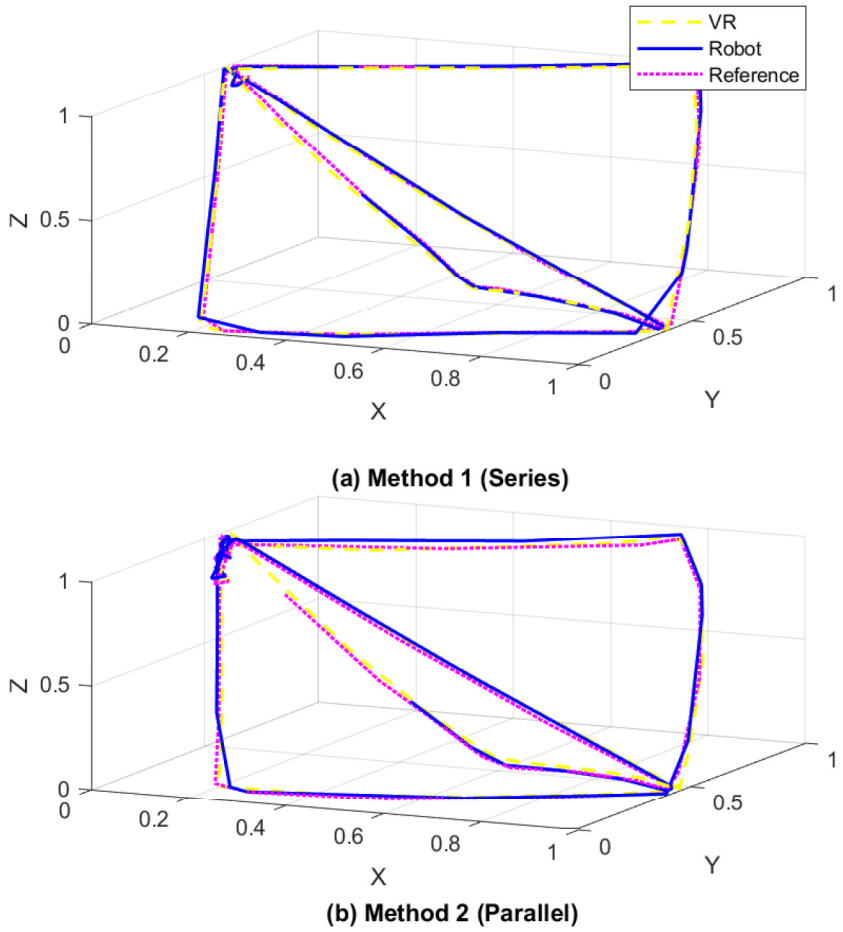


Figure 6.10: Position comparison between VR, reference , and robot trajectories for $\lambda = 1024$.

CHAPTER 7

CONCLUSION

With the ever-expanding integration of networked autonomous systems, our society becomes increasingly dependent on digital asset. While there is no question the positive impact this has had, such dependency presents a new threat of cyber-attacks crippling these systems.

This thesis presented expression rewrite rules for encrypted dynamic control schemes to reduce the depth of somewhat homomorphic encryption and to improve numerical stability. Algorithms to automate the associative and distributive rewrite rules have been implemented into depth analyzer software. An illustrative numerical example was presented to demonstrate the usefulness of the proposed approach. Note that further depth reduction may be possible if specific plant information is known. For example, operations between link parameters may be grouped as a combined parameter if they are known to be time-invariant. Grouping of operations between state variables before encryption is also conceptually possible. Further improvements including experimental demonstration will will be reported in our future publications.

This thesis also presented a physical system implementing a somewhat homomorphic encryption algorithm to secure communication between distinct systems in the form of a VR hand-held remote controller and a 6DOF manipulator. Messages between the local and remote systems were in the form of a homogeneous transformation matrix. A method to accelerate computationally heavy, encrypted calculations exploiting the parallel nature of matrices was devised and evaluated for algorithmic complexity. The new method was shown to significantly reduce update times once security parameters were large enough to offset communication overhead inherent in parallel computing. Among the tested security parameters the median computation time was 7.22 times faster compared to when paral-

lel computation was not applied. This improvement in computational time was shown to meaningfully decrease positional deviation of the robot end effector from the ideal path up to 2.25 times. Usage of a larger security parameter will result in even larger performance gains. Further research would implement filtering in cipherspace in the form of an extended Kalman filter and model predictive controllers leveraging improved performance of matrix operations.

