Documentation on Encrypted Dynamic Control Simulation Code using Ring-LWE based Cryptosystems

Yeongjun Jang* · Joowon Lee* · Junsoo Kim**

*ASRI, the Department of Electrical and Computer Engineering, Seoul National University, South Korea
**the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, South Korea
*E-mail: junsookim@seoultech.ac.kr

Keywords: Encrypted Control, Ring-LWE (Ring-Learning With Error), Implementation
JL 002/02/4202–0086

1. Introduction

Networked control is to outsource computations of physical devices to cloud servers in order to achieve computational efficiency and higher performance. However, it also introduces the risk of cyber-attacks, as data that are transmitted and processed via public network are vulnerable to eavesdropping and manipulation by unauthorized third parties. Encrypted control addresses this issue by employing homomorphic cryptosystems to perform control computations over encrypted data without decryption [1–3]. This ensures the confidentiality of all data throughout the transmission and computation stages.

While significant theoretical advancements have been made in encrypted control, applying it to real-time systems remains challenging, primarily due to the high computational complexity of cryptosystems. To ensure that all computations are completed within the sampling period, results found in the literature are typically limited to simple systems or compromise the level of security as a trade-off for reducing the computational burden, as in [4, 5].

This issue could be relieved by utilizing some library implementations, such as Microsoft SEAL [6], HElib [7], and Lattigo [8]; they support optimized implementations of Ring-Learning With Errors (LWE) based cryptosystems [9–12], which are among the most efficient and advanced homomorphic cryptosystem available. However, such libraries often lack instructions for those without a strong cryptographic background, and thus, have not been actively used in the field of encrypted control.

This paper aims to provide an accessible guideline for implementing Ring-LWE based encrypted controllers using the Lattigo library, which supports an efficient implementation of state-of-the-art cryptosystems. In particular, we focus on the implementations of the encrypted controllers presented in [13] and [14]. We briefly review each result, and provide some example codes, fully available at: `https://github.com/CDSL-EncryptedControl/CDSL`, to assist our explanations.

The remainder of this paper is organized as follows. Section 2 briefly introduces Ring-LWE based cryptosystems. Sections 3 and 4 provide code-based explanations on implementing the encrypted controllers presented in [13] and [14], respectively. Section 5 concludes the paper.

Notation: The set of integers, nonnegative integers, positive integers, and real numbers are denoted by $\mathbb{Z}$, $\mathbb{Z}_{\geq 0}$, $\mathbb{N}$, and $\mathbb{R}$, respectively. The Kronecker product and the Hadmard product (element-wise multiplication of vectors) are written by $\otimes$ and $\circ$, respectively. The Moore-Penrose inverse and the vectorization of a matrix A are denoted by $A^\dagger$ and $\mathrm{vec}(A)$, respectively. For scalars $a_1, \ldots, a_n$, we define $\mathrm{col}\{a_i\}_{i=1}^n := [a_1, \ldots, a_n]^\top$. The vector $\mathbf{1}_n$ is defined as $[1, \ldots, 1]^\top \in \mathbb{R}^n$.

2. Ring-LWE based Cryptosystem

This section provides a brief overview of Ring-LWE based cryptosystems [9–12, 15]. Ring-LWE based cryptosystems are based on the polynomial ring $R_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$, which is defined by parameters $q \in \mathbb{N}$ and $N \in \mathbb{N}$, where N is a power of two. The elements of R_q can be represented by polynomials with indeterminate X, coefficients in $\mathbb{Z}_q := \mathbb{Z} \cap [-q/2, q/2)$, and degree less than N. The ring R_q is closed under modular addition and multiplication, which regards $X^N \equiv -1$ and map each coefficient to the set $\mathbb{Z}_q$ via the modulo operation de-

fined by $a \bmod q := a - \lfloor (a+q/2)/q \rfloor q$ for all $a \in \mathbb{Z}$. The l_∞ norm of a polynomial $m = \sum_{i=0}^{N-1} m_i X^i \in R_q$ is defined as $\|m\| := \max_{0 \le i < N} |m_i|$.

In what follows, the setup, encryption and decryption algorithms, and homomorphic addition are described.

- *Parameters* (N, q, σ): The degree $N \in \mathbb{N}$ is a power of two and the modulus $q \in \mathbb{N}$ is a prime satisfying $1 = 2N \bmod q$. The parameter σ gives a bound on the error distribution.
- *Key generation*: Generate the secret key $\mathsf{sk} \in R_q$.
- *Encryption*: $\mathsf{Enc} : R_q \to R_q^2$
- *Decryption*: $\mathsf{Dec} : R_q^2 \to R_q$
- *Homomorphic addition*: $\oplus : R_q^2 \times R_q^2 \to R_q^2$

The encryption and decryption algorithms satisfy the *correctness* property, i.e., for any $m \in R_q$,

$$\mathsf{Dec}(\mathsf{Enc}(m)) = m + e \mod q, \tag{1}$$

where $e \in R_q$ is the error polynomial injected during the encryption of the message m bounded by $\|e\| \le \sigma$. The scheme also satisfies the *additively homomorphic* property, written as

$$\mathsf{Dec}(\mathbf{c}_1 \oplus \mathbf{c}_2) = \mathsf{Dec}(\mathbf{c}_1) + \mathsf{Dec}(\mathbf{c}_2) \mod q,$$

for any $\mathbf{c}_1 \in R_q^2$ and $\mathbf{c}_2 \in R_q^2$. The *multiplicatively homomorphic* property will be described in Section 4.

