Diese Arbeit wurde vorgelegt am Institut für Regelungstechnik der RWTH Aachen University

(Leveled) Homomorphic Encrypted Model Predictive Control

Hackathon Projekt

von

Shouran Ma, B.Sc.
shouran.ma@rwth-aachen.de
OChicken@github.com

Abstract

RLWE-based homomorphic encryption (HE) is a post-quantum secure advanced cryptographic technique to perform arithmetic operations over ciphertext, which can be used to enhance the security of the cyber-physical system (CPS). In the secure enhanced scheme, the CPS performs arithmetic operations over the encrypted signals without decrypting, so that the signal leakage risk is avoided. In this project, we use the CKKS scheme — one of the HE schemes — to develop an encrypted model predictive controller (MPC). With the CSTR model as a benchmark system, we verified the correctness of the encrypted MPC compared with the unencrypted settings. We also compared the time and RAM consumption of encrypted/unencrypted MPC: despite our simplification, the encrypted controller still consume much more resources than the unencrypted counterpart. We propose several possible optimization directions in the final discussion.

Contents

1 Introduction

1.1 Research Background

1.1.1 Using Homomorphic Encryption in Control Systems

Homomorphic encryption (HE) is known to have abundant applications in cryptography, e.g. *privacy-preserving computation*. In 1978, R.Rivest, L.Adleman, and M.Dertouzos [RAD78] proposed the concept "privacy homomorphism" when investigating the private data bank, which allows an untrusted third party to perform computation, query, etc to the encrypted data while the same time hiding its content. The result is thereafter sent back to the owner and decrypted. This is the ancestor of homomorphic encryption. Concretely, let pt be the plaintext and $\mathsf{ct} = \mathsf{enc}(\mathsf{pt})$ is the ciphertext. The computation/query is the operation $f(\mathsf{ct})$. The homomorphic property states that $f(\mathsf{enc}(\mathsf{pt})) = \mathsf{enc}(f(\mathsf{pt}))$, so after the decryption we yield $f(\mathsf{pt})$.

For many years the proposed cryptosystems are either modular *addition homomorphic*, e.g. Paillier [Pai99] or modular *multiplication homomorphic*, e.g. RSA and ElGamal — they are called *partially-homomorphic* or *semi-homomorphic*. This changed in 2009 when C.Gentry proposed a cryptosystem that supports both types as well as unlimited operations [Gen09a, Gen09b], which is known as *fully-homomorphic encryption (FHE)*. C.Gentry's seminal work generated tremendous excitement and was quickly followed by many works, e.g. [BGV12, FV12, CKKS17]. These constructions are lattice-based cryptosystems, in contrast to the Paillier and ElGamal. Lattice-based cryptosystems are of further advantage: they are post-quantum secure.

The idea of using HE in cyber-physical system (CPS) was originally proposed in [KF15] and generalized in the review [SDAQP21], which got its name *encrypted controller*. As we know, CPSs are ubiquitously used in the industrial control system, which is of high importance to the economy and society. Examples of CPSs are supervisory control and data acquisition (SCADA), cloud service providers, or remote sensors exposed to the public networks, which are untrustable in nature but essential for the system to make correct decisions, e.g. in the scenario of automotive driving. CPSs become more vulnerable in the IoT (Internet of Things) era since we have to decrypt the signal inside the CPS and perform arithmetic operations over plaintext, which increases the signal leakage risk by some backdoor programs. Via the HE techniques, the CPSs are able to evaluate encrypted signals so that the signal leakage risk is avoided. Therefore, consider such an encrypted controller is meaningful.

For a long time, the explorations of using HE to model the encrypted controller are restricted to partial homomorphic encryption. The endeavors include the ElGamal-based control system in K.Kogiso and T.Fujita [KF15] and Paillier-based control system in M.Schulze Darup, I.Shames and F.Faroki's series works [SDRS+18, Dar20]. These schemes are not lattice-based, so thus not post-quantum secure.

Among various lattice-based HE schemes, the CKKS's scheme [CKKS17] is known for its capability of treating floating-point numbers. This treatment cure a huge problem in floating-point homomorphic multiplication, which opens a large variety of application scenarios in secure-enhanced CPS. In this project, we extend our scope to the lattice-based homomorphic encrypted *model predictive controller (MPC)* using CKKS's scheme and *Continuous stirred tank reactor (CSTR)* model as a benchmark system which is verifiable with [RMD17].

1.1.2 Lattice-based Homomorphic Encryption is Post-Quantum Secure

Besides avoiding signal leakage risk, a CPS that performs homomorphically encrypted control should further be post-quantum secure. Lattice-based homomorphic encryption cryptosystems achieve both goals. In recent years, there has been a substantial amount of research on quantum computers - machines that exploit quantum mechanical phenomena to solve mathematical problems that are difficult or intractable for conventional computers, which poses a threat to public-key cryptosystems. The ongoing NIST post-quantum cryptography (PQC) project aims to select cryptographic systems that are secure against both quantum and classical computers and can interoperate with existing communications protocols and networks. Classical cryptosystems with not known exponential quantum speedup are candidates for PQC. Table 1.1 shows their comparison, adapted from [Ste18].

Table 1.1: Comparison of public-key cryptosystems

The underlying hardness problem	Candidates	PQC safe?
Integer factorization	RSA, Paillier, Goldwasser-Micali	No
Diffie-Hellman discrete logarithm	ElGamal, elliptic curves family	No
Code-based	McEliece, Niederreiter	Yes
Lattice-based	NTRU, LWE, RLWE, MLWE, ...	Yes
Isogenies	supersingular elliptic curve isogenies	Yes

Lattice-based cryptosystem is one of the most important candidates of the PQC. It exploits the conjectured hard problems on point lattices in $\mathbb{R}^n$ as the foundation for secure cryptographic systems. Beside resistance to quantum attacks (in contrast with most number-theoretic cryptography), its attractive features also include high asymptotic efficiency and parallelism, security under worst-case intractability assumptions, and solutions to fully homomorphic encryption [Pei16]. On July.05, 2022, the MLWE-based cryptosystem Crystal-Kyber was selected as the reference implementation of the public-key encryption standardization. Its predecessor, NewHope, is a RLWE-based cryptosystem, on which [FV12, CKKS17] are based.

1.2 Contributions

With the CSTR model as a benchmark system, we investigated the correctness and performance of an RLWE-based homomorphic encrypted MPC by comparing it with an unencrypted setting in a closed-loop simulation. To this end, we first develop a series of HE operators that performs CKKS's homomorphic encryption scheme, including HE primitives (addition, multiplication, rotation, conjugation), linear transformations (matrix-vector multiplication, vector sum, index fetching), nonlinear functions (exponential, inverse, square root) and ciphertext comparison — this task is accomplished in GPQHE. At the same time, to determine the essential component to perform HE control, we develop a minimum MPC system that includes a stiff ordinary differential equation solver, quadratic programming solver, optimal control solver, etc — this task is accomplished in the software HECTR. By replacing that component with the HE counterparts, an encrypted MPC is obtained.

Many performance tests are performed before comparing the plain (unencrypted) and encrypted MPC. Due to the fact that the performance and versatility of a HE software are dependent on the parameter settings, we test many parameter combinations and select the one that occupies the least system resources as well as is capable to accomplish the control task as the parameter settings for the encrypted MPC. So that both systems are compared under the fully simplified settings. Both systems yield the same results: their relative difference is of the order $\sim 10^{-12}$.

Accompany with this project, we provide three software: GPQHE, HECTR and Libpmu. All of them are developed in C.

GPQHE[1] is a reference implementation of CKKS homomorphic encryption scheme. It is a free software under the terms of *GNU Lesser General Public License (LGPL)* version 2.1, and its architecture is designed analog to HEAAN, NewHope and Crystal-Kyber. This is the reason for its name: "G" stands for LGPL series license, "PQ" stands for post-quantum, and "HE" stands for homomorphic encryption. Except using Libgcrypt for multi-precision integer management, GPQHE does not rely on any existing HE library. The architecture and design details of GPQHE are explained in Section 3.4 (the polynomial arithmetics), Chapter 4 and Section 6.2.1.

HECTR is a reference implementation of encrypted model predictive control. It is also a software under the terms of LGPL version 2.1 and it uses LAPACK for basic linear algebra subroutines. "CTR" in its name stands for "controller". Most of its function interfaces are designed analog to that of Matlab and Octave and we have verified its correctness.

Libpmu[2] is a performance monitor unit. It is a free software under the terms of MIT license and. It provides three metrics to monitor the resource usage by a code snippet: the CPU cycle/tick, the nanosecond-precision time elapse count, the maximum resident set size (RAM usage). Its interface is designed analog to GoogleTest.

[1] https://github.com/OChicken/gpqhe.
[2] https://github.com/OChicken/pmu.

1.3 Project Outline

The contents of this project are organized as follows:

Chapter 2 - Notation: this chapter is a specification of the symbols used throughout the project, including different meaning of bold face, italic face, upper and lower case letters in different contexts. Explanations are in the following text.

Chapter 3 - RLWE Cryptosystem: this chapter explains the theoretical background of RLWE cryptography, including basic algebraic number theory, the computational problems over lattices and RLWE problem, polynomial arithmetics, random distribution. The polynomial arithmetic is discussed in depth, which includes Montgomery/Barrett reduction, polynomial multiplication with *number theoretic transformation (NTT)*, conversion between multi-precision integers and *residual number systems (RNS)*, polynomial rotation and conjugation with automorphism property. They are fundamental in most RLWE-based cryptosystems, and the details in polynomial arithmetics are the primitives of GPQHE.

Chapter 4 - Leveled Homomorphic Encryption for Approximate Numbers: this chapter explains the mechanism of the basic encode/encryption/evaluation procedures in CKKS scheme, and the advanced operations supported by this scheme, including linear transformation, nonlinear functions, and ciphertext comparison.

Chapter 5 - (Encrypted) Model Predictive Control: this chapter explains the role of each component in a model predictive controller, including plant, regulator, estimator and target selector. The regulator is the component to be encrypted. Throughout this chapter, we use CSTR model from [RMD17] as a concrete case study. How to modify the closed-loop simulation to the encrypted setting is discussed in the end.

Chapter 6 - Implementations and Results Analysis: this chapter covers some implementation details of our software, the performance and precision analysis, and the comparison of unencrypted and encrypted CSTR model.

Chapter 7 - Conclusion and Future Works: this chapter covers the pros and cons of the homomorphic encrypted model predictive control based on the implementation results, and the possible optimization direction as the future works.

2 Notation

This chapter is a specification of the symbols used throughout the project, including font family and font style.

2.1 General Symbols

We use sans serif fonts to denote the proper names of functions and/or solvers, for example, quadprog (for quadratic programming), Eval (in homomorphic evaluation) etc.

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ are natural number field, integer field, rational number field, real number field and complex number field. $\mathbb{F}$ denotes the general number field. R denotes a general ring. We only consider commutative ring in this project.

$i \in [m]$ is the abbreviation for the iteration $i = 1, \cdots, m$ or $i = 0, \cdots, m-1$. E.g. $\prod_{i=1}^{m}$ is abbreviated as $\prod_{i \in [m]}$. $i \in [m] \backslash j$ is the iteration without index j: $i = 1, \cdots, j-1, j+1, \cdots, m$ or $i = 0, \cdots, j-1, j+1, \cdots, m-1$. $\{a_i\}_{i=1}^{n} \in \mathbb{F}$ or alternatively $\{a_i\}_{i \in [n]} \in \mathbb{F}$ is the abbreviation form of "$a_i \in \mathbb{F}$ for $i = 1, \cdots, n$".

$\lfloor x \rfloor, \lceil x \rceil, \lfloor x \rceil (= \lfloor x + \frac{1}{2} \rfloor \in \mathbb{Z})$ denote rounding $x \in \mathbb{R}$ to its floor, ceil, and nearest integer.

$\log, \ln, \lg$ denote logarithm base $2, e, 10$, respectively.

$\wedge$ denotes bitwise-AND; $\oplus$ denotes bitwise-XOR.

The italic alphabets *without boldface* denote numbers in $\mathbb{F}$. E.g., v_i denotes a vector element of $\boldsymbol{v}$ and A_{ij} denotes a matrix element of $\boldsymbol{A} \in \mathbb{F}^{n \times n}$. The uppercase alphabets usually denote general sets, groups, rings.

The bold italic lowercase alphabets denote either *polynomial* (in the context of cryptography) or *state* (in the context of control theory) since they have "components". E.g. $\boldsymbol{f} \in \mathbb{F}[x]$ or $\boldsymbol{f}(x) \in \mathbb{F}[x]$ denotes polynomial with coefficients in $\mathbb{F}[x]$; $\boldsymbol{x} = (x_1, \cdots, x_n) \in \mathbb{R}^n$ denotes system state, with its non-boldface counterpart with one index being the vector entry.

The bold italic uppercase alphabets denote matrices, e.g. $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. The non-boldface counterparts with two indices denote the corresponding matrix entry, e.g. A_{ij} of $\boldsymbol{A}$.

The *bold upright* alphabets denote "packing" of the objects represented by their *bold italic* counterparts. E.g. if $\{\boldsymbol{f}_i\}_{i=1}^{m} \in R$ are a series of polynomials, then $\mathbf{f} = (\boldsymbol{f}_1, \cdots, \boldsymbol{f}_m) \in R^m$; if $\{\boldsymbol{u}_k\}_{k=0}^{N-1}$ are a series of controls, then $\mathbf{u} = (\boldsymbol{u}_0, \cdots, \boldsymbol{u}_{N-1})$ is their packing; the controllability matrix $\mathbf{C} = [\boldsymbol{B}\ \boldsymbol{A}\boldsymbol{B}\ \cdots\ \boldsymbol{A}^{k-1}\boldsymbol{B}] \in \mathbb{R}^{n \times mn}$ is represented by the bold upright $\mathbf{C}$. Bold upright lowercase alphabets also denote basis in Euclidean space, e.g. $\{\mathbf{b}_i\}_{i=1}^{n} \in \mathbb{R}^n$. In some cases,

such style should not be regarded as a meaningful "vector" in a space, e.g. $\{e_i\}_{i=1}^m \in \mathbb{R}$ are sampled from Gaussian, then the packing $\mathbf{e} = (e_1, \cdots, e_m)$ is not a meaningful vector.

The identity matrix is $\boldsymbol{I}_n = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$, and by reversing its columns we yield $\boldsymbol{J}_n = \begin{pmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{pmatrix}$. $\mathbf{0}_{n\times m}$ and $\mathbb{1}_{n\times m}$ denote the matrices with all entries being zero and one, respectively.

D^S with index set $S \subset \mathbb{N}, 0 < |S| < \infty$ denotes a vector space over domain D; each component of a vector is labeled by the index in S, e.g. for a vector $\boldsymbol{v} \in \mathbb{R}^{[n]}$, its components are $\{v_i\}_{i\in[n]}$. $D^{R\times C}$ with index set $R, C \subset \mathbb{N}, 0 < |R|, |C| < \infty$ denotes a matrix space over domain D each component of a matrix is labeled by the index in $R \times C$, e.g. for a matrix $\boldsymbol{A} \in \mathbb{R}^{[m]\times[n]}$, its components are $\{A_{ij}\}_{i\in[m],j\in[n]}$.

2.2 Number Theory

card$X$ or $|X|$ denotes the cardinality of a set X. When $|X| < \infty$, the cardinality is the number of elements of X.

$A \subset B$ denotes that A is a *proper/strict* subset of B, i.e. $A \neq B$ and $|A| < |B|$. $A \subseteq B$ denotes A is a subset of B, i.e. it could be $A \subset B$ or $A = B$.

The Hadamard product, aka element-wise product or point-wise product, is denoted by $\mathbf{a} \circ \mathbf{b} = (a_1 b_1, \cdots, a_n b_n)$ for two vectors $\mathbf{a}, \mathbf{b} \in R^n$. $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the multiplication between two polynomials $\boldsymbol{a}, \boldsymbol{b} \in R$. However, in the context of mapping-composing, e.g. let π and σ are two mappings, then $\pi \circ \sigma$ and $\sigma^{-1} \circ \pi^{-1}$ are composite mappings.

Let $\boldsymbol{x}$ be a vector in a vector space equipped with norm. $\|\boldsymbol{x}\|_p$ denotes the ℓ_p norm with $p \in [1, \infty]$. When $p < \infty$, $\|\boldsymbol{x}\|_p = \left(\sum_{i\in[n]} |x_i|^p\right)^{1/p}$; when $p = \infty$, $\|\boldsymbol{x}\|_\infty = \max_{i\in[n]} |x_i|$. When $p = 2$ (i.e. Euclidean norm) is specified, we drop the subscript p and write $\|\mathbf{x}\|$ for simplicity.

(q) denotes principal ideal, which is the coset $\{kq : k \in R\}$. When R is a commutative ring, (q) is alternatively denoted by qR. When $R = \mathbb{Z}$, $q\mathbb{Z} = \{kq : k \in \mathbb{Z}\}$.

$R_q = R/qR = R/(q)$ denotes quotient rings. When $R = \mathbb{Z}$, $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z} = \{a + q\mathbb{Z} : a \in \mathbb{Z}\}$.

$d|a$ denotes d divides a ($a = kd$ for some integer k).

$a \bmod q$ or $a \bmod^+ q$ or $[a]_q$ or $[a]_q^+$ denotes the unique representative in $\mathbb{Z}_q \cap [0, q)$.

$a \bmod^\pm q$ or $[a]_q^\pm$ denotes the unique representative in $\mathbb{Z}_q \cap [-\frac{q}{2}, \frac{q}{2})$.

$\phi(n) = \prod_i (p_i - 1) p_i^{e_i - 1}$ denotes the Euler totient of $n = \prod_i p_i^{e_i}$, with $\{p_i\}$ are distinct primes and e_i the multiplicity.

$\mathbb{Z}_m^\times = \{x \in \mathbb{Z}_m : \gcd(x, m) = 1\}$ denotes a multiplicative cyclic group modulus m with size $|\mathbb{Z}_m^\times| = \phi(m)$.

char$\mathbb{F}$ denotes the characteristic of a field $\mathbb{F}$.

$R \cong R'$ denotes that rings R and R' are isomorphic.

Let α be an algebraic element. Let $\boldsymbol{f} \in \mathbb{F}[x]$ be the minimal polynomial s.t. $\boldsymbol{f}(\alpha) = 0$, then $\mathbb{K} = \mathbb{F}(\alpha) = \mathbb{F}[\alpha] \cong \mathbb{F}[x]/(\boldsymbol{f}(x))$ is a splitting field, and $\mathbb{K}$ is thus a Galois extension of $\mathbb{F}$. $[\mathbb{K} : \mathbb{F}]$ or $\dim_{\mathbb{F}}(\mathbb{K})$ denotes the degree of $\mathbb{K}$ over $\mathbb{F}$. If $\deg \boldsymbol{f} = n$, then $[\mathbb{K} : \mathbb{F}] = n$. The Galois group $\mathrm{Gal}(\mathbb{K}/\mathbb{F})$ composes of altogether n automorphisms $\tau_k : \mathbb{K} \to \mathbb{K}$.

$\mathbb{F}_{p^n}$ and/or $\mathrm{GF}(p^n)$ denotes Galois field/finite field. Here p must be prime.

$\mathrm{Tr}_{\mathbb{K}/\mathbb{F}}(\alpha) = \sum_{\tau \in \mathrm{Gal}(\mathbb{K}/\mathbb{F})} \tau(\alpha)$ and $\mathrm{N}_{\mathbb{K}/\mathbb{F}}(\alpha) = \prod_{\tau \in \mathrm{Gal}(\mathbb{K}/\mathbb{F})} \tau(\alpha)$ are trace and norm of $\alpha \in \mathbb{K}$. If $\boldsymbol{f}(x) = x^n + \sum_{i=0}^{n-1} a_i x^i \in \mathbb{F}[x]$, then they are $-a_{n-1}$ and $(-1)^n a_0$ respectively.