This thesis also proposed a concept to enhance cyber security for networked motion controllers via somewhat homomorphic encryption. We have demonstrated the feasibility of encrypting the entire motion control scheme of a teleoperated system, such that real time performance is still possible. This thesis has identified large integer arithmetic as the main source of computational burden. Specialized hardware and algorithms could mitigate these issues.

Note that the algorithm proposed by [46] is a symmetric-key encryption system, though not as secure as an asymmetric-key system, does allow both homomorphic addition ad multiplication. This improves security, by removing holes in the system at the controller. Dyer's encryption is not stable for all security and encoding parameters. If (Equation A.3) and (Equation A.4) are not satisfied, the scheme ceases to be homomorphic.

This thesis also presented a FMU co-simulation environment for various components in an encrypted dynamic system using SHE. The architecture consists of external codes that implement the encrypted calculations and FMU's dynamic systems. The feasibility of performing the co-simulation in FMU was demonstrated in two case studies. The FMU co-simulation presented the success/fail scenarios for both systems and showed the choices of security parameters when $\mathcal{L}$ or $\mathcal{R}$ of (Equation B.3) dominates. This thesis discussed how the relationship among the security parameters and time delay in co-simulation impacts the simulation performance. The developed interface may serve as a convenient test bed for evaluating the numerical stability of different cryptographic systems.

This thesis also presented a realization of fast cryptographic key generation on a FPGA

board. Both random number generation with feedback shift registers and the Miller-Rabin primality check were implemented. The realized FPGA key generation was confirmed to be 2 orders of magnitude faster than that with a CPU. Future work includes the management of key switching and further performance validation.

Appendices

Appendix A

Dyer's Cryptosystem

This study adopts the SHE algorithm proposed in [68] that can be summarized as follows:

Gen**:** Set security parameters λ, ρ, ρ'. Let:

$$\nu = \rho' - \rho \tag{A.1}$$

$$\eta = \frac{\lambda^2}{\rho'} - \lambda \tag{A.2}$$

Randomly choose a λ-bit prime p, a ν-bit prime κ, and an η-bit prime q. Generate a key $k = (\kappa, p)$ and publish $N = pq$. In range of plaintext integer numbers: $\mathcal{M} = \{0, 1, 2, ..., M-1\}$, to compute any polynomial expression: $P(m_1, m_2, ..., m_n)$ and $P(m_1 + s_1\kappa, m_2 + s_2\kappa, ..., m_n + s_n\kappa)$ up to the degree of d, key lengths κ and p are lower-bounded by the power of d given by:

$$\kappa > (n+1)^d M^d \tag{A.3}$$

$$p > (n+1)^d (M + \kappa^2)^d \tag{A.4}$$

where $s_i \in \{0, 1, ..., \kappa - 1\} (i = 1, ..., n)$ are random integers.

Enc: Plaintext $m \in \mathcal{M}$ is encrypted by:

$$c = m + s\kappa + rp \mod N \tag{A.5}$$

where $s \in \{0, 1, ..., \kappa - 1\}$ and $r \in \{0, 1, ..., q - 1\}$ are random noise.

Dec: Ciphertext $c \in \mathcal{C}$ is decrypted by:

$$m = (c \mod p) \mod \kappa \tag{A.6}$$

Add: Additive homomorphism $\mathsf{Enc}(m) \oplus \mathsf{Enc}(m') \mod N = \mathsf{Enc}(m+m'), \forall m, m' \in \mathcal{M}$ is realized if:

$$m + m' < \kappa \tag{A.7}$$

$$(m + m') + (s + s')\kappa < p \tag{A.8}$$

where s' is random noise corresponding to m'.

Mult: Multiplicative homomorphism $\mathsf{Enc}(m) \otimes \mathsf{Enc}(m') \mod N = \mathsf{Enc}(mm'), \forall m, m' \in \mathcal{M}$ is realized if:

$$mm' < \kappa \tag{A.9}$$

$$mm' + (ms' + m's + ss'\kappa)\kappa < p \tag{A.10}$$

Equations (Equation A.3), (Equation A.4), (Equation A.7), (Equation A.8), (Equation A.9), and (Equation A.10) are conditions that must be satisfied at all times.

Appendix B

Quantization

B.1 Quantization Error

Any encryption algorithms can treat only plaintext integer numbers $m \in \mathcal{M}$. Real numbers used in control schemes must be mapped onto $\mathcal{M}$, which is equivalent to quantization of parameters and signals using an encoder and decoder:

$$\mathsf{Ecd}_\Delta : \mathbb{R} \to \mathbb{Z} : x \mapsto \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor \tag{B.1}$$

$$\mathsf{Dcd}_\Delta : \mathbb{Z} \to \mathbb{R} : m \mapsto \Delta m \tag{B.2}$$

where $\Delta \in (0, 1)$ is a sensitivity factor. Consider $Q := \mathsf{Dcd}_\Delta \circ \mathsf{Ecd}_\Delta$ that functions as a quantizer. Then, the quantization error of Q is bounded by $\Delta/2$, namely $|x - Q(x)| \leq \Delta/2$. Note that Δ cannot be arbitrarily small due to the risk of overflow.