In [13], a method called "external product" [15] has been used to perform multiplications over encrypted data. The external product is a product between a *ciphertext* (encrypted data) in R_q^2 and another type of ciphertext generated by the Ring-GSW (Gentry-Sahai-Waters) encryption algorithm [15], which is described below.

- *Parameters* (d, ν, P): Choose $d \in \mathbb{N}$ and a power of two $\nu \in \mathbb{N}$ such that $\nu^{d-1} < q \le \nu^d$, and the *special modulus* $P \in \mathbb{N}$ as a prime.
- *Ring-GSW encryption*: $\mathsf{Enc}' : R_q \to R_{qP}^{2\times 2d}$
- *External product*: $\boxdot$: $R_{qP}^{2\times 2d} \times R_q^2 \to R_q^2$

For any $m \in R_q$ and $\mathbf{c} \in R_q^2$, it holds that

$$\mathsf{Dec}(\mathsf{Enc}'(m) \boxdot \mathbf{c}) = m \cdot \mathsf{Dec}(\mathbf{c}) + e \mod q, \tag{2}$$

where $e \in R_p$ is a "newborn" error under the external product bounded by $\|e\| \le \sigma_{\mathsf{Mult}} := P^{-1} d N \sigma \nu + (N+1)/2$. The property (2) enables the recursive multiplications on the encrypted state of the controller proposed in [13]. The special modulus P is used to temporarily "lift" the modulus from q to qP during the execution of the external product, thus reducing the bound σ_{Mult}.

For simplicity, we abuse notation and define $\mathsf{Enc}(\cdot)$, $\mathsf{Dec}(\cdot)$, and $\|\cdot\|$ component-wisely for vectors throughout the paper.

3. Recursive Encrypted Controller

In this section, we illustrate the implementation of the encrypted controller proposed in [13], focusing on its application to a four-tank system [16], using Lattigo. The encrypted controller enables an unlimited number of recursive homomorphic multiplications (by an encrypted integer matrix) without bootstrapping, which typically incurs high computational cost. The computation speed can be further accelerated by applying a "coefficient-packing" technique that encodes a vector into a polynomial. Example codes and instructions are included to assist readers in applying Lattigo for different systems or approaches as they choose, which are fully available at: `https://github.com/CDSL-EncryptedControl/CDSL/tree/main/ctrRGSW`

The plant is a four-tank system [16] discretized using the sampling time 100 ms, written as

$$\begin{aligned} x_p(t+1) &= Ax_p(t) + Bu(t), \quad x_p(0) = x_p^{\mathsf{ini}}, \\ y(t) &= Cx_p(t), \end{aligned} \tag{3}$$

where

$$A = \begin{bmatrix} 0.9984 & 0 & 0.0042 & 0 \\ 0 & 0.9989 & 0 & -0.0033 \\ 0 & 0 & 0.9958 & 0 \\ 0 & 0 & 0 & 0.9967 \end{bmatrix}, B = \begin{bmatrix} 0.0083 & 0 \\ 0 & 0.0063 \\ 0 & 0.0048 \\ 0.0031 & 0 \end{bmatrix},$$
$$C = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix}.$$

With $n = 4$, $m = 2$, and $l = 2$, $x_p(t) \in \mathbb{R}^n$ is the state with the initial value $x_p^{\mathsf{ini}} \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$ is the input, and $y(t) \in \mathbb{R}^l$ is the output.

An observer-based controller that stabilizes (3) is designed as

$$\begin{aligned} x(t+1) &= Fx(t) + Gy(t), \quad x(0) = x^{\mathsf{ini}}, \\ u(t) &= Hx(t), \end{aligned} \tag{4}$$

where $x(t) \in \mathbb{R}^n$ is the state with the initial value $x^{\mathsf{ini}} \in \mathbb{R}^n$, and the matrices are given by $F = A + BK - LC$, $G = L$, and $H = K$ with

$$K = \begin{bmatrix} -0.7905 & 0.1579 & -0.2745 & -0.2686 \\ -0.1552 & -0.7874 & -0.3427 & 0.3137 \end{bmatrix},$$
$$L = \begin{bmatrix} 0.7815 & 0 & 0.3190 & 0 \\ 0 & 0.7816 & 0 & -0.3199 \end{bmatrix}^\top.$$

It can be easily verified that (4) is controllable and observable.

The objective is to design an encrypted controller that performs the operations of (4) over encrypted data exploiting the homomorphic properties described in Section 2. The design should guarantee that the performance of the encrypted controller can be made arbitrarily close to that of the unencrypted controller (4) with an appropriate choice of parameters.