$a^{-1} = \mathsf{invm}(a, q)$ denotes modular inverse, calculated via extended Euclidean algorithm. If q is a prime, then $\mathsf{invm}(a, q) = a^{q-2} \mathrm{mod} q$.

$R_{\mathrm{mont}} = 2^{64}$ denotes the Montgomery R correspond to a 64-bit machine word-size.

2.3 Cryptography

λ exclusively denotes the security level.

$(a||b)$ denotes the concatenation of a and b.

$\mathcal{A}, \mathcal{B}$ denotes adversary in security games.

$\mathsf{Adv}(\mathcal{A})$ denotes the probability of the adversary $\mathcal{A}$ takes advantage / wins.

$\mathsf{negl}(\lambda)$ denotes a negligible function with security level λ as parameter.

$\mathcal{B}$ denotes the set $\{0, \cdots, 255\}$, i.e. the the of 8-bit unsigned integers (bytes). $\mathcal{B}^n$ denotes the set of byte arrays of length n and $\mathcal{B}^*$ denotes the set of byte arrays of arbitrary length (aka byte streams).

$\{0,1\}^n$ denotes the set of all bit-strings of length n (the *most significant bit (MSB)* must be 1). $\{0,1\}^{\leq n}$ denotes the set of all bit-strings of length *at most* n (the MSB may not be 1). $\{0,1\}^*$ denotes the set of all finite bit-strings. 0^n (resp., 1^n) denotes the string comprised of n zeros (resp., n ones).

$\Theta(\cdot), O(\cdot), \Omega(\cdot)$ denote asymptotic tight/upper/lower bound, respectively.

$x \leftarrow S$ denotes sampling x uniformly from a set S. E.g., $x \leftarrow \mathbb{Z}_q$ means sampling x uniformly in $\mathbb{Z}_q \cap [0, q)$.

$x \leftarrow \chi$ or $x \leftarrow \chi(S)$ denotes sampling x according to the probability distribution χ over a set S. E.g., $x \leftarrow \mathcal{D}_\sigma$ means sampling x from Gaussian distribution with standard deviation σ.

$x \leftarrow \{0,1\}^k$ denotes sampling x as a random bit-string with length k

2.4 Control Theory

$|\boldsymbol{x}_k|^2_{\boldsymbol{P}} = \boldsymbol{x}_k^\top \boldsymbol{P} \boldsymbol{x}_k$ is the abbreviation of quadratic form of a system state $\boldsymbol{x}_k$ under the weighting matrix $\boldsymbol{P}$.

We denote the *horizontal* concatenation of vectors and/or matrices using a vertical bar: $[\boldsymbol{A}|\boldsymbol{B}]$, and the *vertical* concatenation of them using a semicolon: $[\boldsymbol{A};\boldsymbol{B}]$.

$\boldsymbol{A}^+$ denotes the Moore-Penrose pseudo-inverse of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, where $m \neq n$ is permitted.

2.5 C Implementation

Typewriter family with prefix "`u`" denotes unsigned integers and prefix "`s`" denotes signed integers. E.g. `u8` denotes unsigned char, `u64` and `u128` denote unsigned 64-bit integers and unsigned 128-bit integers, while `s64` denote signed 64-bit integers. `MPI` denotes multi-precision integer.

For an unsigned integer a, $a/2^n$ is alternatively denoted by right-shift $a >> n$, and $a \cdot 2^n$ is alternatively denoted by left-shift $a << n$.

For an unsigned integer a, q, $a \bmod q = a \wedge (q-1)$ iff the modulo $q = 2^k$ is 2's power, because the binary representation of $q-1$ is 1^k.

$\mathsf{nbits}(a) = \lfloor \log a \rfloor + 1 = \lceil \log a \rceil$ denotes the number of bits / the length of the binary representation of $a \in \mathbb{Z}_{>0}$ written with the MSB being 1. It holds: $\log a < \mathsf{nbits}(a) \leq \log a + 1$.

$\mathsf{tbit}(a,k) = (a >> k) \wedge 1$ denotes testing the k-th bit of a in order to see whether it is 1 or 0, where $k = 0, \cdots, \mathsf{nbits}(a) - 1$.

$\mathsf{sbit}(a,k) = a|(1 << k)$ denotes setting the k-th bit of a to 1, where $k = 0, \cdots, \mathsf{nbits}(a) - 1$.

$\mathsf{bitrev}(a,n) = \sum_{k=0}^{\mathsf{nbits}(n-1)-1} \mathsf{tbit}(a,k) << (\mathsf{nbits}(n-1)-1-k)$ denotes the bit-reversal operation to a, where n must be 2's power and the permitted integer a is $0, \cdots, n-1$.

3 RLWE Cryptosystem

3.1 Introduction

Lattice-based homomorphic encryption has its roots in *Ring Learning With Errors (RLWE)*. This chapter explains the necessary mathematical background of RLWE cryptography, which is beneficial to understand each component of GPQHE and/or any other RLWE implementation it has referenced, e.g. HEAAN, NewHope and Crystal-Kyber. In general, the architecture of a complete RLWE cryptosystem is described in Figure 3.1.

NewHope has a well-stated algorithm specification submitted to NIST PQC election, and its successor Crystal-Kyber, though based on *Module Learning With Error (MLWE)*, has been selected as the post-quantum public-key encryption standardization, so most parts in this chapter, especially Section 3.3, are referred the specification of NewHope.

Section 3.2 explains the algebraic number theory background required to understand the basic notations. Section 3.3 briefly explains the history of lattice problem and RLWE problem. Section 3.4 explains the cornerstones of the polynomial arithmetics implemented in GPQHE. Section 3.5 explains the random distributions involved in RLWE. Section 3.6 explain the CPA-secure key encapsulation mechanism used in RLWE, which is a collection of all of the mentioned techniques.

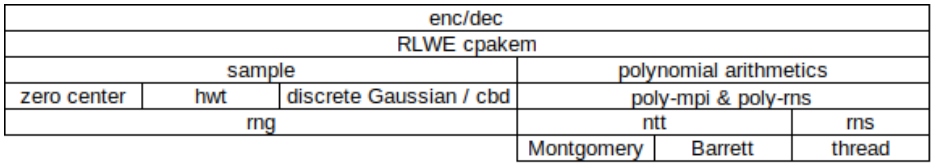

Figure 3.1: The architecture of a general RLWE cryptosystem. The abbreviations from bottom to top, left to right, are: rng - random number generator; hwt - Hamming weight ternary; cbd - centered binomial distribution; Montgomery and Barrett refer to their reduction; ntt - number theoretic transform; rns - residual number system; mpi - multi-precision integer; cpa - chosen plaintext attack secure; kem - key encapsulation mechanism; enc - encryption; dec - decryption.

3.2 Algebraic Number Theory Background

For a complete introduction on the standard concepts, we recommend [Rot15, Qiu03, Con22a, LPR10, LPR13], chapter 31 of [THC11], and Appendix C of [DPSZ12]. This section basically

follows Section 2 and 8 of [LPR13].

3.2.1 Finite Cyclic Group

The *finite cyclic groups* are naturally equipped with multiplication, which implies that every element in G can be expressed as a power of $a \in G$, i.e. a^k for some $k \in \mathbb{Z}$. Let $\langle a \rangle$ denote the set of non-repeated a^k for $k \in \mathbb{Z}$, we say that a generates a finite cyclic group $\langle a \rangle \subseteq G$ and it satisfies $a^{|\langle a \rangle|} = 1$. Regarding the relation between $|\langle a \rangle|$ and $|G|$, we use the Lagrange theorem, which illustrates the relation between the cardinality of a finite group and its subgroup:

Theorem 1 (Lagrange)**:** If H is a subgroup of a finite group G, then $|H|$ is a divisor of $|G|$: $|G| = [G : H]|H|$

So that, if $|\langle a \rangle| = |G|$, then $\langle a \rangle = G$ and a is called the *generator* of G (when $|G| \geq 3$, the number of G's generators are $\phi(|G|)$). In many cases, $|\langle a \rangle| < |G|$, then $\langle a \rangle \subset G$, but Theorem 1 states that $(|\langle a \rangle|)|(|G|)$. $a^{|G|} = 1$ holds in both cases.

An application of the finite cyclic group is the investigation of *m-th roots of unity*, the elements $\zeta \in \mathbb{F}$ that solve $x^m - 1 = 0$ for a given $\mathbb{N} \ni m \geq 1$ and a general field $\mathbb{F}$. Let $U_m \subseteq \mathbb{F}$ (of order $|U_m| = m$) denote the cyclic subgroup formed by these distinct roots. Only $\phi(m) = |\mathbb{Z}_m^\times|$ of U_m are *primitive*, i.e. the elements $\zeta \in U_m$ that satisfying $\langle \zeta \rangle = U_m$, which are exactly the generators of U_m; while other $\zeta \in U_m$ satisfy $\langle \zeta \rangle \subset U_m$ and $\zeta^d - 1 = 0$, where $d = |\langle \zeta \rangle|$ and $d|m, d < m$. For example, consider $\mathbb{F}_7$ and $m = |\mathbb{F}_7^\times| = 6$: the $2(= \phi(6))$ generators of $U_6 = \{1, 2, 3, 4, 5, 6\}$ are 3 and 5.

3.2.2 General Number Field

There is an important theorem in algebra:

Theorem 2 (Fundamental Theorem of Algebra)**:** Complex field $\mathbb{C}$ is algebraically closed, i.e. a non-constant polynomial $\boldsymbol{f} \in \mathbb{C}[x]$ has $n = \deg \boldsymbol{f}$ complex roots.

Corollary 3: A non-constant $\boldsymbol{f} \in \mathbb{R}[x]$ with $n = \deg \boldsymbol{f}$ has s_1 real roots and s_2 complex conjugate pairs of roots, then it holds: $n = s_1 + 2s_2$.

Let an *algebraic number* (resp. *algebraic integer*) $\zeta \in \mathbb{C}$ be the root of a polynomial (resp. monic polynomial) with coefficients in $\mathbb{Q}$ (resp. $\mathbb{Z}$). The *minimal polynomial* of ζ is the unique monic irreducible $\boldsymbol{f} \in \mathbb{Q}[x]$ with degree n which has ζ as a root. Then we have a *simple algebraic extension* $\mathbb{K} = \mathbb{Q}(\zeta) \cong \mathbb{Q}[x]/(\boldsymbol{f}(x))$, the field of rational polynomials with degree less than n modulo the polynomial $\boldsymbol{f}(x)$. Note that $\boldsymbol{f}$ has altogether n roots, they are related by the *automorphism* $\tau_k \in \mathrm{Gal}(\mathbb{K}/\mathbb{Q})$: by acting τ_k transitively (for n times) to a certain ζ gets all roots. We say these roots lie on the same $\mathrm{Gal}(\mathbb{K}/\mathbb{Q})$-*orbit* under the action of $\tau_k \in \mathrm{Gal}(\mathbb{K}/\mathbb{Q})$, and they are $\mathbb{Q}$-*conjugate*. The ring of integers $R = \mathcal{O}_\mathbb{K} \subset \mathbb{K}$ is the subring of $\mathbb{K}$ consisting of all elements whose minimal polynomial has $\mathbb{Z}$ coefficients.

According to Corollary 3, $\mathbb{K} = \mathbb{Q}(\zeta) \cong \mathbb{Q}[x]/(\boldsymbol{f}(x))$ is said to have *signature* (s_1, s_2). So there are n field *embeddings* $\{\sigma_i : \mathbb{K} \hookrightarrow \mathbb{R}\}_{1 \leq i \leq s_1}$ and $\{\sigma_{s_1+i} : \mathbb{K} \hookrightarrow \mathbb{C}\}_{1 \leq i \leq 2s_2}$. Let $\zeta_i = \sigma_i(\zeta)$

for $i \in [n]$ be the roots of $\boldsymbol{f}$, which are numbered in the following way: $\zeta_i \in \mathbb{R}$ for $1 \le i \le s_1$, and $\zeta_{s_1+i} = \overline{\zeta_{s_1+2s_2-i}}$ for $1 \le i \le s_2$. The numbering scheme of the complex roots is not unique: some literature define $\zeta_{s_1+i} = \overline{\zeta_{s_1+s_2+i}}$, e.g. [LPR10, DPSZ12]. The complex roots are organized in such a centered-symmetric manner such that the conjugate property behaves the same as the FFT.

$\mathbb{K}$ can be viewed as a vector space over $\mathbb{Q}$ of dimension $n = \phi(m) = [\mathbb{K} : \mathbb{Q}]$. For each $i \in [n]$, the root ζ_i spans a basis $\{\zeta_i^j\}_{j\in[n]} = (1, \zeta_i, \cdots, \zeta_i^{n-1}) \in \mathbb{K}^{[n]}$, so-called the *power basis* of $\mathbb{K}$.

$\{\sigma_i\}_{i=1}^n$ bring an element $\boldsymbol{a}(\zeta) = \sum_{j=0}^{n-1} a_j \zeta^j \in \mathbb{K}$ to $\mathbb{R}$ and $\mathbb{C}$ via $\sigma_i(\boldsymbol{a}) = \sum_{j=0}^{n-1} a_j \sigma_i(\zeta)^j = \sum_{j=0}^{n-1} a_j \zeta_i^j = \boldsymbol{a}(\zeta_i)$, which is nothing but the evaluation of $\boldsymbol{a}(\zeta_i)$ at the roots $\{\zeta_i\}_{i=1}^n$ of $\boldsymbol{f}(x)$ ([CKKS17]). Define the *canonical embedding* $\sigma : \mathbb{K} \hookrightarrow \mathbb{R}^{s_1} \times \mathbb{C}^{2s_2}$

$$\begin{aligned}\sigma(\boldsymbol{a}) &= \{\sigma_i(\boldsymbol{a})\}_{i\in[n]} = \{\boldsymbol{a}(\zeta_i)\}_{i\in[n]} \\ &= (\underbrace{\boldsymbol{a}(\zeta_1), \cdots, \boldsymbol{a}(\zeta_{s_1})}_{\in\mathbb{R}^{s_1}}, \underbrace{\boldsymbol{a}(\zeta_{s_1+1}), \cdots, \boldsymbol{a}(\zeta_{s_1+s_2})}_{\in\mathbb{C}^{s_2}}, \underbrace{\overline{\boldsymbol{a}(\zeta_{s_1+s_2})}, \cdots, \overline{\boldsymbol{a}(\zeta_{s_1+1})}}_{\in\mathbb{C}^{s_2}}) \in \mathbb{R}^{s_1} \times \mathbb{C}^{2s_2} \end{aligned} \tag{3.1}$$

Now define the following $\mathbb{H}$ space (with "$\sigma(\mathbb{K})$" be the abbreviation of "$\sigma(\boldsymbol{a})\ \forall \boldsymbol{a} \in \mathbb{K}$"):

$$\mathbb{H} = \{\sigma(\mathbb{K}) : \sigma_{s_1+j} = \overline{\sigma_{n+1-j}}\ \forall j = 1, \cdots, s_2\} \subseteq \mathbb{R}^{s_1} \times \mathbb{C}^{2s_2} \tag{3.2}$$

associated with inner product, ℓ_p-norm and ℓ_∞-norm induced by $\mathbb{C}^n$. With the induced norms, we define the *canonical embedding norm*

$$\|\boldsymbol{a}\|_p^{\mathsf{can}} = \|\sigma(\boldsymbol{a})\|_p = \left(\sum_{i\in[n]} |\sigma_i(\boldsymbol{a})|^p\right)^{1/p} = \left(\sum_{i\in[n]} |\boldsymbol{a}(\zeta_i)|^p\right)^{1/p} \tag{3.3}$$

3.2.3 Cyclotomic Number Fields and Embeddings

The term "cyclotomic" means "circle-dividing", which comes from the fact that the m-th roots of unity in $\mathbb{C}$ divide a circle into m arcs of equal length. There are not many general methods known for constructing abelian extensions (i.e. the Galois extensions with abelian Galois group); cyclotomic extensions are essentially the only construction that works over all fields.

Corresponding to the $\mathbb{Q}$-conjugate primitive m-th roots of unity $\{\omega_m^k\}_{k\in\mathbb{Z}_m^\times}$ with $\omega_m = e^{2\pi i/m} \in \mathbb{C}$ which solve $x^m - 1 = 0$, a series of cyclotomic extensions $\{\mathbb{Q}(\omega_m^k)\}_{k\in\mathbb{Z}_m^\times}$ have a common minimal polynomial in $\mathbb{Q}$ which have no repeated roots. Let this polynomial be $\Phi_m(x) \in \mathbb{Q}[x]$ and define it as such linear product (guaranteed by Theorem 2):

$$\Phi_m(x) = \prod\nolimits_{k\in\mathbb{Z}_m^\times} \left(x - \omega_m^k\right) \tag{3.4}$$

with degree $\deg \Phi_m = \phi(m)$, thus we yield the following *separable normal field extension*:

$$\mathbb{K} = \mathbb{Q}(\zeta) \cong \mathbb{Q}[x]/(\Phi_m(x)) \tag{3.5}$$

with ζ takes the values $\{\omega_m^k\}_{k\in\mathbb{Z}_m^\times}$, the roots of $\Phi_m(x)$. m is called *cyclotomic index* and the field $\mathbb{K} = \mathbb{Q}(\zeta_m)$ is *m-th cyclotomic number field*, a Galois extension of $\mathbb{Q}$. Note that among m different m-th roots of unity of $x^m - 1 = 0$, only $\phi(m)$ of them are primitive, while others are primitive d-th roots of unity with $d|m, d < m$. Each d gives a cyclotomic polynomial $\Phi_d(x)$. On another hand, Theorem 2 states that $x^m - 1 = \prod_{\zeta^m=1, d|m}(x - \zeta)$, so we have the relation $x^m - 1 = \prod_{d|m} \Phi_d(x)$ and $m = \sum_{d|m} \phi(d)$. Here are some examples of $\Phi_m(x)$:

- $\Phi_1(x) = x - 1$;
- If m is odd prime, then $\Phi_m(x) = \frac{x^m-1}{x-1} = x^{m-1} + x^{m-2} + \cdots + x + 1$ and $n = \phi(m) = m - 1$;
- If m is a 2's power, then $\Phi_m(x) = x^{m/2} + 1 = x^n + 1$ and $n = \phi(m) = m/2$.

By definition, the modulus polynomial $\Phi_m(x)$ has no real roots (in contrast to Section 3.2.2), so $s_1 = 0, s_2 = n/2$. The canonical embedding is $\sigma_i(\zeta) = \omega_m^i$ and satisfies the conjugate relation $\sigma_i = \overline{\sigma_{n+1-i}}\ \forall i = 1, \cdots, n/2$. Under this setting, $\mathbb{H}$ in Eq (3.2) becomes

$$\mathbb{H} = \{\sigma(\mathbb{K}) : \sigma_i = \overline{\sigma_{n+1-i}}\ \forall i = 1, \cdots, n/2\} \subseteq \mathbb{C}^n \tag{3.6}$$

The set of all automorphisms of $\mathbb{K}$ that fix the base field $\mathbb{Q}$ forms a Galois group $\mathrm{Gal}(\mathbb{K}/\mathbb{Q})$, which contains such automorphism $\tau_k : \mathbb{K} \to \mathbb{K}$ with $k \in \mathbb{Z}_m^\times$. It is conventionally defined by $\tau_k : \zeta \mapsto \omega_m^k$ for $k \in \mathbb{Z}_m^\times$; However, when working on 2's power cyclotomic index m, we can do less works: $\mathbb{Z}_m^\times$ is the collection of all odd numbers less than m, and further exploit the conjugate relation, we can define the automorphism as $\tau_i : \zeta \mapsto \omega_m^{g^i \bmod m}$, with g is chosen as the order of $m/4 = n/2$ modulo m, e.g. $g = 5$ as in [CHK+18]. So that, by iterating $i \in [n/2]$, we essentially yield all elements in $\mathbb{H}$.