B.2 Quantization to prevent overflow in SHE

From (Equation A.3) and (Equation A.4) it follows:

$$M(n, d, \lambda, \nu) := \left\lfloor \min \left\{ \frac{\sqrt[d]{\kappa}}{n+1}, \frac{\sqrt[d]{p}}{n+1} - \kappa^2 \right\} \right\rfloor \tag{B.3}$$

The factor Δ should satisfy $\mathsf{Ecd}_{\Delta}(x_{\max}) < M$ where $x_{\max}$ is the largest possible value among all signals, parameters, and products between them, achieving $\mathsf{Dcd}_{\Delta}(\mathsf{Dec}(\mathsf{Enc}(\mathsf{Ecd}_{\Delta}(x)))) \approx x$. Note that Δ's depth is accumulated by each multiplication, for example, $\mathsf{Enc}(\mathsf{Ecd}_{\Delta}(x)) \otimes \mathsf{Enc}(\mathsf{Ecd}_{\Delta}(x')) = \mathsf{Enc}(\mathsf{Ecd}_{\Delta^2}(xx'))$, where the depth of each term on the left-hand side is one, but that on the right-hand side is two.

REFERENCES

[1] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *System Sciences (HICSS), 2016 49th Hawaii International Conference on*, IEEE, pp. 3928–3937, ISBN: 0769556701.

[2] A. Teixeira, D. Pérez, H. Sandberg, and K. H. Johansson, "Attack models and scenarios for networked control systems," in *Proceedings of the 1st international conference on High Confidence Networked Systems*, pp. 55–64.

[3] L. Thames and D. Schaefer, *Cybersecurity for industry 4.0*. Springer, 2017, ISBN: 3319506595.

[4] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE international conference on automation, quality and testing, robotics*, IEEE, pp. 1–4, ISBN: 1479937320.

[5] Y. Z. Lun, A. D'Innocenzo, F. Smarra, I. Malavolta, and M. D. Di Benedetto, "State of the art of cyber-physical systems security: An automatic control perspective," *Journal of Systems and Software*, vol. 149, pp. 174–216, 2019.

[6] S. M. Dibaji, M. Pirani, D. B. Flamholz, A. M. Annaswamy, K. H. Johansson, and A. Chakrabortty, "A systems and control perspective of cps security," 2019.

[7] Z. A. Biron, S. Dey, and P. Pisu, "Resilient control strategy under denial of service in connected vehicles," in *2017 American Control Conference (ACC)*, pp. 4971–4976.

[8] S. Amin, A. A. Cárdenas, and S. S. Sastry, "Safe and secure networked control systems under denial-of-service attacks," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 31–45.

[9] F. Farokhi, *Privacy in Dynamical Systems*. Springer, 2020, ISBN: 981150492X.

[10] P. Hespanhol, M. Porter, R. Vasudevan, and A. Aswani, "Dynamic watermarking for general lti systems," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 1834–1839.

[11] P. Hespanhol, M. Porter, R. Vasudevan, and A. Aswani, "Statistical watermarking for networked control systems," in *2018 Annual American Control Conference (ACC)*, 2018, pp. 5467–5472.

[12] M. Porter, P. Hespanhol, A. Aswani, M. Johnson-Roberson, and R. Vasudevan, "Detecting generalized replay attacks via time-varying dynamic watermarking," *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3502–3517, 2021.

[13] M. Porter *et al.*, "Detecting deception attacks on autonomous vehicles via linear time-varying dynamic watermarking," in *2020 IEEE Conference on Control Technology and Applications (CCTA)*, 2020, pp. 1–8.

[14] J. E. Sullivan and D. Kamensky, "How cyber-attacks in ukraine show the vulnerability of the us power grid," *The Electricity Journal*, vol. 30, no. 3, pp. 30–35, 2017.

[15] A. B. Alexandru, M. Morari, and G. J. Pappas, "Cloud-based mpc with encrypted data," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 5014–5019.

[16] M. S. Darup, A. Redder, and D. E. Quevedo, "Encrypted cloud-based mpc for linear systems with input constraints," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 535–542, 2018.