3.1 Encryption setting

In Lattigo, one way to setup the parameters of Ring-LWE based cryptosystems is as follows:

```
params, _ := rlwe.NewParametersFromLiteral(
rlwe.ParametersLiteral{
        LogN: 13,           % polynomial degree
        LogQ: []int{56},    % ciphertext modulus q
        LogP: []int{51}})   % special modulus P
```

The above code sets $N = 2^{13}$, and automatically selects primes $q \approx 2^{56}$ and $P \approx 2^{51}$ meeting the requirement $1 = 2N \bmod q$, which are $q = 72057594037616641$ and $P = 2251799813554177$ in this example. The chosen parameters ensure the 128-bit security according to [17]. By default, the error distribution is defined as a discrete zero-mean Gaussian distribution with standard deviation 3.2 and bound $\sigma = 19.2$, and the parameters for the Ring-GSW encryption are set as $\nu = q$ and $d = 1$.

Given the parameters, we generate the secret key using

```
kgen := rlwe.NewKeyGenerator(params)
sk := kgen.GenSecretKeyNew() % secret key sk ∈ R_q
```

where each coefficient of sk is sampled uniformly at random from the set $\{-1, 0, 1\}$ by default.

Next, we briefly describe the packing technique proposed in [13]. Let us define $\tau \in \mathbb{N}$ as the smallest power of two satisfying $\tau \geq \max\{n, m, l\}$. For any $k \in \mathbb{N}$ such that $k \leq \tau$, the packing algorithm $\mathsf{Pack}^k_{\mathsf{coeff}} : \mathbb{Z}_q^k \to R_q$ encodes each component of a vector as coefficients of the terms $X^0, X^{N/\tau}, \ldots, X^{(k-1)N/\tau}$ in sequential order, and the corresponding unpacking algorithm $\mathsf{UnpackPt}^k_{\mathsf{coeff}} : R_q \to \mathbb{Z}_q^k$ reconstructs the original vector, i.e.,

$$\mathsf{UnpackPt}^k_{\mathsf{coeff}}(\mathsf{Pack}^k_{\mathsf{coeff}}(a)) = a, \quad \forall a \in \mathbb{Z}_q^k.$$

In fact, an algorithm $\mathsf{UnpackCt}^k_{\mathsf{coeff}} : R_q^2 \to R_q^{2k}$ that homomorphically evaluates $\mathsf{UnpackPt}^k_{\mathsf{coeff}}$ is proposed in [13], which satisfies

$$\mathsf{UnpackPt}^k_{\mathsf{coeff}}(\mathsf{Dec}(\mathbf{c})) = \mathsf{Dec}(\mathsf{UnpackCt}^k_{\mathsf{coeff}}(\mathbf{c})) + e$$

for any $\mathbf{c} \in R_q^2$, with some $e \in R_q^k$ bounded by $\|e\| \leq \sigma_{\mathsf{Mult}} \log_2 \tau$. Notably, this allows efficient matrix-vector multiplications over encrypted data; see [13, Section IV] for more details.

To make use of the unpacking algorithm $\mathsf{UnpackCt}^k_{\mathsf{coeff}}$, we need to define some additional *evaluation keys* as follows:

```
rlk := kgen.GenRelinearizationKeyNew(sk)
evkRGSW := rlwe.NewMemEvaluationKeySet(rlk)
evkRLWE := rlwe.NewMemEvaluationKeySet(rlk,
        kgen.GenGaloisKeysNew(galEls, sk)...)
```

3.2 Encrypted controller design [13]

It is known that the state matrix of the controller needs to consist of integers for a dynamic system to operate on encrypted data [18]. The code `ctrRGSW/conversion.m` converts the state matrix F of (4) into integers based on the approach of [4], which "re-encrypts" the output: we reformulate the state dynamics of (4), as

$$x(t+1) = (F - RH)x(t) + Gy(t) + Ru(t),$$

where $R \in \mathbb{R}^{n \times m}$ is designed as

$$R = \begin{bmatrix} -1.6879 & 0.4148 & 0.2880 & -0.7385 \\ -0.6892 & -2.1054 & 4.1931 & -2.0807 \end{bmatrix}^\top,$$

so that $\bar{F} := (F - RH) \in \mathbb{Z}^{n \times n}$. We now consider $\bar{F}$ as the state matrix of (4) regarding $u(t)$ as a fed-back input.