Finally, we quote the key fact from algebraic number theory [LPR13]:

Proposition 4: Let m have prime-power factorization $m = \prod_l m_l$. Then $\mathbb{K} = \mathbb{Q}(\zeta_m)$ is isomorphic to the tensor product $\bigotimes_l \mathbb{K}_l$ with $\mathbb{K}_l = \mathbb{Q}(\zeta_{m_l})$

$$\mathbb{K} \cong \bigotimes\nolimits_l \mathbb{K}_l = \bigotimes\nolimits_l \mathbb{Q}(\zeta_{m_l}) = \mathbb{Q}(\zeta_{m_1}, \cdots, \zeta_{m_l}) = \mathbb{Q}[x_1, \cdots, x_l] / (\Phi_{m_1}(x_1), \cdots, \Phi_{m_l}(x_l)) \tag{3.7}$$

The isomorphism is given by the correspondence $\prod_l a_l \leftrightarrow (\bigotimes_l a_l)$, where $a_l \in \mathbb{K}$ and on the left we implicitly embed each $a_l \in \mathbb{K}_l$ to $\mathbb{K}$.

3.2.4 Ring of Integers and Ideal Lattice

Let $R = \mathcal{O}_\mathbb{K} \subset \mathbb{K}$ denotes the set of all *algebraic integers* in a number field $\mathbb{K}$. Under the usual addition and multiplication operations in $\mathbb{K}$, this set forms a *ring of integers* of $\mathbb{K}$.

For the cyclotomic number field, where $\mathbb{K} = \mathbb{Q}(\zeta_m)$ of degree $n = \phi(m)$, the ring of integers happens to be

$$R = \mathcal{O}_\mathbb{K} = \mathbb{Z}[\zeta_m] \cong \mathbb{Z}[x]/(\Phi_m(x)) \tag{3.8}$$

and hence the power basis $\{\zeta_m^k\}_{k\in[n]}$ is $\mathbb{Z}$-basis of cardinality n. If $m = \prod_l m_l$ is the prime-power factorization, then R can be viewed as a tensorial product of subrings $R_l = \mathbb{Z}[\zeta_{m_l}]$:

$$R \cong \bigotimes\nolimits_l R_l = \bigotimes\nolimits_l \mathbb{Z}[\zeta_{m_l}] = \mathbb{Z}[x_1, \cdots, x_l] / (\Phi_{m_1}(x_1), \cdots, \Phi_{m_l}(x_l)) \tag{3.9}$$

Let $\mathbb{H}$ be the general form in Eq (3.2). The *lattice* is defined as a discrete additive subgroup of $\mathbb{H}$. We deal here exclusively with full-rank lattices generated as an ordered set of all integer linear combination of $\mathbf{B} = \{\mathbf{b}_i\}_{i\in[n]} \subset \mathbb{H}$ by

$$\Lambda = \mathcal{L}(\mathbf{B}) = \left\{\sum\nolimits_{i\in[n]} x_i\mathbf{b}_i : x_i \in \mathbb{Z}\right\} \subset \mathbb{H} \tag{3.10}$$

Its *dual lattice* is $\Lambda^\vee = \{\boldsymbol{y} \in \mathbb{H} : \forall \boldsymbol{x} \in \Lambda, \langle \boldsymbol{x}, \bar{\boldsymbol{y}}\rangle = \sum_i x_i\bar{y}_i \in \mathbb{Z}\}$. It's clear that $(\Lambda^\vee)^\vee = \Lambda$. The *minimum distance* $\lambda_1(\Lambda)$ of a lattice Λ in some norm $\|\cdot\|_p$ is the length of a shortest nonzero lattice vector: $\lambda_1^{(p)}(\Lambda) = \min_{\boldsymbol{x}\in\Lambda\setminus\{\mathbf{0}\}} \|\boldsymbol{x}\|_p$. If $p = 2$ (i.e. Euclidean norm) is specified, we use $\lambda_1(\Lambda)$ as simplified notation.

The ideal $\mathcal{I} \subseteq R = \mathcal{O}_\mathbb{K} \subset \mathbb{K}$ is called *integral ideal*. The ideal $\mathcal{I} \subset \mathbb{K}$ is called *fractional ideal* if $d\mathcal{I} \subseteq R$ is an integral ideal for some $d \in R$. Any fractional ideal $\mathcal{I}$ embeds under σ as a lattice $\sigma(\mathcal{I})$ in $\mathbb{H}$, which is the *ideal lattice*. Concretely, this is nothing but replacing the condition "$\sigma(\mathbb{K})$" in the definition of $\mathbb{H}$ in Eq (3.2,3.6) by "$\sigma(\mathcal{I})$", then redefine the basis $\mathbf{B} \subset \mathbb{H}$ and lattice $\Lambda = \mathcal{L}(\mathbf{B})$ analogously.

For any fractional ideal $\mathcal{I} \subset \mathbb{K}$, its *dual* is $\mathcal{I}^\vee = \{a \in \mathbb{K} : \mathrm{Tr}_{\mathbb{K}/\mathbb{Q}}(a\mathcal{I}) \subseteq \mathbb{Z}\}$. It's clear that $(\mathcal{I}^\vee)^\vee = \mathcal{I}$, and, under the canonical embedding into $\mathbb{H}$, $\mathcal{I}^\vee$ embeds exactly as the complex conjugate of the dual lattice: $\sigma(\mathcal{I}^\vee) = \overline{\sigma(\mathcal{I})}$. R's dual, $R^\vee$, is often called *codifferent ideal* and it satisfies $R \subset R^\vee$. Particularly, when $R = \mathbb{Z}[x]/(x^n + 1)$ being a ring of integer with 2's power cyclotomic index, then $R^\vee = n^{-1} \cdot R$ [Con22b, CKKS17]. For a fractional ideal $\mathcal{I}$, there is a relation $\mathcal{I}^\vee = \mathcal{I}^{-1}R^\vee$.

3.2.5 Prime Splitting and RNS Decomposition

Residual number system (RNS) utilizes *Chinese Remainder Theorem (CRT)* to express a large integer as the remainder on each CRT basis.

Consider an integer prime $q \equiv 1 \mod m$, its principal ideal $(q) \subset R = \mathbb{Z}[\zeta_m]$, the field $\mathbb{Z}_q$ has primitive m-th roots of unity $\{\omega_m^k\}_{k\in\mathbb{Z}_m^\times} \in \mathbb{Z}_q$ (the number of these distinct roots is $n = \phi(m)$), because $\mathbb{Z}_q^\times$ (the multiplicative group of $\mathbb{Z}_q$) is cyclic with order $q-1$. Therefore, the cyclotomic polynomial $\Phi_m(x)$ factors in $\mathbb{Z}_q[x]$ as $\Phi_m(x) = \prod_{k\in\mathbb{Z}_m^\times}(x - \omega_m^k)$. The ideal $(q) \subset \mathcal{O}_\mathbb{K}$ then "*splits completely*" into n distinct prime ideals: $(q) = \prod_{k\in\in\mathbb{Z}_m^\times} \mathfrak{p}_k$, where $\mathfrak{p}_k = \langle q, \zeta_m - \omega_m^k\rangle = (q) + (\zeta_m - \omega_m^k)$ is prime and has norm q. Therefore, each quotient ring $R_{\mathfrak{p}_k} = R/(\mathfrak{p}_k) \cong \mathbb{Z}_q$ via the mapping $\zeta_m \mapsto \omega_m^k$.

The CRT says that if $\{\mathfrak{p}_k\}_{k\in\mathbb{Z}_m^\times}$ are coprime ideals in R, then the natural ring homomorphism from $R/(\prod_{k\in\mathbb{Z}_m^\times}\mathfrak{p}_k) = R/(q)$ to the product ring $\prod_{k\in\mathbb{Z}_m^\times}(R/\mathfrak{p}_k)$ is in fact an isomorphism:

$$R/(q) = R\Big/\Big(\prod\nolimits_{k\in\mathbb{Z}_m^\times}\mathfrak{p}_k\Big) \cong \prod\nolimits_{k\in\mathbb{Z}_m^\times}(R/\mathfrak{p}_k) \tag{3.11}$$

with the following lemma, we support efficient operations in $R_q = R/qR$:

Lemma 5: Let $q \equiv 1 \mod m$ be prime, m-th roots of unity $\{\omega_m^k\}_{k\in\mathbb{Z}_m^\times} \in \mathbb{Z}_q$ and ideals $\mathfrak{p}_k$ defined as above, then the natural ring homomorphism $R/(q) \to \prod_{k\in\mathbb{Z}_m^\times}(R/\mathfrak{p}_k) \cong (\mathbb{Z}_q)^n$ is an isomorphism.

3.3 RLWE Problem

Here we briefly introduce the computational problems based on lattices and learning with errors, based on the overview of [Pei16, D'A21] and the specification of NewHope. Their hardness assumptions are the security guard for the post quantum cryptography.

3.3.1 A Gentle Example: Knapsack Problem

In the theory of computational complexity and its application to the foundations of cryptography, there is a famous knapsack problem and its variants. The simplest variant is the *0-1 knapsack problem*: Given a set of items $i = 1, \cdots, m$ with weight w_i and value v_i, determine the number of each item x_i to include in a collection so that the total weight is less than W and the total value is as large as possible:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} w_i x_i \le W \text{ and } x_i \in \{0, 1\} \end{aligned}$$

Changing the constraint conditions yields many different variants. The *optimization* task can be solved by *dynamic programming*[1], a pseudo-polynomial time algorithm. However, the *decisional* task — given v_i and a target value t, determine whether there is a $(x_1, \cdots, x_m)$ to exactly fill the knapsack with target value t

$$\sum_{i=1}^{m} v_i x_i = t \quad \text{with } \{x_i\}_{i=1}^{m} \in S$$

(where S could be $\{0, 1\}$ or $\{0, 1, \cdots, c\}$ or even unbounded) is NP-complete hard: the required exhaustive trial and error search over all 2^m possible $(x_1, \cdots, x_m)$ is computationally infeasible if m is larger than 100 or 200. The above decisional knapsack problem is a simple instance of various variants of the *subset-sum problems*. The enthusiasm generated by the knapsack or *subset-sum based cryptosystem* started from R.Merkle and M.Hellman [MH78] in 1978 to exploit the NP-hard property to guarantee the security of the cryptosystem [Mic07].

3.3.2 Computational Problem on Lattices

The *shortest vector problem (SVP)* and the *closest vector problem (CVP)* are two fundamental problems in lattices and their conjectured intractability is the foundation for a large number of cryptographic applications of lattices.

Definition 6 ((Approximate) Shortest Vector Problem (SVP))**:** Given an arbitrary basis $\mathbf{B}$ of some lattice $\Lambda = \mathcal{L}(\mathbf{B})$, find a shortest vector $\boldsymbol{v} \in \Lambda \backslash \{\mathbf{0}\}$, such that $\|\boldsymbol{v}\| = \lambda_1(\Lambda)$. In the γ-approximate SVP_γ, for a slack parameter $\gamma(n) \ge 1$, find a shortest vector $\boldsymbol{v} \in \Lambda \backslash \{\mathbf{0}\}$, such that $\|\boldsymbol{v}\| \le \gamma(n) \cdot \lambda_1(\Lambda)$.

[1] This is a typical linear/integer programming, in contrast to the quadratic programming in Section 5.2.3.

Definition 7 ((Approximate) Closest Vector Problem (CVP))**:** Given an arbitrary basis $\mathbf{B}$ of some lattice $\Lambda = \mathcal{L}(\mathbf{B})$ and a target vector $\boldsymbol{t} \in \Lambda$, find the vector $\boldsymbol{v} \in \Lambda$, such that $\|\boldsymbol{v} - \boldsymbol{t}\|$ is minimized. In the γ-approximate CVP_γ, for a slack parameter $\gamma(n) \geq 1$, find a vector $\boldsymbol{v} \in \Lambda$, such that $\|\boldsymbol{v} - \boldsymbol{t}\| \leq \gamma \cdot \inf_{\boldsymbol{w} \in \Lambda} \|\boldsymbol{w} - \boldsymbol{t}\|$. Here, $\inf_{\boldsymbol{w} \in \Lambda} \|\boldsymbol{w} - \boldsymbol{t}\|$ is the distance of $\boldsymbol{t}$ to Λ.

The notation $\gamma(n)$ implies that γ is a function of the lattice dimension. The SVP_γ and CVP_γ are easier for an increasing γ. Known (up till [Pei16] in 2016) *polynomial-time* algorithms like LLL [LLL82] obtain slightly sub-exponential slack parameter $\gamma \sim 2^{\Theta(n \log\log n / \log n)}$ for both problems.

For a good introduction to the history of lattices in cryptography, O.Regev recommended [DM02] in [Reg09].

3.3.3 Ring-LWE Problem

The *learning with errors (LWE)* problem was popularized by O.Regev in his seminal work [Reg09], where he showed that, under a quantum reduction, solving a random LWE instance is as hard as solving certain worst-case instances of certain lattice problems. The LWE problem is usually used to build primitives such as CPA or CCA-secure public-key encryption, *identity-based encryption (IBE)*, or fully homomorphic encryption schemes [Reg10].

LWE is parameterized by n, m, q, χ: n the lattice dimension (the number of unknowns), $m \geq n$ the available samples, $q \geq 2$ the modulus, χ the error distribution. All of them depend on a pre-specified security level λ. There are two main versions of the LWE problem: the *search-LWE (sLWE)* is to find the secret $\boldsymbol{s}$ while the *decisional-LWE (dLWE)* is to distinguish between LWE samples and uniformly random ones:

Definition 8 (Search LWE - $\mathsf{sLWE}_{n,m,q,\chi}$)**:** For a secret $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$, sample $\{\boldsymbol{a}_i\}_{i=1}^m \leftarrow \mathbb{Z}_q^n$, $\{e_i\}_{i=1}^m \leftarrow \chi(\mathbb{Z}_q)$, output $(\boldsymbol{a}_i, b_i = [\langle \boldsymbol{s}, \boldsymbol{a}_i \rangle + e_i]_q)_{i=1}^m$. Task: find $\boldsymbol{s}$ from the output. Formally,

$$\mathsf{Adv}^{\mathsf{sLWE}}_{n,m,q,\chi}(\mathcal{A}) = \Pr\left[\boldsymbol{s}' = \boldsymbol{s} : \begin{array}{c} \boldsymbol{s} \leftarrow \mathbb{Z}_q^n; \{\boldsymbol{a}_i\}_{i=1}^m \leftarrow \mathbb{Z}_q^n; \{e_i\}_{i=1}^m \leftarrow \chi(\mathbb{Z}_q) \\ \boldsymbol{s}' \leftarrow \mathcal{A}\left((\boldsymbol{a}_i, b_i = [\langle \boldsymbol{s}, \boldsymbol{a}_i \rangle + e_i]_q)_{i=1}^m\right) \end{array}\right] \tag{3.12a}$$

The $\mathsf{sLWE}_{n,m,q,\chi}$ assumption is that $\mathsf{Adv}^{\mathsf{sLWE}}_{n,m,q,\chi}(\mathcal{A}) < \mathsf{negl}(\lambda)$.

Definition 9 (Decisional LWE - $\mathsf{dLWE}_{n,m,q,\chi}$)**:** For a secret $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$, sample $\{\boldsymbol{a}_i\}_{i=1}^m \leftarrow \mathbb{Z}_q^n$, $\{e_i\}_{i=1}^m \leftarrow \chi(\mathbb{Z}_q)$, output $(\boldsymbol{a}_i, b_i = [\langle \boldsymbol{s}, \boldsymbol{a}_i \rangle + e_i]_q)_{i=1}^m$. Task: distinguish this output from a uniform distribution. Formally,

$$\mathsf{Adv}^{\mathsf{dLWE}}_{n,m,q,\chi}(\mathcal{A}) = \left| \begin{array}{l} \Pr\left[b' = 1 : \begin{array}{c} \boldsymbol{s} \leftarrow \mathbb{Z}_q^n; \{\boldsymbol{a}_i\}_{i=1}^m \leftarrow \mathbb{Z}_q^n; \{e_i\}_{i=1}^m \leftarrow \chi(\mathbb{Z}_q) \\ b' \leftarrow \mathcal{A}\left((\boldsymbol{a}_i, b_i = [\langle \boldsymbol{s}, \boldsymbol{a}_i \rangle + e_i]_q)_{i=1}^m\right) \end{array}\right] \\ - \Pr\left[b' = 1 : \begin{array}{c} \{\boldsymbol{a}_i\}_{i=1}^m \leftarrow \mathbb{Z}_q^n; \{u_i\}_{i=1}^m \leftarrow \mathbb{Z}_q \\ b' \leftarrow \mathcal{A}\left((\boldsymbol{a}_i, u_i)_{i=1}^m\right) \end{array}\right] \end{array} \right| \tag{3.12b}$$

The $\mathsf{dLWE}_{n,m,q,\chi}$ assumption is that $\mathsf{Adv}^{\mathsf{dLWE}}_{n,m,q,\chi}(\mathcal{A}) < \mathsf{negl}(\lambda)$.

If the error term $\{e_i\}_{i=1}^m$ is absent, then the output of LWE is $(\boldsymbol{a}_i, b_i = [\langle \boldsymbol{s}, \boldsymbol{a}_i \rangle]_q)_{i=1}^m$, which is nothing but a linear equation and thus the sLWE would be easily solved.

A nice property of the LWE problem is the equivalence of the sLWE and dLWE: a solver for the sLWE can be used to solve the dLWE problem, and vice versa [AAB+17]:

Theorem 10 (Decision to Search Reduction for LWE)**:** For $\mathsf{LWE}_{n,m,q,\chi}$, if there exists a *probabilistic polynomial time (PPT)* algorithm that solves $\mathsf{dLWE}_{n,m,q,\chi}$ with non-negligible probability, then there exists a PPT algorithm that solves $\mathsf{sLWE}_{n,m',q,\chi}$ for some $m' = m \cdot \mathsf{poly}(n)$ with non-negligible probability.

The lattices underlying LWE and its variants are *module lattices*, as in *NTRU* [HPS98] proposed in 1998[2], and its hardness can be related to the worst case hardness of finding short vectors in ideal lattices [LPR10]. *Ring-LWE*, defined and studied by V.Lyubashevsky, C.Peikert and O.Regev [LPR10] in 2010, is one of a variants of LWE relying on the ring of integer of a number field (or polynomial rings). The general definition involves the so-called *codifferent* $R^\vee$ (in Section 3.2.4 and Section 2.5.4 of [LPR13]).

RLWE is parameterized by R_q, m, q, χ and all of them depend on a pre-specified security level λ. m, q, χ have the same meaning as that in LWE, while $R_q = R/qR$ is a general ring of integer. For simplicity we restrict our definition to the case of *cyclotomic ring* with a 2's power cyclotomic index (c.f. Section 3.2.3). In this case, $R = \mathbb{Z}[x]/(x^n + 1)$ and $R^\vee = n^{-1} \cdot R$. The sample number m is required to be "*polynomially many*" according to [LPR10], and a typical choice of m is $m = n$.[3]

Definition 11 (Search RLWE - $\mathsf{sRLWE}_{R_q,m,q,\chi}$)**:** For a secret $\boldsymbol{s} \leftarrow R^\vee$, sample $\{\boldsymbol{a}_i\}_{i=1}^m \leftarrow R_q$, $\boldsymbol{e} \leftarrow \chi(R^\vee)$, output $(\boldsymbol{a}_i, \boldsymbol{b}_i = [\boldsymbol{a}_i \cdot \boldsymbol{s} + \boldsymbol{e}]_q)_{i=1}^m$. Task: find $\boldsymbol{s}$ from the output. Formally,

$$\mathsf{Adv}^{\mathsf{sRLWE}}_{R_q,m,q,\chi}(\mathcal{A}) = \Pr\left[\boldsymbol{s}' = \boldsymbol{s} : \begin{array}{c} \boldsymbol{s} \leftarrow R^\vee; \{\boldsymbol{a}_i\}_{i=1}^m \leftarrow R_q; \boldsymbol{e} \leftarrow \chi(R^\vee) \\ \boldsymbol{s}' \leftarrow \mathcal{A}\left((\boldsymbol{a}_i, \boldsymbol{b}_i = [\boldsymbol{a}_i \cdot \boldsymbol{s} + \boldsymbol{e}]_q)_{i=1}^m\right) \end{array}\right] \tag{3.13a}$$

The $\mathsf{sRLWE}_{R_q,m,q,\chi}$ assumption is that $\mathsf{Adv}^{\mathsf{sRLWE}}_{R_q,m,q,\chi}(\mathcal{A}) < \mathsf{negl}(\lambda)$.