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

[18] J. Kim *et al.*, "Encrypting controller using fully homomorphic encryption for security of cyber-physical systems," *IFAC-PapersOnLine*, vol. 49, no. 22, pp. 175–180, 2016.

[19] Y. Lin, F. Farokhi, I. Shames, and D. Nešić, "Secure control of nonlinear systems using semi-homomorphic encryption," in *IEEE Conference on Decision and Control*, Miami Beach, FL, 2018, pp. 5002–5007.

[20] A. B. Alexandru and G. J. Pappas, "Encrypted LQG using labeled homomorphic encryption," in *ACM/IEEE International Conference on Cyber-Physical Systems*, Montreal, 2019, pp. 129–140.

[21] K. Teranishi, K. Kogiso, and J. Ueda, "Encrypted feedback linearization and motion control for manipulator with somewhat homomorphic encryption," in *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, IEEE, 2020, pp. 613–618, ISBN: 1728167949.

[22] K. Teranishi, M. Kusaka, N. Shimada, J. Ueda, and K. Kogiso, "Secure observer-based motion control based on controller encryption," in *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 2978–2983, ISBN: 1538679264.

[23] Y. Qiu and J. Ueda, "Encrypted motion control of a teleoperation system with security-enhanced controller by deception," in *Dynamic Systems and Control Conference*, vol. 59148, American Society of Mechanical Engineers, 2019, V001T07A006, ISBN: 0791859142.

[24] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–35, 2018.

[25] S. Kosieradzki, Y. Qiu, K. Kogiso, and J. Ueda, "Rewrite rules for automated depth reduction of encrypted control expressions with somewhat homomorphic encryption," in *2022 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2022, pp. 804–809.

[26] S. Kosieradzki, X. Zhao, H. Kawase, Y. Qiu, K. Kogiso, and J. Ueda, "Secure teleoperation control using somewhat homomorphic encryption," in *Modeling, Estimation and Control Conference 2022*, 2022, pp. 6836–6843.

[27] W. Xu, Y. Zhan, Z. Wang, B. Wang, and Y. Ping, "Attack and improvement on a symmetric fully homomorphic encryption scheme," *IEEE Access*, vol. 7, pp. 68 373–68 379, 2019.

[28] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

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

[30] 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, 1985.

[31] M. Schulze Darup, A. B. Alexandru, D. E. Quevedo, and G. J. Pappas, "Encrypted control for networked systems: An illustrative introduction and current challenges," *IEEE Control Systems Magazine*, vol. 41, no. 3, pp. 58–78, 2021.

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

[33] K. Kogiso, "Attack detection and prevention for encrypted control systems by application of switching-key management," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, pp. 5032–5037, ISBN: 1538613956.

[34] A. Sultangazin and P. Tabuada, "Towards the use of symmetries to ensure privacy in control over the cloud," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, pp. 5008–5013, ISBN: 1538613956.

[35] 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)*, IEEE, pp. 5020–5025, ISBN: 1538613956.

[36] Y. Lin, F. Farokhi, I. Shames, and D. Nešić, "Secure control of nonlinear systems using semi-homomorphic encryption," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, pp. 5002–5007, ISBN: 1538613956.

[37] R. Fritz, M. Fauser, and P. Zhang, "Controller encryption for discrete event systems," in *2019 American Control Conference (ACC)*, IEEE, pp. 5633–5638, ISBN: 1538679264.

[38] K. Kogiso, "Upper-bound analysis of performance degradation in encrypted control system," in *2018 Annual American Control Conference (ACC)*, IEEE, pp. 1250–1255, ISBN: 1538654288.

[39] A. B. Alexandru, K. Gatsis, Y. Shoukry, S. A. Seshia, P. Tabuada, and G. J. Pappas, "Cloud-based quadratic optimization with partially homomorphic encryption," *arXiv preprint arXiv:1809.02267*, 2018.

[40] M. S. Darup, A. Redder, I. Shames, F. Farokhi, and D. Quevedo, "Towards encrypted mpc for linear constrained systems," *IEEE Control Systems Letters*, vol. 2, no. 2, pp. 195–200, 2017.

[41] J. Kim *et al.*, "Encrypting controller using fully homomorphic encryption for security of cyber-physical systems," *IFAC-PapersOnLine*, vol. 49, no. 22, pp. 175–180, 2016.

[42] 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.

[43] J. Ueda and T. Yoshikawa, "Force-reflecting bilateral teleoperation with time delay by signal filtering," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 613–619, 2004.

[44] F. Boemer, S. Kim, G. Seifu, F. D. de Souza, V. Gopal, *et al.*, *Intel HEXL (release 1.2)*, https://github.com/intel/hexl, 2021.