In `ctrRGSW/packing/main.go`, each column of the control parameters $\{F, G, H, R\}$ is quantized using a scale factor $1/\mathsf{s} \in \mathbb{N}$, packed, and then encrypted, as

$$\begin{aligned}
\mathbf{F}_i &:= \mathsf{Enc}'(\mathsf{Pack}^n_{\mathsf{coeff}}(F_i \bmod q)), i = 0, \ldots, n-1, \\
\mathbf{G}_i &:= \mathsf{Enc}'(\mathsf{Pack}^n_{\mathsf{coeff}}(\lceil G_i/\mathsf{s} \rfloor \bmod q)), i = 0, \ldots, l-1, \\
\mathbf{H}_i &:= \mathsf{Enc}'(\mathsf{Pack}^m_{\mathsf{coeff}}(\lceil H_i/\mathsf{s} \rfloor \bmod q)), i = 0, \ldots, n-1, \\
\mathbf{R}_i &:= \mathsf{Enc}'(\mathsf{Pack}^n_{\mathsf{coeff}}(\lceil R_i/\mathsf{s} \rfloor \bmod q)), i = 0, \ldots, m-1,
\end{aligned}$$

where $\{F_i, G_i, H_i, R_i\}$ denote the $(i+1)$-th column of $\{F, G, H, R\}$. This can be realized in the case of G as

```
GBar := utils.ScalMatMult(1/s, G) % scale up
ctG := RGSW.EncPack(Gbar, tau, encryptorRGSW,
  levelQ, levelP, ringQ, params) % pack and
      encrypt
```

Here, the variable `ctG` represents a l-dimensional vector of Ring-GSW type ciphertexts $\mathbf{G}_i$'s.

Similarly, the initial state x^{ini} is quantized, packed, and encrypted as

$$\mathbf{x}^{\mathsf{ini}} := \mathsf{Enc}(\mathsf{Pack}^n_{\mathsf{coeff}}(\lceil x^{\mathsf{ini}}/(\mathsf{rs}) \rfloor / \mathsf{L} \bmod q)),$$

which is realized via

```
xBar := utils.ScalVecMult(1/(r*s),x_ini)
xCtPack := RLWE.EncPack(xScale, tau, 1/L,
        *encryptorRLWE, ringQ, params)
```

The parameter $\mathsf{r} > 0$ represents the quantization step size of the sensor, and the additional scale factor $1/\mathsf{L} \in \mathbb{N}$ is introduced to reduce the effect of errors that can be found in (1) and (2), for example. The parameters $\{\mathsf{r}, \mathsf{L}, \mathsf{s}\}$ are set as

```
s := 1/10000.0    L := 1/10000.0    r := 1/10000.0
```

At each time step $t \in \mathbb{Z}_{\geq 0}$ of the online control procedure, the plant output $y(t)$ is encrypted at the sensor, as

$$\mathbf{y}(t) := \mathsf{Enc}(\mathsf{Pack}^{l}_{\mathsf{coeff}}(\lceil y(t)/\mathsf{r} \rfloor / \mathsf{L} \mod q)).$$

Then, it is sent to the encrypted controller, written by

$$\begin{aligned}\mathbf{x}(t+1) = &\left(\sum_{i=0}^{n-1} \mathbf{F}_i \boxdot \mathbf{x}_i(t)\right) \oplus \left(\sum_{i=0}^{l-1} \mathbf{G}_i \boxdot \mathbf{y}_i(t)\right) \\ &\oplus \left(\sum_{i=0}^{m-1} \mathbf{R}_i \boxdot \mathbf{u}_i(t)\right), \qquad (5)\\ \mathbf{u}(t) = &\sum_{i=0}^{n-1} \mathbf{H}_i \boxdot \mathbf{x}_i(t), \qquad \mathbf{x}(0) = \mathbf{x}^{\mathsf{ini}},\end{aligned}$$

where we let the output $\mathbf{u}(t)$ be decrypted at the actuator to obtain the plant input $u(t)$ and the re-encrypted fed-back inputs $\mathbf{u}_i(t)$'s, as

$$\begin{aligned}&u(t) = \mathsf{r}^2\mathsf{L} \cdot \mathsf{UnpackPt}^{m}_{\mathsf{coeff}}(\mathsf{Dec}(\mathbf{u}(t))),\\ &[\mathbf{u}_0(t); \cdots; \mathbf{u}_{m-1}(t)] := \mathsf{Enc}(\lceil u(t)/\mathsf{r} \rfloor / \mathsf{L} \mod q),\end{aligned}$$

and $\mathbf{x}_i(t)$'s and $\mathbf{y}_i(t)$'s are obtained through unpacking:

$$\begin{aligned}[\mathbf{x}_0(t); \cdots; \mathbf{x}_{n-1}(t)] &:= \mathsf{UnpackCt}^{n}_{\mathsf{coeff}}(\mathbf{x}(t)),\\ [\mathbf{y}_0(t); \cdots; \mathbf{y}_{l-1}(t)] &:= \mathsf{UnpackCt}^{l}_{\mathsf{coeff}}(\mathbf{y}(t)).\end{aligned}$$