Definition 12 (Decisional RLWE - $\mathsf{dRLWE}_{R_q,m,q,\chi}$)**:** For a secret $\boldsymbol{s} \leftarrow R^\vee$, sample $\{\boldsymbol{a}_i\}_{i=1}^m \leftarrow R_q$, $\boldsymbol{e} \leftarrow \chi(R^\vee)$, output $(\boldsymbol{a}_i, \boldsymbol{b}_i = [\boldsymbol{a}_i \cdot \boldsymbol{s} + \boldsymbol{e}]_q)_{i=1}^m$. Task: distinguish this output from a uniform distribution. Formally,

$$\mathsf{Adv}^{\mathsf{dRLWE}}_{R_q,m,q,\chi}(\mathcal{A}) = \left| \begin{array}{l} \Pr\left[b' = 1 : \begin{array}{c} \boldsymbol{s} \leftarrow R^\vee; \{\boldsymbol{a}_i\}_{i=1}^m \leftarrow R_q; \boldsymbol{e} \leftarrow \chi(R^\vee) \\ b' \leftarrow \mathcal{A}\left((\boldsymbol{a}_i, \boldsymbol{b}_i = [\boldsymbol{a}_i \cdot \boldsymbol{s} + \boldsymbol{e}]_q)_{i=1}^m\right) \end{array}\right] \\ - \Pr\left[b' = 1 : \begin{array}{c} \{\boldsymbol{a}_i\}_{i=1}^m \leftarrow R_q; \boldsymbol{u} \leftarrow R^\vee \\ b' \leftarrow \mathcal{A}\left((\boldsymbol{a}_i, \boldsymbol{u})_{i=1}^m\right) \end{array}\right] \end{array} \right| \tag{3.13b}$$

The $\mathsf{dRLWE}_{R_q,m,q,\chi}$ assumption is that $\mathsf{Adv}^{\mathsf{dRLWE}}_{R_q,m,q,\chi}(\mathcal{A}) < \mathsf{negl}(\lambda)$.

[2] The meaning of the acronym NTRU is somewhat mysterious. Plausible candidates include "N-th degree Truncated polynomial Ring Units" and "Number Theorists 'R' Us".

[3] Attention on symbol abuse: m denotes sample number in the context of LWE/RLWE, but it also used to denote the cyclotomic index in Section 3.2. Due to the fact that when cyclotomic index is 2's power, n is its half, so to avoid confusion we do not use m to denote cyclotomic index in the context of RLWE.

Remark 13 ($\langle \boldsymbol{s}, \boldsymbol{a}_i \rangle$ in LWE and $\boldsymbol{a}_i \cdot \boldsymbol{s}$ in RLWE)**:** $\langle \boldsymbol{s}, \boldsymbol{a}_i \rangle$ in LWE is the inner product of two vectors and results in a number, $\boldsymbol{a}_i \cdot \boldsymbol{s}$ in RLWE is the multiplication of two polynomials and results in a polynomial. Though they share a similarity of "an integer array with n elements", they are two different objects in nature.

3.4 Polynomial Arithmetics

RLWE works on polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. The complexity of addition/subtraction is $O(n)$ while the (naive) multiplication is $O(n^2)$. Clearly, as n increasing, polynomial multiplication becomes computationally expensive. *Number theoretic transform (NTT)* is a way to reduce the complexity to $O(n \log n)$. As shown in Figure 3.1, we will first discuss the Montgomery reduction (Section 3.4.1) and Barrett reduction (Section 3.4.2), two key techniques in the NTT algorithm (Section 3.4.3). They are basically adapted from [MKvOV96, ADPS16] and GPQHE.

On another hand, the RLWE module q is pretty large: for classical security level $\lambda = 128$ and a polynomial degree $n = 2^{14}$, the upper bound of q is $\log(q) = 438$. Such a large number is stored as *multi-precision integer (MPI)* in most of the cryptographic software, including OpenSSL and Libgcrypt in Linux kernel. MPI is not convenient for NTT operations, so in practice, two polynomials $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}_q$ in MPI representation are firstly converted to RNS representation $\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}}$ to perform NTT operation in a point-wise manner, then convert back to get the result in MPI representation. Section 3.4.4 will talk about how it works.

Polynomial rotation and conjugation both utilize the automorphism property, which will be explained in Section 3.4.5.

3.4.1 Montgomery Reduction

Montgomery reduction is a technique which allows efficient implementation of modular multiplication without explicitly carrying out the classical modular reduction "%", which involves trial division in assembly level.

To simplify our discussion, we focus on the portable C implementation. Let `u64` $\mathfrak{p}$ being a prime modulus and a `u128` Montgomery $R_{\text{mont}} = 2^{64}$. It's obviously that $R_{\text{mont}} > \mathfrak{p}$, $\mathsf{gcd}(\mathfrak{p}, R_{\text{mont}}) = 1$ and $R_{\text{mont}} - 1 =$ `U64_MAX` is exactly `u64`. Let `u128` t satisfying $0 \le t < \mathfrak{p} R_{\text{mont}}$. Montgomery reduction aiming to calculate $tR_{\text{mont}}^{-1} \bmod \mathfrak{p}$, *the reduction of* t *modulo* $\mathfrak{p}$ *w.r.t.* R_{mont} without using "%". The pre-computed quantity is $\mathfrak{p}_{\text{mont}}^{-1} = \mathsf{invm}(\mathfrak{p}, R_{\text{mont}})$ and its procedure is shown in Algorithm 1.

It's necessary to draw the readers' attention that there are two genres in the sign settings of Montgomery reduction: either $R_{\text{mont}} R'_{\text{mont}} - \mathfrak{p}\mathfrak{p}_{\text{mont}}^{-1} = 1$ defined in [Mon85], yielding $\mathfrak{p}_{\text{mont}}^{-1} = -\mathsf{invm}(\mathfrak{p}, R_{\text{mont}})$ (e.g., $12287 = -\mathsf{invm}(12289, 2^{18})$ in NewHope), or utilize the same sign convention in extended Euclidean $R_{\text{mont}} R'_{\text{mont}} + \mathfrak{p}\mathfrak{p}_{\text{mont}}^{-1} = 1$, yielding $\mathfrak{p}_{\text{mont}}^{-1} = \mathsf{invm}(\mathfrak{p}, R_{\text{mont}})$ (e.g., $62209 = \mathsf{invm}(3329, 2^{16})$ in Kyber, and HEAAN and GPQHE). In fact, both conventions yield correct results and do not affect the nature of Montgomery reduction.

Algorithm 1 Montgomery reduction

Input: u64 $\mathfrak{p}, \mathfrak{p}^{-1}_{\text{mont}}$, u128 $R_{\text{mont}}, t (t < \mathfrak{p}R_{\text{mont}})$
Output: u64 $tR^{-1}_{\text{mont}} \bmod \mathfrak{p}$

1: u64 $t_{\text{lo}} = t \wedge$ U64_MAX
2: u64 $t_{\text{hi}} = (tR^{-1}_{\text{mont}}) \wedge$ U64_MAX
3: u64 $u = (t_{\text{lo}}\mathfrak{p}^{-1}_{\text{mont}}) \wedge$ U64_MAX
4: u64 $v = ((u\mathfrak{p})R^{-1}_{\text{mont}}) \wedge$ U64_MAX
5: **return** $(t_{\text{hi}} < v)?\ t_{\text{hi}} - v + \mathfrak{p} : t_{\text{hi}} - v$

3.4.2 Barrett Reduction

Barrett reduction computes $r = a \bmod \mathfrak{p}$. It is advantageous if many reductions are performed with a single modulus (e.g. in RSA). The pre-computed quantity is $\mathfrak{p}^{-1}_{\text{barr}} = \lfloor 2^{2\mathsf{nbits}(\mathfrak{p})}/\mathfrak{p} \rfloor$ and its procedure is shown in Algorithm 2.

Algorithm 2 Barrett reduction

Input: u64 $\mathfrak{p}, \mathfrak{p}^{-1}_{\text{barr}}$, u128 a
Output: u64 $r = a \bmod \mathfrak{p}$

1: u64 $a_{\text{lo}} = a \wedge$ U64_MAX
2: u64 $a_{\text{hi}} = (t >> 64) \wedge$ U64_MAX
3: u128 $t = ((a_{\text{lo}}\mathfrak{p}^{-1}_{\text{barr}}) >> 64) | (a_{\text{hi}}\mathfrak{p}^{-1}_{\text{barr}})$
4: u64 $s = 2\mathsf{nbits}(\mathfrak{p}) - 64$
5: u64 $r = (a - \lfloor t/2^s \rfloor \mathfrak{p}) \wedge$ U64_MAX
6: **return** $(r < \mathfrak{p})?\ r : r - \mathfrak{p}$

Remark 14 (on Algorithm 1 and 2)**:** Any fields that involves multiplication of two u64 should firstly be cast to u128 to avoid overflow. Besides, due to $R_{\text{mont}} = 2^{64}$, any fields in the form aR^{-1}_{mont} is exactly right-shift $a >> 64$.

3.4.3 Polynomial Multiplication with NTT

Consider two polynomials $\boldsymbol{a}, \boldsymbol{b} \in R_{\mathfrak{p}} = \mathbb{Z}_{\mathfrak{p}}[x]/(x^n + 1)$ with $\{a_i\}_{i=0}^{n-1}, \{b_i\}_{i=0}^{n-1}$ as coefficients and prime modulus $\mathfrak{p}$. Their (schoolbook) multiplication is

$$\begin{aligned} \boldsymbol{a} \cdot \boldsymbol{b} &= \left(\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j x^{i+j} \bmod \mathfrak{p} \right) \bmod (x^n + 1) \\ &= \sum_{i=0}^{2n-2} (-1)^{\lfloor i/n \rfloor} \left(\sum_{j=0}^{i} a_j b_{i-j} \bmod \mathfrak{p} \right) x^{i \bmod n} \end{aligned} \tag{3.14}$$

Concretely, $(-1)^{\lfloor i/n \rfloor} = +1$ if $i < n$, and -1 otherwise. The complexity of (3.14) is $O(n^2)$, which makes it unfeasible to use for a typical $n = 2^{10}$ in RLWE.

NTT reduces the cost of multiplication to a quasi-linear complexity $O(n \log n)$. The procedure to obtain $\boldsymbol{a} \cdot \boldsymbol{b}$ is pretty similar to FFT and IFFT:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \mathsf{NTT}^{-1} \left(\mathsf{NTT}(\boldsymbol{a}) \circ \mathsf{NTT}(\boldsymbol{b}) \right) \tag{3.15}$$

NTT can indeed be regarded as a variant of FFT. In NTT, polynomials are working on a finite field $\mathbb{F}_\mathfrak{p} = \mathbb{Z}_\mathfrak{p}$ instead of in $\mathbb{C}$ in FFT. The *twiddle factor* $\omega_m = e^{2\pi i/m} (m = n)$ in FFT is replaced by the integer counterpart $\omega_m \in \mathbb{F}_\mathfrak{p}$, where m is the cyclotomic index of $R_\mathfrak{p} = \mathbb{Z}_\mathfrak{p}[x]/(x^n + 1)$ with $n = \phi(m) = m/2$. The NTT exists iff $\mathfrak{p} = 1 \bmod m$.

The forward NTT transforms $\boldsymbol{a}(x) = \sum_{i=0}^{n-1} a_i x^i \in \mathbb{Z}_\mathfrak{p}[x]$ to $\hat{\boldsymbol{a}}(X) = \sum_{i=0}^{n-1} \hat{a}_i X^i \in \mathbb{Z}_\mathfrak{p}[X]$. Their coefficients are related by (with $i = 0, \cdots, n-1$)

$$\hat{a}_i = \sum_{j=0}^{n-1} a_j \omega_m^{ij} \bmod \mathfrak{p} \qquad a_i = \frac{1}{n} \sum_{j=0}^{n-1} \hat{a}_j \omega_m^{-ij} \bmod \mathfrak{p} \tag{3.16a}$$

The the most time-consuming part is the modular multiplication appeared in the sum and Montgomery reduction is a weapon to accelerate it. Take $a_j \omega_m^{ij} \bmod \mathfrak{p}$ as an example. Let $\tilde{a}_j = a_j R_{\text{mont}} \bmod \mathfrak{p}$, $\tilde{\omega}_m^{ij} = \omega_m^{ij} R_{\text{mont}} \bmod \mathfrak{p}$ and $t = \tilde{a}_j \tilde{\omega}_m^{ij}$. The Montgomery reduction gives $t R_{\text{mont}}^{-1} \bmod \mathfrak{p} = a_j \omega_m^{ij} R_{\text{mont}} \bmod \mathfrak{p}$. Therefore, if we modify the coefficient relation in (3.16a) to

$$\hat{a}_i = \sum_{j=0}^{n-1} a_j \omega_m^{ij} R_{\text{mont}} \bmod \mathfrak{p} \qquad a_i = \frac{1}{n} \sum_{j=0}^{n-1} \hat{a}_j \omega_m^{-ij} R_{\text{mont}} \bmod \mathfrak{p} \tag{3.16b}$$

and let $t = a_j \omega_m^{ij} R_{\text{mont}}$, then $t R_{\text{mont}}^{-1} \bmod \mathfrak{p} = a_j \omega_m^{ij} \bmod \mathfrak{p}$. In summary, scaling the twiddle factor ω_m, ω_m^{-1} by R_{mont} enables us to utilize Montgomery reduction.

The *scaled* twiddle factor $\omega_m R_{\text{mont}}, \omega_m^{-1} R_{\text{mont}} \in \mathbb{F}_\mathfrak{p}$ is the quantity that required to be pre-computed. ω_m is nothing but the m-th root of unity in $\mathbb{F}_\mathfrak{p}$ that satisfying $(\omega_m)^m = 1 \bmod \mathfrak{p}$:

$$\omega_m = g^{(\mathfrak{p}-1)/m} \bmod \mathfrak{p}, \text{ with } g = \mathsf{generator}(\mathfrak{p}) \tag{3.17}$$

$\{[\omega_m^k]_\mathfrak{p}\}_{k \in \mathbb{Z}_m^\times}$ and $\{[\omega_m^{-k}]_\mathfrak{p}\}_{k \in \mathbb{Z}_m^\times}$ both gives $n = m/2$ values. The pre-computed quantities are stored in table, shown in Algorithm 3. The algorithm for NTT and NTT^{-1} are shown in Algorithm 4 and 5 respectively.

Algorithm 3 Pre-compute the R_{mont} scaled twiddle factor

Input: prime modulus $\mathfrak{p}$, cyclotomic index m, polynomial degree n
Output: $\mathsf{Table}_{\omega_m}$, $\mathsf{Table}_{\omega_m^{-1}}$

1: $\omega_m = g^{(\mathfrak{p}-1)/m} \bmod \mathfrak{p}$, with $g = \mathsf{generator}(\mathfrak{p})$
2: $\omega_m^{-1} = \mathsf{invm}(\omega_m, \mathfrak{p}) = \omega_m^{\mathfrak{p}-2} \bmod \mathfrak{p}$
3: **for** $i = 0$ to $n - 1$ **do**
4: $\quad j = \mathsf{bitrev}(i, n)$
5: $\quad \mathsf{Table}_{\omega_m}[j] = \omega_m^i R_{\text{mont}} \bmod \mathfrak{p}$
6: $\quad \mathsf{Table}_{\omega_m^{-1}}[j] = \omega_m^{-i} R_{\text{mont}} \bmod \mathfrak{p}$
7: **end for**

3.4.4 MPI-RNS Conversion

The modulus q for $\boldsymbol{a} \in R_q = \mathbb{Z}_q[x]/(x^n + 1)$ is stored as `MPI`, however, NTT is performed in *NTT domain* (a.k.a. *RNS* domain, there is no common name for this), with every coefficients are `u64`. Recall Section 3.2.5, there is a prime splitting of q. In this subsection, we will talk about how this conversion works.

Algorithm 4 Forward NTT transform

Input: $n, \{a_i\}_{i=0}^{n-1}, \mathsf{Table}_{\omega_m}, \mathfrak{p}, \mathfrak{p}_{\text{mont}}^{-1}$
Output: Coefficients $\{\hat{a}_i\}_{i=0}^{n-1}$, with $\hat{\boldsymbol{a}} = \mathsf{NTT}(\boldsymbol{a})$

$j = 0, k = 1, \mathsf{memcpy}(\hat{\boldsymbol{a}}, \boldsymbol{a}, n \cdot \mathsf{sizeof}(\mathsf{u64}))$
for $(l = \frac{n}{2}; l \geq 1; l >>= 1)$ **do** ▷ "l" means "length"
  for $(s = 0; s < n; s = j + l)$ **do** ▷ "s" means "start index"
    $[\omega_m^k R_{\text{mont}}]_{\mathfrak{p}} = \mathsf{Table}_{\omega_m}[k++]$
    for $(j = s; j < s + l; j++)$ **do**
      $t = \mathsf{montgomery\_reduce}(\hat{a}_{j+l}[\omega_m^k R_{\text{mont}}]_{\mathfrak{p}}, \mathfrak{p}, \mathfrak{p}_{\text{mont}}^{-1})$ ▷ i.e. $a_{j+l}\omega_m^k \bmod \mathfrak{p}$
      $\hat{a}_{j+l} = (\hat{a}_j \geq t)?\ \hat{a}_j - t : \hat{a}_j - t + \mathfrak{p}$
      $\hat{a}_j = (\hat{a}_j \leq \mathfrak{p} - t)?\ \hat{a}_j + t : \hat{a}_j + t - \mathfrak{p}$
    end for
  end for
end for

Algorithm 5 Inverse NTT transform

Input: $n, \{\hat{a}_i\}_{i=0}^{n-1}, \mathsf{Table}_{\omega_m^{-1}}, \mathfrak{p}, \mathfrak{p}_{\text{mont}}^{-1}$
Output: Coefficients $\{a_i\}_{i=0}^{n-1}$, with $\boldsymbol{a} = \mathsf{NTT}^{-1}(\hat{\boldsymbol{a}})$

$j = 0, \mathsf{memcpy}(\boldsymbol{a}, \hat{\boldsymbol{a}}, n \cdot \mathsf{sizeof}(\mathsf{u64}))$
for $(l = 1; l \leq \frac{n}{2}; l <<= 1)$ **do** ▷ "l" means "length"
  $k = \frac{n}{2l}$
  for $(s = 0; s < n; s = j + l)$ **do** ▷ "s" means "start index"
    $[\omega_m^{-k} R_{\text{mont}}]_{\mathfrak{p}} = \mathsf{Table}_{\omega_m^{-1}}[k++]$
    for $(j = s; j < s + l; j++)$ **do**
      $t = a_j$
      $a_j = (a_{j+l} \leq \mathfrak{p} - t)?\ t + a_{j+l} : t + a_{j+l} - \mathfrak{p}$
      $a_{j+l} = (a_{j+l} \leq t)?\ t - a_{j+l} : t - a_{j+l} + \mathfrak{p}$
      $a_{j+l} = \mathsf{montgomery\_reduce}(a_{j+l}[\omega_m^{-k} R_{\text{mont}}]_{\mathfrak{p}}, \mathfrak{p}, \mathfrak{p}_{\text{mont}}^{-1})$ ▷ i.e. $a_{j+l}\omega_m^{-k} \bmod \mathfrak{p}$
    end for
  end for
end for
for $i = 0$ to $n - 1$ **do**
  $a_i = \mathsf{montgomery\_reduce}(a_i n^{-1}, \mathfrak{p}, \mathfrak{p}_{\text{mont}}^{-1})$
end for

Consider $P = \prod_{k=1}^{\mathsf{dim}} \mathfrak{p}_k$ with $\mathfrak{p}_k \equiv 1 \bmod m$ and $m = 2n$ is cyclotomic index, "dim" is the dimension of the RNS space. Define $\texttt{MPI}\ \hat{\mathfrak{p}}_k = P/\mathfrak{p}_k = \prod_{j \in [\mathsf{dim}] \setminus k} \mathfrak{p}_j$, and $\mathfrak{b}_k = \hat{\mathfrak{p}}_k(\hat{\mathfrak{p}}_k^{-1} \bmod \mathfrak{p}_k)$. $\{\mathfrak{b}_k\}_{k=1}^{\mathsf{dim}}$ is so-called the RNS base.