[45] Boost, *Boost C++ Libraries*, http://www.boost.org/, Last accessed 2015-06-30, 2015.

[46] J. Dyer, M. Dyer, and J. Xu, "Practical homomorphic encryption over the integers for secure computation in the cloud," in *IMA International Conference on Cryptography and Coding*, Springer, 2019, pp. 44–76.

[47] P. Aubry, S. Carpov, and R. Sirdey, "Faster homomorphic encryption is not enough: Improved heuristic for multiplicative depth minimization of boolean circuits," in *Cryptographers' Track at the RSA Conference*, Springer, 2020, pp. 345–363.

[48] S. Carpov, P. Aubry, and R. Sirdey, "A multi-start heuristic for multiplicative depth minimization of boolean circuits," in *International Workshop on Combinatorial Algorithms*, Springer, 2017, pp. 275–286.

[49] T. Blochwitz *et al.*, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," in *Proceedings of the 9th International Modelica Conference*, The Modelica Association, 2012, pp. 173–184, ISBN: 978-91-7519-826-2.

[50] C. Bertsch, J. Neudorfer, E. Ahle, S. S. Arumugham, and K. Ramachandran, "Fmi for physical models on automotive embedded targets," in *Proceedings of the 11th International Modelica Conference*, The Modelica Association, 2015, ISBN: 978-91-7685-955-1.

[51] *Functional mock-up interface*, https://fmi-standard.org/docs/3.0/, May 2022.

[52] C. Gomes, G. Abbiati, and P. G. Larsen, "Seismic hybrid testing using fmi-based co-simulation," in *Proceedings of 14th Modelica Conference 2021*, The Modelica Association, 2021, ISBN: 978-91-7929-027-6.

[53] *Fmpy (version 0.3.12)*, https://github.com/CATIA-Systems/FMPy, Aug. 2022.

[54] X. Zhao, S. Kosieradzki, and J. Ueda, *Encrypted Simulation Research 2022*.

[55] T. Blockwitz *et al.*, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," in *9th International Modelica Conference*, 2012.

[56] K. Kogiso, "Attack detection and prevention for encrypted control systems by application of switching-key management," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 5032–5037.

[57] Y. Chen and P. Q. Nguyen, "Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers," in *Advances in Cryptology – EUROCRYPT 2012*, D. Pointcheval and T. Johansson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 502–519, ISBN: 978-3-642-29011-4.

[58] H. Cohn and N. Heninger, "Approximate common divisors via lattices," *The Open Book Series*, vol. 1, no. 1, pp. 271–293, Nov. 14, 2013.

[59] M. D. Gupta and R. K. Chauhan, "Recent development of hardware-based random number generators on fpga for cryptography," in *VLSI, Microwave and Wireless Technologies*, B. Mishra and M. Tiwari, Eds., Singapore: Springer Nature Singapore, 2023, pp. 489–500, ISBN: 978-981-19-0312-0.

[60] H. B. Meitei and M. Kumar, "Fpga implementation of true random number generator architecture using all digital phase-locked loop," *IETE Journal of Research*, vol. 68, no. 3, pp. 1561–1570, 2022.

[61] J. Rajput and A. Bajpai. "Study on Deterministic and Probabilistic Computation of Primality Test." (Feb. 24, 2019), (visited on 09/01/2023), preprint.

[62] A. K. Tarafder and T. Chakroborty, "A comparative analysis of general, sieve-of-eratosthenes and rabin-miller approach for prime number generation," in *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, 2019, pp. 1–4.

[63] E. Bach, "Explicit bounds for primality testing and related problems," *Math. Comp.*, vol. 55, no. 191, pp. 355–380, 1990.

[64] U. Daepp and P. Gorkin, "Fermat's little theorem," in *Reading, Writing, and Proving: A Closer Look at Mathematics*. New York, NY: Springer New York, 2011, pp. 315–323, ISBN: 978-1-4419-9479-0.

[65] R. Cheung, A. Brown, W. Luk, and P. Cheung, "A scalable hardware architecture for prime number validation," in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, 2004, pp. 177–184.

[66] J. Sorenson and J. Webster, "Strong pseudoprimes to twelve prime bases," *Mathematics of Computation*, vol. 86, no. 304, pp. 985–1003, Jun. 2016.

[67] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Optimised multiplication architectures for accelerating fully homomorphic encryption," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2794–2806, 2016.

[68] J. Dyer, M. Dyer, and J. Xu, "Practical homomorphic encryption over the integers for secure computation in the cloud," *International Journal of Information Security*, vol. 18, pp. 549–579, 2019.