The overall procedure at each time step can be simulated using the following code:

```
y := utils.MatVecMult(C, xp)     % @ plant
yBar := utils.RoundVec(utils.ScalVecMult(1/r,y))
                                 % @ sensor
yCtPack := RLWE.EncPack(yBar, tau, 1/L,
 *encryptorRLWE, ringQ, params) % @ sensor
xCt := RLWE.UnpackCt(xCtPack, n, tau,
    evaluatorRLWE,
  ringQ, monomials, params)      % @ controller
yCt := RLWE.UnpackCt(yCtPack, p, tau,
    evaluatorRLWE,
  ringQ, monomials, params)      % @ controller
uCtPack := RGSW.MultPack(xCt, ctH, evaluatorRGSW,
        ringQ, params)           % @ controller
u := RLWE.DecUnpack(uCtPack, m, tau, *
    decryptorRLWE,
        r*s*s*L, ringQ, params) % @ actuator
uBar := utils.RoundVec(utils.ScalVecMult(1/r, u))
                                 % @ actuator
uReEnc := RLWE.Enc(uBar, 1/L, *encryptorRLWE,
        ringQ, params)           % @ actuator
FxCt := RGSW.MultPack(xCt, ctF, evaluatorRGSW,
        ringQ, params)           % @ controller
GyCt := RGSW.MultPack(yCt, ctG, evaluatorRGSW,
        ringQ, params)           % @ controller
RuCt := RGSW.MultPack(uReEnc, ctR, evaluatorRGSW,
        ringQ, params)           % @ controller
xCtPack = RLWE.Add(FxCt, GyCt, RuCt, params)
                                 % @ controller
xp = utils.VecAdd(utils.MatVecMult(A, xp),
        utils.MatVecMult(B, u)) % @ plant
```

We define the performance error as the difference between the plant input generated by the unencrypted controller (4) and the encrypted controller (5). In this example, the maximum and average of the performance error for 1000 iterations are obtained as 0.0089 and 0.0028, respectively. Also, it took 16.6 ms on average to compute the control input at each time step, which falls within the sampling time. The encrypted controller implemented without packing can be found in the code `ctrRGSW/noPacking/main.go`, which exhibits a slower operation.

Remark 1. To decrease the performance error, the parameters $\{\mathsf{r}, \mathsf{L}, \mathsf{s}\}$ can be decreased. However, this may lead to overflow, where the encrypted messages grow out of $\mathbb{Z}_q$, resulting in controller malfunctions. The modulus q can be increased to avoid such overflow, but greater q often leads to greater N under a fixed level of security [17], which significantly affects the computation time. Another way to decrease the performance error is to increase P, but again, this may require larger N.

4. Non-recursive Encrypted Controller

This section presents an alternative method [14] to encrypt a dynamic controller using Ring-LWE based cryptosystems, provided with a code implementation based on Lattigo. This method offers a faster computation and can be applied to various Ring-LWE based schemes such as BGV [11], BFV [10], and CKKS [12]. However, unlike the method of Section 3, it avoids recursive homomorphic multiplications, and therefore inherently requires re-encryption of the controller output.

The implementation is again based on the four-tank system, where the plant and the controller are the same as (3) and (4), respectively. In this example, the BGV scheme [11] is utilized, but the code can be easily modified for the use of other Ring-LWE based schemes. On average, it takes less than 12 ms for the resulting encrypted controller to compute the control input at each time step. Readers can refer to the entire set of codes at: `https://github.com/CDSL-EncryptedControl/CDSL/tree/main/ctrRLWE`

4.1 Encryption setting

To begin with, we introduce another parameter for encryption, the *plaintext* (message to be encrypted) modulus $p \in \mathbb{N}$, which is a prime satisfying $p = 1 \mod 2N$. In the BGV scheme, the plaintext space is R_p, and the ciphertext space is R_q^2, where q is referred as the ciphertext modulus. Therefore, the encryption and decryption algorithms are defined as $\mathsf{Enc} : R_p \to R_q^2$ and $\mathsf{Dec} : R_q^2 \to R_p$, respectively, throughout this section. Under the assumption that $p \ll q$, the correctness property is rewritten as

$$\mathsf{Dec}(\mathsf{Enc}(m)) = m \mod p, \quad \forall m \in \mathbb{Z}_p.$$

The method of [14] utilizes homomorphic multiplication $\odot : R_q^2 \times R_q^2 \to R_q^3$, which increases the dimension of ciphertexts by one. Accordingly, the homomorphic addition $\oplus$ and decryption $\mathsf{Dec}(\cdot)$ are also defined over R_q^3. The multiplicatively homomorphic property holds, i.e.,

$$\mathsf{Dec}(\mathsf{Enc}(m_1) \odot \mathsf{Enc}(m_2)) = m_1 \cdot m_2 \mod p,$$

for any $m_1 \in R_p$ and $m_2 \in R_p$.