Let $\boldsymbol{a}(x) = \sum_{i=0}^{n-1} a_i x^i$, the MPI representation, where $a_i \in \mathbb{Z}_q$. Define

$$\tilde{\boldsymbol{a}}^k(x) = \sum_{i=0}^{n-1} \tilde{a}_i^k x^i, \quad \tilde{a}_i^k = a_i \bmod \mathfrak{p}_k \in \mathbb{F}_{\mathfrak{p}_k} \tag{3.18}$$

for $k = 1, \cdots, \mathsf{dim}$ as the RNS representation of $\boldsymbol{a}(x)$. The $\mathtt{u64}$ $\tilde{a}_i^k$ can be understood as the projection of the polynomial coefficient a_i (in MPI representation) to the k-th RNS base $\mathfrak{b}_k$. $\{\tilde{a}_i^k\}_{k=1}^{\mathsf{dim}}$ and a_i is related by CRT:

$$a_i = \sum_{k=1}^{\mathsf{dim}} \tilde{a}_i^k \mathfrak{b}_k \bmod P \tag{3.19}$$

We turn to ask "what is the size of $\mathfrak{p}_k$" and "what is the proper value of dim"? There is no common standard about this. HEAAN imposes a lower bound of bits on $\{\mathfrak{p}_k\}_{k=1}^{\mathsf{dim}}$ — $\min \log \mathfrak{p}_k = 59$; while in SEAL, another HE library, the number of primes (i.e. the dim) and the minimum bits of each $\mathfrak{p}_k$ requires user-specified input and can smaller than 59. Though schemes are not unique, a valid choice should guarantee that $P = \prod_{k=1}^{\mathsf{dim}} \mathfrak{p}_k > q$ and each $\mathfrak{p}_k$ satisfies NTT existence condition $\mathfrak{p}_k \equiv 1 \bmod m$. Section 6.2.1 will discuss it in detail.

Instead of via schoolbook multiplication (3.14), with the help of NTT and RNS, the calculation of two polynomials $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b}$ can be decomposed into three steps. We firstly decompose the MPI-represented polynomial (with n MPI coefficients) to RNS-represented polynomial (with $\mathsf{dim} \times n$ $\mathtt{u64}$ coefficients), shown in Algorithm 6. Then, two RNS-represented polynomials are multiplied point-wise, shown in Algorithm 7. Finally, convert the result back to MPI domain, shown in Algorithm 8.

Due to the fact that NTT/INTT is a time consuming component, we would preferably store the RNS-represented polynomials $\{\hat{\boldsymbol{a}}^k(X)\}_{k=1}^{\mathsf{dim}}$ instead of storing the MPI-represented polynomials $\boldsymbol{a}(x)$. For example, the secret key and public key in CPAKEM, as well as the evaluation keys in homomorphic encryption, are stored in RNS representation.

Algorithm 6 Decompose a polynomial in MPI domain to RNS domain

Input: Polynomial $\boldsymbol{a}(x) = \sum_{i=0}^{n-1} a_i x^i$ in MPI domain
Output: $\{\hat{\boldsymbol{a}}^k(X)\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{a}(x))$ in RNS domain

1: **for** $k = 1$ to dim **do**
2: **for** $i = 0$ to $n - 1$ **do**
3: $\tilde{a}_i^k = a_i \bmod \mathfrak{p}_k$
4: **end for**
5: $\hat{\boldsymbol{a}}^k(X) = \mathsf{NTT}\left(\tilde{\boldsymbol{a}}^k(x) = \sum_{i=0}^{n-1} \tilde{a}_i^k x^i, \mathsf{Table}_{\omega_{m,k}}, \mathfrak{p}_k, \mathfrak{p}_{\mathrm{mont},k}^{-1}\right)$
6: **end for**
7: **return** $\{\hat{\boldsymbol{a}}^k(X) = \sum_{i=0}^{n-1} \hat{a}_i^k X^i\}_{k=1}^{\mathsf{dim}}$

Algorithm 7 Polynomial multiplication in RNS domain

Input: Polynomial $\{\hat{\boldsymbol{a}}^k(X)\}_{k=1}^{\mathsf{dim}}, \{\hat{\boldsymbol{b}}^k(X)\}_{k=1}^{\mathsf{dim}}$ in RNS domain
Output: $\{\hat{\boldsymbol{c}}^k(X)\}_{k=1}^{\mathsf{dim}} = \mathsf{poly\_rns\_mul}\left(\{\hat{\boldsymbol{a}}^k(X)\}_{k=1}^{\mathsf{dim}}, \{\hat{\boldsymbol{b}}^k(X)\}_{k=1}^{\mathsf{dim}}\right)$, also in RNS domain

1: **for** $k = 1$ to dim **do**
2: $\hat{\boldsymbol{c}}^k(X) = \hat{\boldsymbol{a}}^k(X) \circ \hat{\boldsymbol{b}}^k(X) = \sum_{i=0}^{n-1} \mathsf{barrett\_reduce}(\hat{a}_i^k \hat{b}_i^k, \mathfrak{p}_k, \mathfrak{p}_{\mathrm{barr,k}}^{-1}) X^i$
3: **end for**
4: **return** $\{\hat{\boldsymbol{c}}^k(X) = \sum_{i=0}^{n-1} \hat{c}_i^k X^i\}_{k=1}^{\mathsf{dim}}$ with $\hat{c}_i = \hat{a}_i^k \hat{b}_i^k$

Algorithm 8 Polynomial multiplication

Input: Polynomials $\boldsymbol{a}(x), \boldsymbol{b}(x) \in R_q$
Output: $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b} \in R_q$
1: **for** $k = 1$ to dim **do**
2: $\quad \tilde{\boldsymbol{c}}^k(x) = \mathsf{INTT}\left(\hat{\boldsymbol{c}}^k(X), \mathsf{Table}_{\omega_{m,k}^{-1}}, \mathfrak{p}_k, \mathfrak{p}_{\mathrm{mont,k}}^{-1}\right) = \sum_{i=0}^{n-1} \tilde{c}_i^k x^i$
3: **end for**
4: **for** $i = 0$ to $n-1$ **do**
5: $\quad c_i = \sum_{k=1}^{\mathsf{dim}} \tilde{c}_i^k \mathfrak{b}_k \bmod P$
6: **end for**
7: **return** $\boldsymbol{c} = \sum_{i=0}^{n-1} c_i x^i = \boldsymbol{a} \cdot \boldsymbol{b}$ obtained.

3.4.5 Polynomial Rotation and Conjugation

Recall Section 3.2.3, there are $n = \phi(m)$ automorphisms $\tau_k : \mathbb{K} \to \mathbb{K}$, $k \in \mathbb{Z}_m^*$, defined by $\tau_k : \boldsymbol{a}(x) \mapsto \boldsymbol{a}(x^k)/\Phi_m(x)$ for a polynomial $\boldsymbol{f} \in \mathbb{K}$. Again consider the 2's power cyclotomic ring $R = \mathbb{Z}[x]/(x^n+1)$ and a polynomial $\boldsymbol{a}(x) = \sum_{i=0}^{n-1} a_i x^i \in R$, this automorphism is then $\tau_k : \boldsymbol{a}(x) \mapsto \boldsymbol{a}(x^k)/(x^n+1)$. Typical examples are rotation (3.20) and conjugation (3.21). As will be seen in Section 4.5.5, both of them are basic operations in homomorphic evaluation.

$$\tau_k(\boldsymbol{a}) = \sum_{i=0}^{n-1} a_i x^{i+k}/(x^n+1) = \sum_{i=k}^{n-1} a_{i-k} x^i - \sum_{i=0}^{k-1} a_{n-k+i} x^i \tag{3.20}$$

$$\tau_{-1}(\boldsymbol{a}) = \sum_{i=0}^{n-1} a_i x^{-i}/(x^n+1) = a_0 - \sum_{i=1}^{n-1} a_{n-i} x^i \tag{3.21}$$

3.5 Random Distribution

There are several different distributions used in RLWE. Here are the discussions.

3.5.1 Gaussian Measure and Discrete Gaussian Distribution

The "error" in RLWE is a vector of small numbers in $\mathbb{H}$ which can be regarded as the "clouds" around each lattice point. These "clouds" would "blur" the exact lattice point, which makes it infeasible to obtain the lattice coordinates.

For $r > 0$, we define the Gaussian function $\rho_r : \mathbb{H} \to (0,1]$ as $\rho_r = \exp(-\pi \|\boldsymbol{x}\|_2^2 / r^2)$ with $\boldsymbol{x} \in \mathbb{H}$. By normalizing ρ_r we obtain the *continuous* Gaussian probability distribution $\Gamma_r(\boldsymbol{x}) = r^{-n}\rho_r(\boldsymbol{x})$. Γ_r is extended to elliptical (non-spherical) Gaussian distribution in the basis $\{\mathbf{b}_i\}_{i\in[n]}$ as follows: let $\boldsymbol{r} = (r_1, \cdots, r_n) \in \mathbb{R}_+^n$ with $r_{s_1+i} = r_{n-i}$ for $i = 1, \cdots, s_2$ (in the case of cyclotomic field, $s_1 = 0$ and $s_2 = n/2$), then a sample from $\Gamma_{\boldsymbol{r}}$ is given by $\sum_{i\in[n]} x_i \mathbf{b}_i$ with each x_i chosen independently from the one-dimensional Gaussian Γ_{r_i} over $\mathbb{R}$, i.e. $x_i \leftarrow \Gamma_{r_i}(\mathbb{R})$.

Recall that $\mathbb{K}$ is a Galois extension to $\mathbb{Q}$, but $\Gamma_{\boldsymbol{r}}$ is a distribution over $\mathbb{R}$; in other words, $\Gamma_{\boldsymbol{r}}$ is to $\mathbb{K}$ as if $\mathbb{R}$ is to $\mathbb{Q}$. So we let $\mathbb{K}_{\mathbb{R}} := \mathbb{K} \otimes_{\mathbb{Q}} \mathbb{R}$ and a shifted $\Gamma_{\boldsymbol{r}}$ denoted by ψ, over $\mathbb{K}_{\mathbb{R}}$.

In the original RLWE definition [LPR10], the error is sampled from ψ, which is continuous Gaussian. In practice, the error is sampled from a *discrete* Gaussian $\chi := \lfloor \psi \rceil_{R^\vee}$. To this end, we firstly generate a continuous Gaussian, then round it to the nearest integer to get the discrete Gaussian: $\boldsymbol{e} \leftarrow \mathcal{D}_\sigma$.

The *Box-Muller transform* is a de facto standard for generating *continuous* Gaussian distribution $\exp(-\|\boldsymbol{x}\|_2^2/2\sigma^2)$. Algorithm 9 use Box-Muller transform to generate *discrete* Gaussian numbers by rounding down, which is implemented in HEAAN. According to the HE standard [ACC+18], the standard deviation σ should be chosen as $8/\sqrt{2\pi} \approx 3.2$.

Algorithm 9 Box-Muller transform to generate discrete Gaussian distribution

Input: The size of the error vector $n(=2^k, k \in \mathbb{N})$; standard deviation σ
Output: The error vector $\mathbf{e} = (e_0, e_1, \cdots, e_{n-1})$

1: **for** $i = 0$ to $n-1$, $n+=2$ **do**
2: Sample $(\theta_s, r_s) \leftarrow [0,1)^2$
3: $\theta = 2\pi\theta_s$, $r = \sqrt{-2\ln(r_s)}\sigma$
4: $e_i = \lfloor r\cos\theta \rceil$, $e_{i+1} = \lfloor r\sin\theta \rceil$
5: **end for**

However, starting from the *continuous* Gaussian then rounding the generated number to the nearest integer (line 4) does not actually yield a true *discrete* Gaussian, c.f. [Pei16] Section 4.2.1 footnote 4. Furthermore, although this algorithm can produce a Gaussian-like distribution, it still requires to calculate the nonlinear elementary functions like $\ln, \cos, \sin$, which is vulnerable under *side-channel attack (SCA)*. Actually, calculating the exponent in Gaussian inherently increase the risk of being SCA. To reduce the computation cost induced by discrete Gaussian has been an important subproblem in PQC.

3.5.2 Centered-Binomial Distribution

A replacement to discrete Gaussian distribution is the *centered-binomial distribution (CBD)*, denoted by $\mathtt{cbd}(k)$: first take two random $\{0,1\}^k$ bit-strings, then calculate their *Hamming distance*. Algorithm 10 shows an implementation of obtaining the *hamming weight* of an `u8` number using divide-and-conquer method. Since NewHope in 2016, most RLWE/MLWE-based algorithms utilize CBD instead of the discrete Gaussian. [ADPS16] provides an analysis that such a substitution does not dramatically decrease the security of the scheme, and what's more, need not calculate the exponential, which decrease the risk of being SCA.

Algorithm 10 Hamming weight of an unsigned char (`u8`) number a

Input: `u8` a
Output: `u8` $\mathtt{hw}(a)$

1: $a- = (a >> 1) \wedge \mathtt{0x55}$ ▷ `0x55` = `0b01010101`
2: $a = (a \wedge \mathtt{0x33}) + ((a >> 2) \wedge \mathtt{0x33})$ ▷ `0x33` = `0b00110011`
3: **return** $(a + (a >> 4)) \wedge \mathtt{0x0f}$ ▷ `0x0f` = `0b00001111`

3.5.3 Hamming Weight Distribution

Drawing elements from a *ternary* $\{-1, 0, 1\}$ and return an array with the total counts of the nonzero entries (i.e. ± 1) being h is so-called *Hamming weight distribution*, denoted by $\mathsf{hwt}(h)$. The secret key in HEAAN is sampled from $\mathsf{hwt}(h = 64)$. It's implementation is shown in Algorithm 11.

Algorithm 11 Generate a vector with Hamming weight h: $\boldsymbol{s} \leftarrow \mathsf{hwt}(h)$

Input: The size of the vector $n(= 2^k, k \in \mathbb{N})$; hamming weight h
Output: The vector $\boldsymbol{s} = (s_0, s_1, \cdots, s_{n-1})$ with $\{s_i\}_{i=0}^{n-1} \in \{-1, 0, 1\}$

Initialize $\boldsymbol{s} = 0^n, t \leftarrow \{0,1\}^h$
$\tilde{h} = 0$ ▷ total nonzero entries count
while $\tilde{h} < h$ **do**
    $i \leftarrow \{0,1\}^{\leq \log n}$
    if $s_i == 0$ **then**
        $s_i = (\mathsf{tbit}(t, \tilde{h}) == 0)? + 1 : -1$
        $\tilde{h} + +$
    end if
end while

3.5.4 Zero-Center Distribution

Zero-center distribution is characterized by a real parameter $\rho \in [0, 1]$ and denoted by $\mathsf{zo}(\rho)$, which draws each entry in the vector from the ternary $\{-1, 0, 1\}^n$ with probability $\rho/2$ for -1 and $+1$, and $1-\rho$ for 0. GPQHE and HEAAN use $\rho = 0.5$. The implementation of $\mathsf{zo}(\rho = 0.5)$ is shown in Algorithm 12.

Algorithm 12 Generate a vector with $(\rho = 0.5)$

Input: The size of the vector $n(= 2^k, k \in \mathbb{N})$; parameter $\rho = 0.5$
Output: The vector $\boldsymbol{v} = (v_0, v_1, \cdots, v_{n-1})$ with $\{v_i\}_{i=0}^{n-1} \in \{-1, 0, 1\}$

$t \leftarrow \{0,1\}^{2n}$
for $i = 0$ to $n - 1$ **do**
    $v_i = (\mathsf{tbit}(t, 2i) == 0)?0 : (\mathsf{tbit}(t, 2i + 1) == 0)? + 1 : -1$
end for

3.6 CPA Key Encapsulation Mechanism

We consider the high-level implementation of RLWE problem $\mathsf{RLWE}_{R_q,n,q,\chi}$, where $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ is the cyclotomic ring, implemented in GPQHE based on HEAAN. The CPA-secured *key encapsulation mechanism (KEM)* is shown in Algorithm 13, 14, 15, 16.

Algorithm 13 Key generation

Output: $\mathsf{sk}, \mathsf{pk} = \mathsf{keypair}(q)$

1: $\mathsf{sk} \leftarrow \mathsf{hwt}(h = 64)$
2: $\mathsf{e} \leftarrow \mathcal{D}_\sigma, \boldsymbol{a} \leftarrow R_q$
3: $\boldsymbol{b} = \left[-\boldsymbol{a} \cdot \mathsf{sk} + \boldsymbol{e}\right]_q^{\pm}$
4: $\{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{a}), \{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{b})$
5: $\mathsf{pk} = \left(\{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}}, \{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}}\right)$
6: **return** sk, pk

Algorithm 14 Public key encryption

Input: pt, pk

Output: $\mathsf{ct} = \mathsf{enc}_{\mathsf{pk}}(\mathsf{pt}, \mathsf{pk}, q)$

1: $\boldsymbol{v} \leftarrow \mathsf{zo}(\rho = 0.5), \boldsymbol{e}_0 \leftarrow \mathcal{D}_\sigma, \boldsymbol{e}_1 \leftarrow \mathcal{D}_\sigma$
2: $\boldsymbol{b} = \left[\boldsymbol{v} \cdot \mathsf{pk}.\{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}} + \mathsf{pt} + \boldsymbol{e}_0\right]_q^{\pm}$
3: $\boldsymbol{a} = \left[\boldsymbol{v} \cdot \mathsf{pk}.\{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}} + \boldsymbol{e}_1\right]_q^{\pm}$
4: $\{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{a}), \{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{b})$
5: **return** $\mathsf{ct} = \left(\{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}}, \{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}}\right)$

Algorithm 15 Secret key encryption

Input: pt, sk

Output: $\mathsf{ct} = \mathsf{enc}_{\mathsf{sk}}(\mathsf{pt}, \mathsf{sk}, q)$

1: $\boldsymbol{e} \leftarrow \mathcal{D}_\sigma, \boldsymbol{a} \leftarrow R_q$
2: $\boldsymbol{b} = \left[-\boldsymbol{a} \cdot \mathsf{sk} + \mathsf{pt} + \boldsymbol{e}\right]_q^{\pm}$
3: $\{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{a}), \{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{b})$
4: **return** $\mathsf{ct} = \left(\{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}}, \{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}}\right)$

Algorithm 16 Decryption

Input: ct, sk

Output: $\mathsf{pt} = \mathsf{dec}(\mathsf{ct}, \mathsf{sk}, q)$

1: **return** $\mathsf{pt} = \left[\mathsf{ct}.\{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}} \cdot \mathsf{sk} + \mathsf{ct}.\{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}}\right]_q^{\pm}$

4 Leveled Homomorphic Encryption for Approximate Numbers

4.1 Introduction

It has been well studied that RLWE cryptosystems can be used to construct fully homomorphic encryption schemes [Reg10]. Before C.Gentry's initial construction [Gen09a, Gen09b] there are several cryptosystems that support homomorphic operations: Paillier [Pai99] supports unlimited addition homomorphic operations, while RSA and ElGamal support unlimited multiplication homomorphic operation. Both of them are categorized into *partial homomorphic encryption*, in contrast to C.Gentry's scheme.

C.Gentry's construction is commonly called *fully homomorphic encryption (FHE)*. However, in his construction, *bootstrap* is a separate procedure to support *unlimited* add/mult operations when the modulus of the ring is not large enough to correctly decrypt. In other words, informally, the constructions yet supporting add/mult homomorphic but not including the bootstrap procedure (i.e. "unlimited operations" is not supported) is not "fully". Such constructions are further categorized into two types: if the ciphertext size increases substantially with each homomorphic operation, this construction is called *somewhat homomorphic encryption (SHE)*; the variant that *restrict* the ciphertext size using an additional parameter "*level*" or "*depth*" is called *leveled homomorphic encryption (LHE)*. FHE can be informally regarded as LHE plus bootstrap procedure. This is a potentially ambiguous point in the HE-related literature.

The "level" or "depth" implies that the permitted homomorphic evaluation is limited by this parameter. The early (before 2017) HE schemes include but not limited to BGV [BGV12], which uses extension ring as plaintext space, and B/FV [FV12], which uses modulus integers as plaintext space. [CKKS17] pointed out that, due to the inherent plaintext modulus t in BGV and B/FV, people must use exponentially large ciphertext modulus q with the depth of an evaluation circuit to protect the homomorphic multiplied ciphertext. This dilemma arises especially in performing homomorphic evaluation to floating point numbers. Therefore, the early schemes are limited in performing HE operations to fixed point numbers.

Motivated by real-world applications such as the cyber-physical system, in which the *least significant bit (LSB)* of the detected signal is usually unimportant ([CHK+18]), CKKS [CKKS17] proposed a LHE scheme for approximated numbers. They abandoned the plaintext modulus t and instead use a rescale factor Δ (a large floating-point number) to replace $\lfloor q/t \rfloor$ in B/FV, and finally yield a decryption structure $\langle \boldsymbol{c}, \mathsf{sk} \rangle = [\boldsymbol{m} + \boldsymbol{e}]_q$. Though $\boldsymbol{e}$ is conventionally interpreted as encrypted "noise" in LWE/RLWE, they instead treat $\boldsymbol{e}$ as

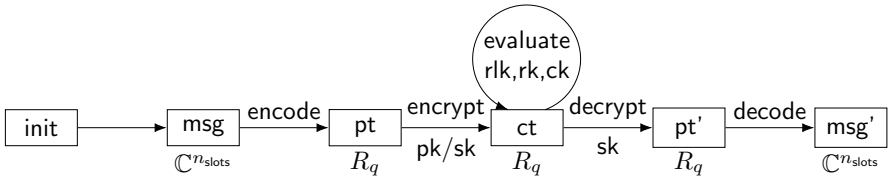

Figure 4.1: Workflow of homomorphic encryption.

Table 4.1: Architecture and supported functionalities of GPQHE.

	Used for ...	Keys or Functions
Advanced	linear transformation	$\mathsf{gemv}(\boldsymbol{A}, \mathsf{ct}, \mathsf{rk})$, $\mathsf{sum}(\mathsf{ct}, \mathsf{rk})$, $\mathsf{idx}(\mathsf{ct}, i, \mathsf{rk})$ $\mathsf{nrm22}(\mathsf{ct}, \mathsf{rlk}, \mathsf{rk}, \mathsf{ck})$
	nonlinear functions	$\exp(\mathsf{ct}, \mathsf{rlk})$, $\ln(\mathsf{ct}, \mathsf{rlk})$, $\mathsf{sigmoid}(\mathsf{ct}, \mathsf{rlk})$
	Comparison	$\mathsf{inv}(\mathsf{ct}, \mathsf{rlk}, d)$, $\mathsf{sqrt}(\mathsf{ct}, \mathsf{rlk}, d)$, $\mathsf{cmp}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rlk}, d, \alpha)$
Primitives	addition	$\mathsf{add}_{\mathsf{pt}}(\mathsf{ct}, \mathsf{pt})$, $\mathsf{add}(\mathsf{ct}_1, \mathsf{ct}_2)$, $\mathsf{neg}(\mathsf{ct})$ $\mathsf{sub}_{\mathsf{pt}}(\mathsf{ct}, \mathsf{pt})$, $\mathsf{sub}(\mathsf{ct}_1, \mathsf{ct}_2)$
	level switch	$\mathsf{rs}_{\ell\to\ell'}(\mathsf{ct})$, $\mathsf{moddown}_{\ell\to\ell'}(\mathsf{ct})$
	multiplication	$\mathsf{mult}_{\mathsf{pt}}(\mathsf{ct}, \mathsf{pt})$, $\mathsf{mult}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rlk})$
	automorphism	$\mathsf{rot}(\mathsf{ct}, r, \mathsf{rk})$, $\mathsf{conj}(\mathsf{ct}, \mathsf{ck})$
Encryptor	encrypt/decrypt	$\mathsf{ct} = \mathsf{enc}_{\mathsf{pk/sk}}(\mathsf{pt}, \mathsf{pk/sk})$, $\mathsf{pt} = \mathsf{dec}(\mathsf{ct}, \mathsf{sk})$
Encoder	encode/decode	$\mathsf{pt} = \mathsf{ecd}(\boldsymbol{z}, \Delta)$, $\boldsymbol{z} = \mathsf{dcd}(\mathsf{pt}, \Delta)$
KEM	public key, secret key	$\mathsf{pk}, \mathsf{sk} = \mathsf{keypair}(\mathsf{ctx})$
	evaluation keys	$\mathsf{rlk} = \mathsf{genswk}(\mathsf{sk} \cdot \mathsf{sk}, \mathsf{sk})$ $\mathsf{rk}_r = \mathsf{genswk}(\tau_{5^r \bmod 2n}(\mathsf{sk}), \mathsf{sk}), 0 \le r < \cdots n_{\text{slots}}$ $\mathsf{ck} = \mathsf{genswk}(\tau_{-1}(\mathsf{sk}), \mathsf{sk})$
mem_mgr	alloc/free context	$\mathsf{ctx} = \mathsf{init}(\lambda, n, q, n_{\text{slots}}, \Delta, h, \sigma)$
	alloc/free keys	pk,sk,evk
	alloc/free objects	ciphertext $(\mathsf{ct}, \ell, \nu, B)$, plaintext $(\mathsf{pt}, \|\mathsf{pt}\|_\infty^{\mathsf{can}})$

part of "LSB error" occurred during approximate computations or signal measurements. By removing the LSB, they prevent the exponential growth of the ciphertext modulus during multiplication. So that in the decoding procedure, $\boldsymbol{m}' = \boldsymbol{m} + \boldsymbol{e}$ is regarded as a "correct" result. This improvement is almost optimal in the view of precision: the resulting message is $\sim 10^{-14}$ to 10^{-10} deviated to the unencrypted floating-point case.

GPQHE does not implement the bootstrap procedure, so our discussion is limited to LHE in this project. Figure 4.1 shows the general workflow of a typical HE and Table 4.1 shows the architecture of GPQHE. This chapter explains how a complete LHE scheme works in GPQHE as well as its supported functions. Section 4.2 explains the parameters required to initialize the HE system. Section 4.3 explains the encode/decode procedure. Section 4.4 explains how a series of keys are generated in HE — this is commonly called KEM. Section 4.5 covers the elementary evaluations that would be frequently used in advanced functionalities. These elementary operations are commonly called *primitives*. Section 4.6 covers the advanced HE evaluations supported in GPQHE.

4.2 Initialize and Precompute

Before entering the work flow, we would firstly initialize the whole system with a series of parameters. The term *context* (with "ctx" in short) is used to denote the parameter settings of the cryptosystem. In GPQHE, the context of the CKKS HE system is denoted by

$$\mathsf{ctx} = \mathsf{init}(\lambda, n, q, n_{\text{slots}}, \Delta, h = 64, \sigma = 3.2) \tag{4.1}$$

The security level $\lambda(= 128, 192, 256)$ corresponds to either classical or quantum settings. We implicitly choose the 2's power polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, so that use n as one of the parameter to denote the polynomial degree. q denotes the modulus as that in RLWE problem explained in the last chapter. λ, n, q should satisfy the security constraints described in [ACC+18].

n_{slots} denotes the message length; it's more commonly called *slots*. $n_{\text{slots}} \leq n/2$ and still required to be 2's power, i.e. it is required to be $n_{\text{slots}} | \frac{n}{2}$. Δ will serve as the *scaling factor* in encode/decode procedure as well as the *rescaling factor* $\mathfrak{p} = \Delta$. h is the hamming weight of sk; σ is the standard deviation of the discrete Gaussian, not refers to the canonical embedding.

Let $q_\ell = q_0 \mathfrak{p}^l$ for $0 < l \leq L$. q_0 and L are actually parametrized by $q_L = q$ (the input modulus) and $\mathfrak{p} = \Delta$. To get them we start from $q = q_L$ and divide $\mathfrak{p}$ iteratively, yielding $q_{L-1} = q_L/\mathfrak{p}, q_{L-2} = q_L/\mathfrak{p}^2, \cdots$ until $q_0 < \mathfrak{p}$, which is no longer able to divide one more $\mathfrak{p}$.

The maximum module shall appear in the HE operation is q^2, appeared in the *key-switching* procedure. With the method described in Section 3.4.4, we search $\mathfrak{p}_k$ that satisfying $\mathfrak{p}_k \equiv 1 \bmod 2n$, and iteratively multiply $\mathfrak{p}_k$ until $P = \prod_{k=1}^{\mathsf{dim}} \mathfrak{p}_k > q^2$. With $\{\mathfrak{p}_k\}_{k=1}^{\mathsf{dim}}$ we compute the quantities (e.g. the Table of ω for NTT and RNS basis $\{\mathfrak{b}_k\}_{k=1}^{\mathsf{dim}}$) discussed in Section 3.4.

4.3 Encode and Decode

Encode and decode are not "encryption algorithm". In the Section 3.6 Algorithm 14 and 15, the inputted plaintext $\mathsf{pt} = \boldsymbol{a}(x) \in R_q$ *can be encoded from any information*, for example, string text, `u32` integer vector, `u64` integer vector, `double`/`float` type vector, or a 32-byte hash of a file. In CKKS's application scenarios, pt is encoded from a `double complex` vector $\boldsymbol{z} \in \mathbb{C}^{n_{\text{slots}}}$. The decode procedure convert pt back to $\boldsymbol{z}$.

We will talk about the compact packing and sparsely packing method in encoding. Compact packing is naturally related to canonical embedding, in which $n_{\text{slots}} = \frac{n}{2}$; sparsely packing is a derivative method of compact packing, cleverly utilize the ring automorphism, which allows us to encode a short message comparing to the polynomial degree n.

4.3.1 Compact Packing

Recall Section 3.2.3 (3.6), let $\boldsymbol{a} \in R$ denote the plaintext pt where R is a cyclotomic ring with index of 2's power, the canonical embedding $\sigma(\boldsymbol{a}) \in \mathbb{H}$ brings the polynomial $\boldsymbol{a}$ with coefficients in $\mathbb{Z}$ to a vector in $\mathbb{H} \subseteq \mathbb{C}^n$ with coefficients in $\mathbb{R}$, where $n = 2s_2$, represented the complex roots and their conjugate, and $n = \phi(m) = m/2$, m being the 2's power cyclotomic index. This mapping is done by defining the *CRT matrix for* m [DPSZ12], i.e. the *Vandermonde matrix* over the complex primitive m-th roots of unity

$$\mathsf{CRT}_m = \begin{pmatrix} \boldsymbol{U}_{\frac{n}{2}\times n} \\ \bar{\boldsymbol{U}}_{\frac{n}{2}\times n} \end{pmatrix}, \text{ with } \boldsymbol{U}_{\frac{n}{2}\times n} = \begin{pmatrix} 1 & \zeta_0 & \zeta_0^2 & \cdots & \zeta_0^{n-1} \\ 1 & \zeta_1 & \zeta_1^2 & \cdots & \zeta_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{\frac{n}{2}-1} & \zeta_{\frac{n}{2}-1}^2 & \cdots & \zeta_{\frac{n}{2}-1}^{n-1} \end{pmatrix} \tag{4.2}$$

and

$$\zeta_0 = e^{2\pi i/m}, \zeta_j = \zeta_0^{5^j \bmod m}, 0 \le j < \tfrac{n}{2} \tag{4.3}$$

Hence $\{\zeta_j, \bar{\zeta}_j : 0 \le j < \frac{n}{2}\}$ forms the set of the primitive m-th roots of unity [CHK+18], and $|\{\zeta_j, \bar{\zeta}_j : 0 \le j < \frac{n}{2}\}| = n = |\mathbb{Z}_m^\times|$. Reason for this choice of number 5 has explained in Section 3.2.3: it generates a cyclic group of order $n/2$, which can exactly span the index range. The inverse of CRT_m is

$$\mathsf{CRT}_m^{-1} = \frac{1}{n}\overline{\mathsf{CRT}_m}^\top = \frac{1}{n}\left[\bar{\boldsymbol{U}}_{n\times\frac{n}{2}}^\top, \boldsymbol{U}_{n\times\frac{n}{2}}^\top\right] \tag{4.4}$$

So that the canonical embedding and its inverse is given by

$$\mathbb{H} \ni \begin{bmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{bmatrix} = \sigma(\boldsymbol{a}) = \mathsf{CRT}_m \boldsymbol{a}, \quad R \ni \boldsymbol{a} = \sigma^{-1}\left(\begin{bmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{bmatrix}\right) = \mathsf{CRT}_m^{-1}\begin{bmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{bmatrix} \tag{4.5}$$

$\left[\begin{smallmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{smallmatrix}\right]$ is packed in the way described in Section 3.2.3, intuitively, $\{z_i = \overline{z_{n-i}}, 1 \le i \le \frac{n}{2}\}$. Such packing extracts the real part and imaginary part of $\boldsymbol{z}$. An alternative packing is $\left[\begin{smallmatrix} \mathrm{Re}\boldsymbol{z} \\ \mathrm{Im}\boldsymbol{z} \end{smallmatrix}\right]$, but in this case, the canonical embedding should change accordingly. No matter in any way, either $\left[\begin{smallmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{smallmatrix}\right]$ or $\left[\begin{smallmatrix} \mathrm{Re}\boldsymbol{z} \\ \mathrm{Im}\boldsymbol{z} \end{smallmatrix}\right]$, we can always encode a complex vector $\boldsymbol{z} \in \mathbb{C}^{n/2}$ to a polynomial $\boldsymbol{a} \in R$ with integer coefficients.

As a higher level interface, where the user only need to input the vector $\boldsymbol{z}$ but need not know the concrete packing/encoding scheme, we use a *natural isomorphism* π to pack $\boldsymbol{z}$ to $\left[\begin{smallmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{smallmatrix}\right]$. Such a mapping could be $\frac{1}{\sqrt{2}}\left(\begin{smallmatrix} \boldsymbol{I}_{n/2} & \boldsymbol{J}_{n/2} \\ i\boldsymbol{J}_{n/2} & -i\boldsymbol{I}_{n/2} \end{smallmatrix}\right)$ defined in [LPR10] or $\frac{1}{\sqrt{2}}\left(\begin{smallmatrix} \boldsymbol{I}_{n/2} & i\boldsymbol{J}_{n/2} \\ \boldsymbol{J}_{n/2} & -i\boldsymbol{I}_{n/2} \end{smallmatrix}\right)$ defined in [LPR13] and [CKKS17]. The natural isomorphism could be denoted by

$$\boldsymbol{z} = \pi\left(\begin{bmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{bmatrix}\right), \quad \pi^{-1}(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{z} \\ \bar{\boldsymbol{z}} \end{bmatrix} \tag{4.6}$$

In CKKS's construction, π and σ together play a role of *encoder*: $\pi \circ \sigma$ means *decode* and $\sigma^{-1} \circ \pi^{-1}$ means *encode*. However, note that elements in R is integer, while in $\mathbb{H}$ is real, so that σ mapping would implicitly taking a "rounding to the nearest integer" step. This cause a problem that if two input numbers differ in < 1, they would be treated as the same

integer. To avoid this problem we multiply a *scaling factor* Δ to $\pi^{-1}(\boldsymbol{z})$ then taking rounding: $\lfloor \Delta \cdot \pi^{-1}(\boldsymbol{z}) \rceil_{\sigma(R)}$.

In summary, the plaintext polynomial $\mathsf{pt} = \boldsymbol{a} \in R$ obtained from a `double complex` message vector $\boldsymbol{z} \in \mathbb{C}^{n/2}$ is

$$\boldsymbol{a} = \mathsf{ecd}(\boldsymbol{z}, \Delta) = \sigma^{-1}\left(\lfloor \Delta \cdot \pi^{-1}(\boldsymbol{z}) \rceil_{\sigma(R)}\right) = \frac{1}{n}\left(\bar{\boldsymbol{U}}^{\top} \lfloor \Delta \cdot \boldsymbol{z} \rceil + \boldsymbol{U}^{\top} \lfloor \Delta \cdot \bar{\boldsymbol{z}} \rceil\right) \in R \tag{4.7}$$

and to decode $\mathsf{pt} \in R$ to the final `double complex` message is

$$\boldsymbol{z} = \mathsf{dcd}(\boldsymbol{a}, \Delta) = \tfrac{1}{\Delta} \boldsymbol{U}_{\frac{n}{2} \times n} \boldsymbol{a} \in \mathbb{C}^{\frac{n}{2}}, \text{ component-wise: } \{z_i = \tfrac{1}{\Delta} \boldsymbol{a}(\zeta_i)\}_{i=0}^{\frac{n}{2}-1} \tag{4.8}$$

which is nothing but the evaluation of $\boldsymbol{a}(x)$ at $\{\zeta_j\}_{j=0}^{n/2-1}$, then scaling down. The whole procedure of CKKS's encode-decode scheme is shown as below:

$$\begin{array}{ccccccc} \mathbb{C}^{n/2} & \xrightarrow{\pi^{-1}} & \mathbb{H} & \xrightarrow{\lfloor \cdot \rceil_{\sigma(R)}} & \sigma(R) & \xrightarrow{\sigma^{-1}} & R \\ \boldsymbol{z} & \longmapsto & \pi^{-1}(\boldsymbol{z}) & \longmapsto & \lfloor \Delta \cdot \pi^{-1}(\boldsymbol{z}) \rceil_{\sigma(R)} & \longmapsto & \sigma^{-1}\left(\lfloor \Delta \cdot \pi^{-1}(\boldsymbol{z}) \rceil_{\sigma(R)}\right) \end{array} \tag{4.9}$$

4.3.2 Sparsely Packing

In the last subsection we discussed the special case of $n_{\text{slots}} = \frac{n}{2}$. In practice, $\log n$ ranges from 10 to 16, however the message length (the "slots") are not so large. So this require a sparsely packing method. Let $\tilde{n} | n$ and $n_{\text{slots}} = \frac{\tilde{n}}{2} \leq \frac{n}{2}$. Recall the cyclotomic ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. We define a new variable $y = x^{n/\tilde{n}}$, the corresponding ring $R'_q = \mathbb{Z}_q[y]/(y^{\tilde{n}} + 1)$ is a subring of R_q. The cyclotomic index of R'_q is then $2\tilde{n}$. The encode-decode relation still holds in R'_q. Recall that the CRT matrix for m is characterized by the m-th root of unity $\zeta_0 = e^{2\pi i/m}$; in R'_q, it becomes

$$\xi_0 = \zeta_0^{n/\tilde{n}}, \xi_j = \xi_0^{5^j \bmod 2\tilde{n}}, 0 \leq j < \tfrac{\tilde{n}}{2} \tag{4.10}$$

Consider a plaintext polynomial $\boldsymbol{a} \in R_q$ and $\tilde{\boldsymbol{a}} \in R'_q$: they are the encoded plaintext from vectors $\boldsymbol{z} \in \mathbb{C}^{\frac{n}{2}}$ and $\tilde{\boldsymbol{z}} \in \mathbb{C}^{\frac{\tilde{n}}{2}}$ respectively. They are related by

$$\tilde{z}_{j \bmod \frac{\tilde{n}}{2}} = \tfrac{1}{\Delta} \tilde{\boldsymbol{a}}(\xi_j) = \tfrac{1}{\Delta} \tilde{\boldsymbol{a}}(\zeta_j^{n/\tilde{n}}) = \tfrac{1}{\Delta} \boldsymbol{a}(\zeta_j) = z_j$$

for $0 \leq j < \frac{n}{2}$. Hence, $\boldsymbol{z} = (z_i)_{i=0}^{\frac{n}{2}-1}$ can be understood as the vector obtained from $\tilde{\boldsymbol{z}} = (\tilde{z})_{i=0}^{\frac{\tilde{n}}{2}-1}$ by concatenating itself for $n/\tilde{n}$ times. The main advantage of sparsely packing is that it reduces the complexity from $O(\frac{n}{2})$ to $O(n_{\text{slots}})$ during linear transformation when $n_{\text{slots}} \ll n$.