In this section, we exploit the packing method based on the number-theoretic transform (NTT) [19], which transforms a vector in $\mathbb{Z}_p^N$ into a polynomial in R_p and vice versa. Let the NTT-style packing and unpacking functions be denoted by $\mathsf{Pack}_{\mathsf{ntt}} : \mathbb{Z}_p^N \to R_p$ and $\mathsf{Unpack}_{\mathsf{ntt}} : R_p \to \mathbb{Z}_p^N$, respectively. Then, the followings hold for any $a \in \mathbb{Z}_p^N$ and $b \in \mathbb{Z}_p^N$:

$$\begin{aligned} \mathsf{Unpack}_{\mathsf{ntt}}(\mathsf{Pack}_{\mathsf{ntt}}(a)) &= a \mod p, \\ \mathsf{Unpack}_{\mathsf{ntt}}(\mathsf{Pack}_{\mathsf{ntt}}(a) + \mathsf{Pack}_{\mathsf{ntt}}(b)) &= a + b \mod p, \\ \mathsf{Unpack}_{\mathsf{ntt}}(\mathsf{Pack}_{\mathsf{ntt}}(a) \cdot \mathsf{Pack}_{\mathsf{ntt}}(b)) &= a \circ b \mod p. \end{aligned} \tag{6}$$

Thus, the element-wise additions and multiplications among vectors can be operated at once over encrypted data. We abuse notation and apply $\mathsf{Pack}_{\mathsf{ntt}}$ to vectors with length shorter than N, regarding that zeros are "padded" at the end to fit the length.

In `ctrRLWE/main.go`, the following code assigns the encryption parameters to be $N = 2^{12}$, $p \approx 2^{26}$, and $q \approx 2^{74}$:

```
logN:=12     ptSize:=uint64(28)   ctSize:=int(74)
```

The above parameters are chosen to meet the 128-bit security, based on [17]. The code automatically finds a suitable prime p that satisfies $p = 1 \mod 2N$, saving the exact value of p as `ptModulus`, which is $p = 268460033$ in this case.

To represent large integers in $\mathbb{Z}_q$, where $q \approx 2^{74}$, we create a chain of ciphertext moduli $\mathbb{Z}_{q'} \times \mathbb{Z}_{q'}$, where $q' \approx 2^{37}$. Thus, the code assigns [37, 37] to `logQ`, from which Lattigo finds a proper prime q' by running the following code:

```
params, _ := bgv.NewParametersFromLiteral(bgv.
    ParametersLiteral{
        LogN:             logN,
        LogQ:             logQ,
        PlaintextModulus: ptModulus,})
```

The NTT-style packing operation is enabled in Lattigo by generating a new "encoder" as follows:

```
encoder := bgv.NewEncoder(params)
```

Meanwhile, the method of [14] only utilizes basic homomorphic addition and multiplication. Therefore, the homomorphic evaluator is constructed simply as follows:

```
eval := bgv.NewEvaluator(params, nil)
```

4.2 Encrypted controller design [14]

The method of [14] first converts the given controller (4) into an equivalent form;

$$u(t) = \sum_{i=1}^{n} H_{u,i} u(t-i) + H_{y,i} y(t-i), \tag{7}$$

where $H_{u,i} \in \mathbb{R}^{m \times m}$ and $H_{y,i} \in \mathbb{R}^{m \times l}$ for $i = 1, \ldots, n$. The matrices in (7) are obtained as

$$\begin{aligned} [H_{u,n}, \ldots, H_{u,1}] &= HF^n\mathcal{O}^\dagger, \\ [H_{y,n}, \ldots, H_{y,1}] &= H\left(\mathcal{C} - F^n\mathcal{O}^\dagger\mathcal{T}\right), \end{aligned}$$

where

$$\mathcal{O} = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n-1} \end{bmatrix}, \mathcal{T} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ HG & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ HF^{n-2}G & \cdots & HG & 0 \end{bmatrix},$$
$$\mathcal{C} = \begin{bmatrix} F^{n-1}G & \cdots & FG & G \end{bmatrix},$$

with 0 denoting zero matrices of appropriate dimensions. Furthermore, to compute $u(t)$ from (7) for $t = 0, \ldots, n-1$, we obtain the following thanks to [14, Lemma 1]:

$$\begin{aligned}\left[y(-n)^\top, \ldots, y(-1)^\top\right]^\top &= \mathcal{C}^\dagger x^{\mathsf{ini}},\\ \left[u(-n)^\top, \ldots, u(-1)^\top\right]^\top &= \mathcal{T}\mathcal{C}^\dagger x^{\mathsf{ini}}.\end{aligned} \tag{8}$$

This is implemented in `ctrRLWE/conversion.m`.

To make use of the properties (6), the matrix-vector multiplications in (7) are reformulated, using the fact that

$$Ab = \mathrm{col}\left\{\left\langle \mathrm{vec}(A^\top) \circ (\mathbf{1}_m \otimes b), e_i \otimes \mathbf{1}_h \right\rangle\right\}_{i=1}^{m} \tag{9}$$

for any $A \in \mathbb{R}^{m\times h}$ and $b \in \mathbb{R}^h$, where $e_i \in \mathbb{R}^m$ is the unit vector whose only nonzero element is its i-th element.