4.4 HE KEM

Besides the secret key sk and public key pk in a typical RLWE cryptosystem (Section 3.6), in HE there are a series of additional keys that used for ciphertext evaluation, called *evaluation keys*, denoted by evk.

4.4.1 Secret Key and Public Key

The generation of sk and pk, and encryption/decryption are explained in Section 3.6 since they are inherent to the RLWE problem. For simplicity, instead of using the RNS-representation $\mathsf{pk} = \left(\{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}}, \{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}}\right)$, where $\boldsymbol{b} = [-\boldsymbol{a}\cdot\mathsf{sk}+\boldsymbol{e}]_q^{\pm}$ and $\{\hat{\boldsymbol{a}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{a}), \{\hat{\boldsymbol{b}}^k\}_{k=1}^{\mathsf{dim}} = \mathsf{rns\_decomposition}(\boldsymbol{b})$, we use the MPI-representation $\mathsf{pk} = (\boldsymbol{b} = [-\boldsymbol{a}\cdot\mathsf{sk}+\boldsymbol{e}]_{q_L}^{\pm}, \boldsymbol{a})$ since both representations are equivalent.

4.4.2 Evaluation Keys

HE's evk includes *relinearization key* rlk, used for ciphertext-ciphertext multiplication, a set of *rotation keys* $\{\mathsf{rk}_r\}_{r=0}^{n_{\text{slots}}-1}$, used for ciphertext rotation, and *conjugation key* ck, used for ciphertext conjugation. Since their generation and encapsulation can be included into the same procedure, the three evaluation keys also got a common name *switching keys* (swk). We firstly discuss how a general swk is generated, then discuss the KEM of rlk, $\{\mathsf{rk}_r\}_{r=0}^{n_{\text{slots}}-1}$ and ck respectively.

The purpose of the *key-switching* operation is to convert a ciphertext into a ciphertext of the same message with respect to a "secret key sk'" other than sk [CHK+18]. swk also has a same structure as pk: $\mathsf{swk} = (\boldsymbol{b}, \boldsymbol{a})$. For $\mathsf{sk}' \in R^{\vee}$, sample $\boldsymbol{a} \leftarrow R_{P\cdot q_L}$ and $\boldsymbol{e} \leftarrow \mathcal{D}_\sigma$, calculate $\boldsymbol{b} = [-\boldsymbol{a}\cdot\boldsymbol{s} + \boldsymbol{e} + P\mathsf{sk}']_{Pq_L}^{\pm}$, we yield the key-switching key

$$\mathsf{swk} = \mathsf{genswk}(\mathsf{sk}', \mathsf{sk}) = (\boldsymbol{b} = [-\boldsymbol{a}\cdot\mathsf{sk} + \boldsymbol{e} + P\mathsf{sk}']_{Pq_L}^{\pm}, \boldsymbol{a}) \tag{4.11}$$

It turns out that what exact the sk' is. In order to be consistent with the polynomial notation, we use $\boldsymbol{s} = \mathsf{sk}$ and $\boldsymbol{s}' = \mathsf{sk}'$. The sk' for the mentioned keys are

$$\boldsymbol{s}'_{\mathsf{rlk}} = \boldsymbol{s}\cdot\boldsymbol{s}, \quad \{\boldsymbol{s}'_{\mathsf{rk}_r} = \tau_{5^r \bmod 2n}(\boldsymbol{s})\}_{r=0}^{n/2-1}, \quad \boldsymbol{s}'_{\mathsf{ck}} = \tau_{-1}(\boldsymbol{s})$$

where "$2n$" refers to the 2's power cyclotomic index. Then

$$\mathsf{rlk} = \mathsf{genswk}(\boldsymbol{s}\cdot\boldsymbol{s}, \boldsymbol{s}) \tag{4.12a}$$

$$\mathsf{rk}_r = \mathsf{genswk}(\tau_{5^r \bmod 2n}(\boldsymbol{s}), \boldsymbol{s}),\ 0 \le r < n_{\text{slots}} \tag{4.12b}$$

$$\mathsf{ck} = \mathsf{genswk}(\tau_{-1}(\boldsymbol{s}), \boldsymbol{s}) \tag{4.12c}$$

4.5 HE Primitives

Besides encryption/decryption in a typical RLWE cryptosystem (Section 3.6), HE also supports ciphertext addition, multiplication, rescale/mod-down, rotation and conjugation. Though encryption/decryption are born with RLWE problem, it's still necessary to recap them, since in HE, three parameters accompany with the ciphertext are important for analysis — the *current* level ℓ. the scaling factor ν, and the noise bound B (i.e. $\|\boldsymbol{e}\|_\infty^{\mathsf{can}} \le B$). We use $(\mathsf{ct}, \ell, \nu, B)$ to denote the ciphertext tagged with its parameters. Similarly, the plaintext has its own property – the canonical embedding norm $\|\boldsymbol{a}\|_\infty^{\mathsf{can}}$. We use $(\mathsf{pt}, \nu = \|\mathsf{pt}\|_\infty^{\mathsf{can}})$ to denote the plaintext tagged with its parameters.

4.5.1 HE Encryption/Decryption

After encoding the inputted message $\boldsymbol{z} \in \mathbb{C}^{n_{\text{slots}}}$ to plaintext $\boldsymbol{a} = \mathsf{pt} = \mathsf{ecd}(\boldsymbol{z}, \Delta)$, we can either encrypt it via pk (Algorithm 14) or sk (Algorithm 15) with $q = q_L$: $\mathsf{ct} = \mathsf{enc}_{\mathsf{pk/sk}}(\mathsf{pt}, \mathsf{pk/sk}, q_L)$. This is at the stage of highest level $\ell = L$ and scaling factor $\nu = \Delta$, we call this ciphertext is the *freshly encrypted* ciphertext $(\mathsf{ct}, L, \nu = \Delta, B_{\text{clean}})$, where Δ should guarantee $\Delta > \|\mathsf{pt}\|_\infty^{\mathsf{can}}$.

$B_{\text{clean}} = 8\sqrt{2\sigma n} + 6\sigma\sqrt{n} + 16\sigma\sqrt{hn}$ denotes the noise bound of the encryption, which can be calculated right after the initialization (4.1). Lemma 1 of [CKKS17] states that, to guarantee that for $\boldsymbol{z} \in \mathbb{Z}^{n/2}$, $\mathsf{pt} = \mathsf{ecd}(\boldsymbol{z}, \Delta)$, $\mathsf{ct} = \mathsf{enc}_{\mathsf{pk}}(\mathsf{pt}, \mathsf{pk}, q_L)$, decrypt and decode $\mathsf{dcd}(\mathsf{dec}(\mathsf{ct}, \mathsf{sk}, q_L))$ recovers the original $\boldsymbol{z}$, the scaling factor should also be $\Delta > n + 2B_{\text{clean}}$.

After several HE evaluations, the ciphertext will become $(\mathsf{ct}, \ell, \nu = \Delta, B)$, where $\ell \le L$ is likely smaller than L so that $q_\ell \le q_L$. Then $\mathsf{pt} = \mathsf{dec}(\mathsf{ct}, \mathsf{sk}) = [\mathsf{ct}.\boldsymbol{a} \cdot \mathsf{sk} + \mathsf{ct}.\boldsymbol{b}]_{q_\ell}^{\pm}$.

4.5.2 HE Addition

There are two cases involving addition: one ciphertext adds another ciphertext, or adds another plaintext. Both cases does not change the level, though.

Add plaintext Let ciphertext $(\mathsf{ct}, \ell, \nu, B)$ with $\mathsf{ct} = (\boldsymbol{b}, \boldsymbol{a})$, and plaintext $(\mathsf{pt}, \|\mathsf{pt}\|_\infty^{\mathsf{can}})$, then

$$\mathsf{add}_{\mathsf{pt}}\left((\mathsf{ct}, \ell, \nu, B), (\mathsf{pt}, \|\mathsf{pt}\|_\infty^{\mathsf{can}})\right) = \left(([\boldsymbol{b} + \mathsf{pt}]_{q_\ell}^{\pm}, \boldsymbol{a}), \ell, \nu + \|\mathsf{pt}\|_\infty^{\mathsf{can}}, B\right) \tag{4.13}$$

Add ciphertext Two ciphertexts can be added iff their level are equal. Let two ciphertexts $(\mathsf{ct}_1, \ell, \nu_1, B_1)$ and $(\mathsf{ct}_2, \ell, \nu_2, B_2)$, with $\mathsf{ct}_1 = (\boldsymbol{b}_1, \boldsymbol{a}_1), \mathsf{ct}_2 = (\boldsymbol{b}_2, \boldsymbol{a}_2)$, then

$$\mathsf{add}\left((\mathsf{ct}_1, \ell, \nu_1, B_1), (\mathsf{ct}_2, \ell, \nu_2, B_2)\right) = \left(([\boldsymbol{b}_1 + \boldsymbol{b}_2]_{q_\ell}^{\pm}, [\boldsymbol{a}_1 + \boldsymbol{a}_2]_{q_\ell}^{\pm}), \ell, \nu_1 + \nu_2, B_1 + B_2\right) \tag{4.14}$$

Taking negative can also be categorized to addition operation. Together with $\mathsf{add}_{\mathsf{pt}}, \mathsf{add}$, HE subtraction $\mathsf{sub}_{\mathsf{pt}}$ $(\mathsf{ct} - \mathsf{pt})$ and sub $(\mathsf{ct}_1 - \mathsf{ct}_2)$ can be easily constructed so we omit them in the text.

Negative Let ciphertext $(\mathsf{ct}, \ell, \nu, B)$ with $\mathsf{ct} = (\boldsymbol{b}, \boldsymbol{a})$, its negative is

$$\mathsf{neg}\left((\mathsf{ct}, \ell, \nu, B)\right) = \left(([-\boldsymbol{b}]_{q_\ell}^{\pm}, [-\boldsymbol{a}]_{q_\ell}^{\pm}), \ell, \nu, B\right) \tag{4.15}$$

4.5.3 HE Rescale and Mod-Down

As we will see, the multiplication (Section 4.5.4) increases the scaling factor of a ciphertext to its square. To cure this, CKKS scheme introduce a *rescale* procedure. A similar procedure is *mod-down*. Both procedure increase the noise bound by $B_{\text{scale}} = \sqrt{n/3}(3 + 8\sqrt{h})$.

Rescale Let ciphertext $(\mathsf{ct}, \ell, \nu, B)$, then rescale from ℓ to $\ell' < \ell$ is

$$\mathsf{rs}_{\ell \to \ell'}(\mathsf{ct}) = \left(\left[\left\lfloor \frac{q_{\ell'}}{q_\ell} \mathsf{ct} \right\rceil\right]_{q_{\ell'}}^{\pm}, \ell', \mathfrak{p}^{\ell' - \ell}\nu, \mathfrak{p}^{\ell' - \ell}B + B_{\text{rs}}\right) \tag{4.16a}$$

A special case is $\ell' = \ell - 1$, which is used right after the ciphertext-ciphertext multiplication:

$$\mathsf{rs}_{\ell \to \ell-1}(\mathsf{ct}) = \left(\left[\left\lfloor \frac{q_{\ell-1}}{q_\ell} \mathsf{ct} \right\rceil \right]^{\pm}_{q_{\ell-1}}, \ell - 1, \mathfrak{p}^{-1}\nu, \mathfrak{p}^{-1}B + B_{\mathrm{rs}} \right) \tag{4.16b}$$

Mod-down Let ciphertext $(\mathsf{ct}, \ell, \nu, B)$, then mod-down from ℓ to $\ell' < \ell$ is

$$\mathsf{moddown}_{\ell \to \ell'}(\mathsf{ct}) = \left([\mathsf{ct}]^{\pm}_{q_{\ell'}}, \ell', \nu, \mathfrak{p}^{\ell'-\ell}B + B_{\mathrm{rs}} \right) \tag{4.17}$$

4.5.4 HE Multiplication

There are two cases involving multiplication: one ciphertext multiplies another ciphertext, or multiplies another plaintext. Both cases double the scaling factor ν, so both cases should do rescale procedure to reset ν back to the original size, which leads to the level decreases by 1: (4.16b) $\mathsf{rs}_{\ell \to \ell-1}(\mathsf{ct}_{\mathsf{mult}})$. That's the reason why LHE support limited deep circuit.

Multiply plaintext Let ciphertext $(\mathsf{ct}, \ell, \nu, B)$ with $\mathsf{ct} = (\boldsymbol{b}, \boldsymbol{a})$, plaintext $(\mathsf{pt}, \|\mathsf{pt}\|^{\mathsf{can}}_{\infty})$, then

$$\mathsf{mult}_{\mathsf{pt}}\left((\mathsf{ct}, \ell, \nu, B), (\mathsf{pt}, \|\mathsf{pt}\|^{\mathsf{can}}_{\infty})\right) = \left(([\boldsymbol{b} \cdot \mathsf{pt}]^{\pm}_{q_\ell}, [\boldsymbol{a} \cdot \mathsf{pt}]^{\pm}_{q_\ell}), \ell, \nu\|\mathsf{pt}\|^{\mathsf{can}}_{\infty}, B\|\mathsf{pt}\|^{\mathsf{can}}_{\infty}\right) \tag{4.18}$$

Multiply ciphertext This case composes of two stages: pure multiplication and *relinearization*. Two ciphertexts can be multiplied iff their level are equal. Let two ciphertexts $(\mathsf{ct}_1, \ell, \nu_1, B_1)$ and $(\mathsf{ct}_2, \ell, \nu_2, B_2)$, with $\mathsf{ct}_1 = (\boldsymbol{b}_1, \boldsymbol{a}_1), \mathsf{ct}_2 = (\boldsymbol{b}_2, \boldsymbol{a}_2)$, pure multiplication among ciphertexts results an intermediate result

$$\mathsf{ct}_1 \cdot \mathsf{ct}_2 = (\boldsymbol{b}_1, \boldsymbol{a}_1) \cdot (\boldsymbol{b}_2, \boldsymbol{a}_2) = ([\boldsymbol{b}_1 \cdot \boldsymbol{b}_2]^{\pm}_{q_\ell}, [\boldsymbol{a}_1 \cdot \boldsymbol{b}_2 + \boldsymbol{a}_2 \cdot \boldsymbol{b}_1]^{\pm}_{q_\ell}, [\boldsymbol{a}_1 \cdot \boldsymbol{a}_2]^{\pm}_{q_\ell}) = (\boldsymbol{d}_0, \boldsymbol{d}_1, \boldsymbol{d}_2) \tag{4.19a}$$

then we perform relinearization with rlk in (4.12a)

$$\mathsf{ct}_{\mathsf{mult}} = \mathsf{relin}\left((\boldsymbol{d}_0, \boldsymbol{d}_1, \boldsymbol{d}_2), \mathsf{rlk}\right) = (\boldsymbol{d}_0, \boldsymbol{d}_1) + [\lfloor P^{-1} \cdot \boldsymbol{d}_2 \cdot \mathsf{rlk} \rceil]_{q_\ell} \tag{4.19b}$$

where $\boldsymbol{d}_2 \cdot \mathsf{rlk}$ is the sloppy form of $(\boldsymbol{d}_2 \cdot \mathsf{rlk}.\boldsymbol{b}, \boldsymbol{d}_2 \cdot \mathsf{rlk}.\boldsymbol{a})$. So that the final result is

$$\mathsf{mult}\left((\mathsf{ct}_1, \ell, \nu_1, B_1), (\mathsf{ct}_2, \ell, \nu_2, B_2), \mathsf{rlk}\right) = (\mathsf{ct}_{\mathsf{mult}}, \ell, \nu_1\nu_2, \nu_1 B_2 + \nu_2 B_1 + B_1 B_2 + B_{\mathrm{mult}}(\ell)) \tag{4.19c}$$

where $B_{\mathrm{mult}}(\ell) = P^{-1} q_\ell B_{\mathrm{ks}} + B_{\mathrm{scale}}, B_{\mathrm{ks}} = 8\sigma n/\sqrt{3}$.

4.5.5 HE Key-Switching

Rotation and conjugation are two use cases of *key-switching* and follow the same form. The key-switching operation to $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ is

$$\mathsf{ks}(\mathsf{ct}, \mathsf{swk}) = (\boldsymbol{c}_0, 0) + [\lfloor P^{-1} \cdot \boldsymbol{c}_1 \cdot \mathsf{swk} \rceil]_{q_\ell} \tag{4.20}$$

where $\boldsymbol{c}_1 \cdot \mathsf{swk}$ is the sloppy form of $(\boldsymbol{c}_1 \cdot \mathsf{swk}.\boldsymbol{b}, \boldsymbol{c}_1 \cdot \mathsf{swk}.\boldsymbol{a})$.