The matrices in (7) are vectorized, quantized, packed, and then encrypted as

$$\begin{aligned}\mathbf{H}_{u,i} &:= \mathsf{Enc}(\mathsf{Pack}_{\mathsf{ntt}}(\left\lceil \mathrm{vec}(H_{u,i}^\top)/\mathsf{s} \right\rfloor \mod p)),\\ \mathbf{H}_{y,i} &:= \mathsf{Enc}(\mathsf{Pack}_{\mathsf{ntt}}(\left\lceil \mathrm{vec}(H_{y,i}^\top)/\mathsf{s} \right\rfloor \mod p)),\end{aligned}$$

for $i = 1, \ldots, n$. In `ctrRLWE/main.go`, $\mathbf{H}_{u,n-j}$ and $\mathbf{H}_{y,n-j}$ correspond to `ctHu[j]` and `ctHy[j]`, respectively, for $j = 0, \ldots, n-1$.

The encrypted controller is designed as

$$\bar{\mathbf{u}}(t) = \sum_{i=1}^{n} \mathbf{H}_{u,i} \odot \mathbf{u}(t-i) \oplus \mathbf{H}_{y,i} \odot \mathbf{y}(t-i), \tag{10}$$

where $\mathbf{y}(t)$ and $\mathbf{u}(t)$ are inputs from the sensor and the actuator, respectively, defined as

$$\begin{aligned}\mathbf{y}(t) &= \mathsf{Enc}(\mathsf{Pack}_{\mathsf{ntt}}(\lceil \mathbf{1}_m \otimes y(t)/\mathsf{r} \rfloor \mod p)),\\ \mathbf{u}(t) &= \mathsf{Enc}(\mathsf{Pack}_{\mathsf{ntt}}(\lceil \mathbf{1}_m \otimes u(t)/\mathsf{r} \rfloor \mod p)).\end{aligned} \tag{11}$$

The initial input-output trajectories (8) are encrypted as in (11) during offline. At the actuator, the plant input is computed as

$$u(t) = \mathrm{col}\left\{\langle \mathsf{rs} \cdot \mathsf{Unpack}_{\mathsf{ntt}}(\mathsf{Dec}(\bar{\mathbf{u}}(t))), e_i \otimes \mathbf{1}_h \rangle\right\}_{i=1}^{m}. \tag{12}$$

In fact, this design requires that all of $\mathbf{1}_m \otimes y(t)$, $\mathbf{1}_m \otimes u(t)$, $\mathrm{vec}(H_{u,i}^\top)$'s, and $\mathrm{vec}(H_{y,i}^\top)$'s have the same length. Thus, zeros are padded to ensure that they are all of length mh, where $h := \max\{m, l\}$. The function `utils.VecDuplicate` is utilized to generate $\mathbf{1}_m \otimes y(t)$ and $\mathbf{1}_m \otimes u(t)$, with zero padding if necessary. The vectorized control parameters with zero padding are precomputed in `ctrRLWE/conversion.m`.

The Go code `ctrRLWE/main.go` implements the encrypted controller (10) as follows:

```
Uout, _ := eval.MulNew(ctHy[0], ctY[0])
eval.MulThenAdd(ctHu[0], ctU[0], Uout)
for j := 1; j < n; j++ {
    eval.MulThenAdd(ctHy[j], ctY[j], Uout)
    eval.MulThenAdd(ctHu[j], ctU[j], Uout)  }
```

Here, the function `eval.MulThenAdd` homomorphically adds the first and the second arguments, assigning the outcome to the third argument. The variables `ctY[j]` and `ctU[j]` correspond to $\mathbf{y}(t-n+j)$ and $\mathbf{u}(t-n+j)$, respectively, for $j = 0, \ldots, n-1$.

The inputs (11) of the encrypted controller are generated as

```
Ysens:=utils.ModVecFloat(utils.RoundVec(utils.
    ScalVecMult(1/r, utils.VecDuplicate(Y, m, h))
    ), params.PlaintextModulus())
Ypacked:=bgv.NewPlaintext(params, params.MaxLevel
    ())
encoder.Encode(Ysens, Ypacked)
Ycin, _:=encryptor.EncryptNew(Ypacked)
```

and

```
Upacked:=bgv.NewPlaintext(params, params.MaxLevel
    ())
encoder.Encode(utils.ModVecFloat(utils.RoundVec(
    utils.ScalVecMult(1/r, utils.VecDuplicate(U,
    m, h))), params.PlaintextModulus()), Upacked)
Ucin, _:=encryptor.EncryptNew(Upacked)
```

where `Y` and `U` refer to the plant output and input, respectively.

Finally, the operations at the actuator (12) are realized as follows:

```
Uact := make([]uint64, params.N())
Usum := make([]uint64, m)
encoder.Decode(decryptor.DecryptNew(Uout), Uact)
for k := 0; k < m; k++ {
    Usum[k] = utils.VecSumUint(Uact[k*h:(k+1)*h],
         params.PlaintextModulus(), bredparams)
    U[k] = float64(r * s * utils.SignFloat(
        float64(Usum[k]), params.PlaintextModulus
        ()))    }
```

The variable `bredparams` is for addition over $\mathbb{Z}_p$ in Lattigo.

In this example, the parameters `r` and `s` are set as 0.0002 and 0.0001, respectively. The performance error is 0.00254 on average, and 0.015788 at maximum. As mentioned in Remark 1, the parameters `r` and `s` can be decreased for better performance, however, one should be aware that the plaintext modulus p

needs to satisfy $\|u(t)\|_\infty/(\mathsf{rs}) < p/2$, in order to avoid overflow.

5. Conclusion

This paper provides example codes with instructions on implementing Ring-LWE based encrypted controllers [13, 14] for a four-tank system using Lattigo. The codes can be easily customized by readers for application to different systems or approaches. It is demonstrated that the methods of [13] and [14] enable fast implementations of encrypted controllers, taking less than 17 ms on average at each time step for computing the control input of a fourth-order controller.

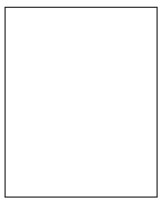
(2025.1.14 Received)

References

1) K. Kogiso and T. Fujita, "Cyber-security enhancement of networked control systems using homomorphic encryption," in *2015 54th IEEE Conference on Decision and Control*, 2015, pp. 6836–6843.
2) 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.
3) J. Kim, D. Kim, Y. Song, H. Shim, H. Sandberg, and K. H. Johansson, "Comparison of encrypted control approaches and tutorial on dynamic systems using learning with errors-based homomorphic encryption," *Annual Reviews in Control*, vol. 54, pp. 200–218, 2022.
4) J. Kim, H. Shim, and K. Han, "Dynamic controller that operates over homomorphically encrypted data for infinite time horizon," *IEEE Transactions on Automatic Control*, vol. 68, no. 2, pp. 660–672, 2023.
5) N. Schlüter, J. Kim, and M. S. Darup, "A code-driven tutorial on encrypted control: From pioneering realizations to modern implementations," in *2024 European Control Conference (ECC)*, 2024, pp. 914–920.
6) "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, Microsoft Research, Redmond, WA.
7) S. Halevi and V. Shoup, "Design and implementation of HElib: A homomorphic encryption library," Cryptology ePrint Archive, Paper 2020/1481, 2020. [Online]. Available: https://eprint.iacr.org/2020/1481
8) "Lattigo v6," Aug. 2024, ePFL-LDS, Tune Insight SA. [Online]. Available: https://github.com/tuneinsight/lattigo
9) V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of the ACM*, vol. 60, no. 6, 2013, Art. no. 43.
10) J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144
11) Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory*, vol. 6, no. 3, 2014, Art. no. 13.
12) J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2017, pp. 409–437.
13) Y. Jang, J. Lee, S. Min, H. Kwak, J. Kim, and Y. Song, "Ring-LWE based encrypted controller with unlimited number of recursive multiplications and effect of error growth," 2024, arXiv:2406.14372.
14) J. Lee, D. Lee, J. Kim, and H. Shim, "Encrypted dynamic control exploiting limited number of multiplications and a method using rlwe-based cryptosystem," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 55, no. 1, pp. 158–169, 2025.
15) I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology – ASIACRYPT 2016*, J. H. Cheon and T. Takagi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 3–33.
16) K. Johansson, "The quadruple-tank process: A multivariable laboratory process with an adjustable zero," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 3, pp. 456–465, 2000.
17) M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption standard," in *Protecting Privacy through Homomorphic Encryption*, K. Lauter, W. Dai, and K. Laine, Eds. Cham: Springer, 2021, pp. 31–62.
18) 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 57th IEEE Conference on Decision and Control*, 2018, pp. 5020–5025.
19) R. Agarwal and C. Burrus, "Fast convolution using fermat number transforms with applications to digital filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 22, no. 2, pp. 87–97, 1974.

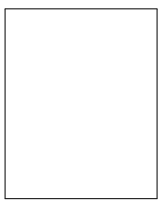
[Biography]

Yeongjun Jang (Non-Member)



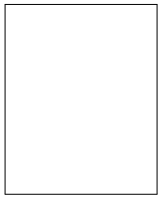
He received the B.S. degree in electrical and computer engineering in 2022, from Seoul National University, South Korea. He is currently a combined M.S./Ph.D. student in electrical and computer engineering at Seoul National University, South Korea. His research interests include data-driven control and encrypted control systems.

Joowon Lee (Non-Member)



She received the B.S. degree in electrical and computer engineering in 2019, from Seoul National University, South Korea. She is currently a combined M.S./Ph.D. student in electrical and computer engineering at Seoul National University, South Korea. Her research interests include data-driven control, encrypted control systems, and cyber-physical systems.

Junsoo Kim (Non-Member)



He received the B.S. degrees in electrical engineering and mathematical sciences in 2014, and the M.S. and Ph.D. degrees in electrical engineering in 2020, from Seoul National University,

South Korea, respectively. He held the Postdoc position at KTH Royal Institute of Technology, Sweden, till 2022. He is currently an Assistant Professor at the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, South Korea. His research interests include security problems in networked control systems and encrypted control systems.