Rotation Let two complex messages $\boldsymbol{z} = (z_i)_{i=0}^{n_{\text{slots}}-1} \in \mathbb{C}^{n_{\text{slots}}}$ and $\tilde{\boldsymbol{z}} = (\tilde{z}_j)_{j=0}^{n_{\text{slots}}-1} \in \mathbb{C}^{n_{\text{slots}}}$. The rotation mapping from $\boldsymbol{z} = (z_0, \cdots, z_{n_{\text{slots}}-1})$ to $\tilde{\boldsymbol{z}} = (z_r, \cdots, z_{n_{\text{slots}}-1}, z_0, \cdots, z_{r-1})$ would satisfy $z_i = \tilde{z}_j$ with $[i-j]_{\frac{n}{2}} = r$ for any $0 \le i, j < n_{\text{slots}}$. According to Section 4.3 (4.8), $\boldsymbol{z}$ and $\tilde{\boldsymbol{z}}$ are the decode of plaintext polynomial $\boldsymbol{a}$ and $\tilde{\boldsymbol{a}}$, related by $z_i = \frac{1}{\Delta}\boldsymbol{a}(\zeta_i)$ and $\tilde{z}_j = \frac{1}{\Delta}\tilde{\boldsymbol{a}}(\zeta_j)$. So it turns out that how to transform $\boldsymbol{a}(\zeta_i)$ to $\tilde{\boldsymbol{a}}(\zeta_j)$. This is accomplished by automorphism. Recall the stated reason in Section 3.2.3 and 3.4.5, let $k = 5^{i-j} \bmod 2n = 5^r \bmod 2n$,

$$\tilde{z}_j \overset{(4.8)}{=} \frac{1}{\Delta}\tilde{\boldsymbol{a}}(\zeta_j) = \frac{1}{\Delta}\tau_{5^{i-j} \bmod 2n}\left(\boldsymbol{a}(\zeta_j)\right) \overset{(3.20)}{=} \frac{1}{\Delta}\boldsymbol{a}\left(\zeta_j^{5^{i-j} \bmod 2n}\right) \overset{(4.3)}{=} \frac{1}{\Delta}\boldsymbol{a}(\zeta_i) \overset{(4.8)}{=} z_i$$

Therefore the sk' of the rotation keys are obtained by $\mathsf{sk}' = \tau_{5^{i-j} \bmod 2n}(\mathsf{sk})$, which verifies the result in (4.12b). There are altogether n_{slots} pieces of rotation keys. To rotate the ciphertext ct by r slots, we do (4.21) with the parameters ℓ, ν, B preserve unchanged.

$$\mathsf{rot}(\mathsf{ct}, r, \mathsf{rk}) = \mathsf{ks}(\tau_{5^r \bmod 2n}(\mathsf{ct}), \mathsf{rk}_r) \tag{4.21}$$

Conjugation Let $\boldsymbol{z} = (z_i)_{i=0}^{n_{\text{slots}}-1}$ with $z_i = \frac{1}{\Delta}\boldsymbol{a}(\zeta_i)$ where $\boldsymbol{a} = \mathsf{pt}$ is the plaintext polynomial. It's conjugate is

$$\bar{z}_i = \frac{1}{\Delta}\overline{\boldsymbol{a}(\zeta_i)} = \frac{1}{\Delta}\boldsymbol{a}(\bar{\zeta}_i) = \boldsymbol{a}(\zeta_i^{-1}) \overset{(3.21)}{=} \tau_{-1}(\boldsymbol{a}(\zeta_i))$$

So $\tau_{-1}(\mathsf{ct})$ is a valid encryption of the plaintext vector $\bar{\boldsymbol{z}}$ with the secret key $\tau_{-1}(\boldsymbol{s})$. Therefore the sk' of the conjugation key is obtained by $\mathsf{sk}' = \tau_{-1}(\mathsf{sk})$, which verifies the result $\mathsf{ck} = \mathsf{genswk}(\tau_{-1}(\boldsymbol{s}), \boldsymbol{s})$ in (4.12c). To take conjugation to a ciphertext ct, we do (4.22) with the parameters ℓ, ν, B preserve unchanged.

$$\mathsf{conj}(\mathsf{ct}, \mathsf{ck}) = \mathsf{ks}(\tau_{-1}(\mathsf{ct}), \mathsf{ck}) \tag{4.22}$$

4.6 Advanced HE Evaluations

This section covers all the advanced HE evaluation functions supported by GPQHE. All of them use at least one of the primitives in Section 4.5. We categorize the advanced operations into linear and nonlinear, according to the fact that whether the evaluation result depends on an additional iteration factor or the Taylor expansion order of the evaluated function. If the result does not depend on the iteration, this operation is said to be "linear", else, is called "nonlinear". Ciphertext comparison is a highly nonlinear operation.

We talk about several instances supported in GPQHE. Matrix-vector multiplication, vector sum, index fetching, $\|\boldsymbol{z}\|_2^2$ are typical examples of linear transformations, explained in Section 4.6.1. Exponential, logarithm and Sigmoid depend on their Taylor expansion order, so they are nonlinear, which will be explained in Section 4.6.2. Finally in Section 4.6.3 we discuss the comparison functionality, including inverse and square root, which requires additional iteration factor.

4.6.1 Linear Transformations

Matrix-Vector Multiplication gemv (**ge**neral **m**atrix-**v**ector multiplication) is a borrowed terminology from *BLAS (Basic Linear Algebra Subprograms)* which implements $\boldsymbol{y} \leftarrow \alpha \boldsymbol{A}\boldsymbol{x} + \beta \boldsymbol{y}$ for a general $m \times n$ matrix $\boldsymbol{A}$. What we do here is $\mathsf{ct}' = \boldsymbol{A}\mathsf{ct}$, the gemv for HE, where $\boldsymbol{A} \in \mathbb{C}^{n_{\text{slots}} \times n_{\text{slots}}}$. Let

$$\boldsymbol{u}_j = (A_{0,j}, A_{1,j+1}, \cdots, A_{n_{\text{slots}}-j-1, n_{\text{slots}}-1}, A_{n_{\text{slots}}-j,0}, \cdots, A_{n_{\text{slots}}-1,j-1})$$

denote the shifted diagonal vector of $\boldsymbol{A}$ for $0 \le j < n_{\text{slots}}$, and

$$\rho(\boldsymbol{z}, r) = (z_r, \cdots, z_{n_{\text{slots}}-1}, z_0, \cdots, z_{r-1})$$

a rotated vector of $\boldsymbol{z} = (z_0, \cdots, z_{n_{\text{slots}}-1})$. We have

$$\boldsymbol{A}\boldsymbol{z} = \textstyle\sum_{0 \le j < n_{\text{slots}}} \boldsymbol{u}_j \circ \rho(\boldsymbol{z}, j) \tag{4.23}$$

Therefore, to calculate $\boldsymbol{A}\mathsf{ct}$, with $\boldsymbol{A}$ a plaintext matrix and $\mathsf{ct} = \mathsf{Enc}(\boldsymbol{z})$ a ciphertext, just replace $\rho(\boldsymbol{z}, j)$ by $\mathsf{rot}(\mathsf{ct}, r, \mathsf{rk})$:

$$\boldsymbol{A}\mathsf{ct} = \textstyle\sum_{0 \le j < n_{\text{slots}}} \boldsymbol{u}_j \circ \mathsf{rot}(\mathsf{ct}, r, \mathsf{rk}) \tag{4.24}$$

Complete implementation is shown in Algorithm 17. However, this algorithm requires $(n_{\text{slots}} - 1)$ rotations and n multiplication with scalar polynomials. Its complexity can be reduced down using the idea of *baby-step giant-step* algorithm [HS21]. Let $n_1 = O(\sqrt{n_{\text{slots}}})$ be a divisor of n_{slots}, and denote $n_2 = n_{\text{slots}}/n_1$. Eq 4.23 can then express as

$$\begin{aligned} \boldsymbol{A}\boldsymbol{z} &= \textstyle\sum_{0 \le i < n_2} \sum_{0 \le j < n_1} \boldsymbol{u}_{i \cdot n_1 + j} \circ \rho(\boldsymbol{z}, i \cdot n_1 + j) \\ &= \textstyle\sum_{0 \le i < n_2} \rho\left(\sum_{0 \le j < n_1} \rho(\boldsymbol{u}_{i \cdot n_1 + j}, -i \cdot n_1) \circ \rho(\boldsymbol{z}, j), i \cdot n_1 \right) \end{aligned} \tag{4.25}$$

So in total, there are $(n_1 - 1) + (n_2 - 1) = O(\sqrt{n_{\text{slots}}})$ rotations and $n_1 \cdot n_2 = O(n_{\text{slots}})$ scalar multiplications. Implementation is shown in Algorithm 18. gemv decreases the ciphertext level to $\ell - 1$.

Algorithm 17 $\mathsf{ct}' = \mathsf{gemv}(\boldsymbol{A}, \mathsf{ct}, \mathsf{rk}) = \boldsymbol{A}\mathsf{ct}$

Input: Plain matrix $\boldsymbol{A} \in \mathbb{C}^{n_{\text{slots}} \times n_{\text{slots}}}$; ciphertext ct; rotation keys rk
Output: $\mathsf{ct}' = \boldsymbol{A}\mathsf{ct} = \mathsf{gemv}(\boldsymbol{A}, \mathsf{ct}, \mathsf{rk})$

1: $\boldsymbol{u}_0 = (A_{0,0}, A_{1,1}, \cdots, A_{n_{\text{slots}}-1, n_{\text{slots}}-1})$
2: $\mathsf{ct}' = \mathsf{mult}_{\mathsf{pt}}(\mathsf{ct}, \mathsf{ecd}(\boldsymbol{u}_0, \Delta))$
3: **for** $j = 1$ to $n_{\text{slots}} - 1$ **do**
4: $\quad \boldsymbol{u}_j = (A_{0,j}, A_{1,j+1}, \cdots, A_{n_{\text{slots}}-j-1, n_{\text{slots}}-1}, A_{n_{\text{slots}}-j,0}, \cdots, A_{n_{\text{slots}}-1,j-1})$
5: $\quad \mathsf{ct}_j = \mathsf{mult}_{\mathsf{pt}}(\mathsf{rot}(\mathsf{ct}, j, \mathsf{rk}), \mathsf{ecd}(\boldsymbol{u}_j, \Delta))$
6: $\quad \mathsf{ct}' = \mathsf{add}(\mathsf{ct}', \mathsf{ct}_j)$
7: **end for**
8: **return** $\mathsf{rs}_{\ell \to \ell-1}(\mathsf{ct}')$

Algorithm 18 $\mathsf{ct}' = \mathsf{gemv}(\boldsymbol{A}, \mathsf{ct}, \mathsf{rk}) = \boldsymbol{A}\mathsf{ct}$ with baby-step giant-step

Input: Plain matrix $\boldsymbol{A} \in \mathbb{C}^{n_{\text{slots}} \times n_{\text{slots}}}$; ciphertext ct; rotation keys rk
Output: $\mathsf{ct}' = \boldsymbol{A}\mathsf{ct}$

$n_1 = \lfloor \sqrt{n_{\text{slots}}} \rceil$
if $n_{\text{slots}} \neq n_1^2$ **then**
    $n_1 = \lfloor \sqrt{2n_{\text{slots}}} \rceil$ ▷ giant step
end if
$n_2 = n_{\text{slots}}/n_1$ ▷ baby step
for $i = 0$ to $n_2 - 1$ **do**
    for $j = 0$ to $n_1 - 1$ **do**
        $k = i \cdot n_1 + j$
        $\boldsymbol{u}_k = \left(A_{0,k}, A_{1,k+1}, \cdots, A_{n_{\text{slots}}-k-1, n_{\text{slots}}-1}, A_{n_{\text{slots}}-k,0}, \cdots, A_{n_{\text{slots}}-1,k-1}\right)$
        $\mathsf{ct}_j = \mathsf{mult}_{\mathsf{pt}}(\mathsf{rot}(\mathsf{ct}, j, \mathsf{rk}), \mathsf{ecd}(\rho(\boldsymbol{u}_k, -i \cdot n_1), \Delta))$
        $\mathsf{ct}'' = (j == 0)?\ \mathsf{ct}_j\ :\ \mathsf{add}(\mathsf{ct}'', \mathsf{ct}_j)$
    end for
    $\mathsf{ct}' = (i == 0)?\ \mathsf{ct}''\ :\ \mathsf{rot}(\mathsf{ct}'', i \cdot n_1, \mathsf{rk})$
end for
return $\mathsf{rs}_{\ell \to \ell-1}(\mathsf{ct}')$

Vector Sum This is a special case of gemv. We set the first row of $\boldsymbol{A} \in \mathbb{C}^{n_{\text{slots}} \times n_{\text{slots}}}$ as $\mathbb{1}_{1 \times n_{\text{slots}}}$, and the rest elements being all zero. Let $\boldsymbol{z} \in \mathbb{C}^{n_{\text{slots}}}$, $\mathsf{ct} = \mathsf{enc}_{\mathsf{pk/sk}}(\mathsf{ecd}(\boldsymbol{z}, \Delta))$ and $\mathsf{ct}' = \mathsf{sum}(\mathsf{ct})$, then $\mathsf{dcd}(\mathsf{dec}(\mathsf{ct}', \mathsf{sk}), \Delta)[0] = \sum_{i=0}^{n_{\text{slots}}-1} z_i$.

Index Fetching This is a special case of gemv. Let $0 \leq l < n_{\text{slots}}$ being the index to be fetched. Use $\boldsymbol{A} = \boldsymbol{0}_{n_{\text{slots}} \times n_{\text{slots}}}$ and set $\boldsymbol{A}_{l,l} = 1$. Let $\boldsymbol{z} \in \mathbb{C}^{n_{\text{slots}}}$, $\mathsf{ct} = \mathsf{enc}_{\mathsf{pk/sk}}(\mathsf{ecd}(\boldsymbol{z}, \Delta))$ and $\mathsf{ct}' = \mathsf{idx}(\mathsf{ct}, l)$, then $\mathsf{dcd}(\mathsf{dec}(\mathsf{ct}', \mathsf{sk}), \Delta)[l] = z_l$.

Square of 2-Norm In practice, calculating $\|\boldsymbol{z}\|_2^2$ or $\|\boldsymbol{z}\|_2$ does not matter, especially in the case of comparing the norm of step size with a small number, c.f. QP in Section 5.2.3. In encrypted regime, taking square root is an exhaustive step, so we calculate the square of the 2-norm, i.e. $\|\mathsf{ct}\|_2^2$. Let ct being an encryption of $\boldsymbol{z}$. The HE 2-norm is calculated in (4.26), where the ciphertext level decreases to $\ell - 2$.

$$\mathsf{nrm22}(\mathsf{ct}, \mathsf{rlk}, \mathsf{rk}, \mathsf{ck}) = \mathsf{sum}(\mathsf{rs}_{\ell \to \ell-1}(\mathsf{mult}(\mathsf{ct}, \mathsf{conj}(\mathsf{ct}), \mathsf{rlk}))) \tag{4.26}$$

4.6.2 Nonlinear Functions

Exponential Consider an expansion to a exponential e^x up to $O(x^8)$:

$$\begin{aligned} e^x =& 1 + x + \tfrac{1}{2!}x^2 + \tfrac{1}{3!}x^3 + \tfrac{1}{4!}x^4 + \tfrac{1}{5!}x^5 + \tfrac{1}{6!}x^6 + \tfrac{1}{7!}x^7 + O(x^8) \\ =& \underbrace{\Big[\underbrace{\underbrace{1 \cdot (1+x)}_{\ell-1}}_{\ell-2} + \underbrace{\underbrace{\tfrac{1}{3!} \cdot (3+x)}_{\ell-1}\underbrace{x^2}_{\ell-1}}_{\ell-2}\Big]}_{\ell-3} + \underbrace{\Big[\underbrace{\underbrace{\tfrac{1}{5!} \cdot (5+x)}_{\ell-1}}_{\ell-2} + \underbrace{\underbrace{\tfrac{1}{7!} \cdot (7+x)}_{\ell-1}\underbrace{x^2}_{\ell-1}}_{\ell-2}\Big]\underbrace{x^4}_{\ell-2}}_{\ell-3} \end{aligned} \tag{4.27}$$

Since HE multiplication would decrease the level by 1. So the above grouping scheme to calculate the exponential would decrease the ciphertext level to $\ell-3$. We label this evaluation as $\mathsf{exp}_{\mathsf{core}}(\mathsf{ct})$.

In practice, we always calculate e^{ax} (both a and x could be complex). To tackle with the a which is so large that invalidate the Taylor expansion, we perform Algorithm 19.

Algorithm 19 $\mathsf{exp}(a \cdot \mathsf{ct}, \mathsf{rlk}, r)$

Input: ciphertext ct; complex constant a, relinearization key rlk; iteration number r
Output: $\mathsf{exp}(a \cdot \mathsf{ct})$

1: $a = a/2^r$
2: $\tilde{\boldsymbol{a}} = (a, \cdots, a)$ ▷ a vector of length n_{slots}
3: $\mathsf{ct}' = \mathsf{rs}_{\ell \to \ell-1}(\mathsf{mult}_{\mathsf{pt}}(\mathsf{ct}, \mathsf{ecd}(\tilde{\boldsymbol{a}}, \Delta)))$ ▷ current level is $\ell - 1$
4: $\mathsf{ct}' = \mathsf{exp}_{\mathsf{core}}(\mathsf{ct}')$ ▷ current level is $\ell - 4$
5: **for** $j = 0$ to $r - 1$ **do**
6: $\quad \mathsf{ct}' = \mathsf{rs}_{\ell \to \ell-1}(\mathsf{mult}(\mathsf{ct}', \mathsf{ct}', \mathsf{rlk}))$ ▷ $(\mathsf{ct}')^2$
7: **end for**
8: **return** ct' ▷ current level is $\ell - 4 - r$

The application scenario of HE exponential function is to calculate $\sin\theta = \frac{1}{2i}(e^{i\theta} - e^{-i\theta})$ and $\cos\theta = \frac{1}{2}(e^{i\theta} + e^{-i\theta})$. Homomorphically evaluate $\sin\theta$ is an essential step in bootstrap.

Logarithm Consider an expansion to a logarithm $\ln(1+x)$ up to $O(x^{11})$:

$$\begin{aligned}\ln(1+x) =& x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 + \tfrac{1}{5}x^5 - \tfrac{1}{6}x^6 + \tfrac{1}{7}x^7 - \tfrac{1}{8}x^8 + \tfrac{1}{9}x^9 - \tfrac{1}{10}x^{10} + O(x^{11}) \\ =& \tfrac{1}{9}x \underbrace{\left[9 + \tfrac{9}{3}x^2 + \tfrac{9}{5}x^4 + \tfrac{9}{7}x^2x^4 + x^8\right]}_{\ell-3} - \tfrac{1}{10}x^2 \underbrace{\left[\tfrac{10}{2} + \tfrac{10}{4}x^2 + \tfrac{10}{6}x^4 + \tfrac{10}{8}x^2x^4 + x^8\right]}_{\ell-3}\end{aligned} \tag{4.28}$$

So $\mathsf{ln}(\mathsf{ct}, \mathsf{rlk})$ would decreases the level to $\ell - 4$.

Sigmoid Consider an expansion to a Sigmoid $\frac{1}{1+\exp(-x)}$ up to $O(x^{10})$:

$$\begin{aligned}\tfrac{1}{1+\exp(-x)} =& \tfrac{1}{2} + \tfrac{1}{4}x - \tfrac{1}{48}x^3 + \tfrac{1}{480}x^5 - \tfrac{17}{80640}x^7 + \tfrac{31}{1451520}x^9 + O(x^{10}) \\ =& \underbrace{\tfrac{1}{2} - \tfrac{1}{48}x\left[\tfrac{1/4}{-1/48} + x^2\right]}_{\ell-2} \underbrace{- \tfrac{17}{80640}x\left[\tfrac{1/480}{-17/80640} + x^2\right]}_{\ell-2} \underbrace{x^4}_{\ell-2} \underbrace{+ \tfrac{31}{1451520}x}_{\ell-1} \underbrace{x^8}_{\ell-3} \\ & \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\ell-3} \qquad\qquad \underbrace{\qquad\qquad\qquad\qquad}_{\ell-4}\end{aligned} \tag{4.29}$$

So $\mathsf{sigmoid}(\mathsf{ct}, \mathsf{rlk})$ would decreases the level to $\ell - 4$.

4.6.3 Comparison

Inverse We exploit the Goldschmidt's division algorithm [CKK+19] (4.30) to compute the inverse of a positive real $x \in [\frac{1}{2}, \frac{3}{2})$. $1+(1-x)^{2^r}$ converges to 1 as $r \to \infty$, so the approximation

holds for large $d > 0$. Algorithm 20 shows the HE inverse, where the ciphertext level decreases to $\ell - d - 1$.

$$\frac{1}{x} = \frac{1}{1-(1-x)} = \prod_{r=0}^{\infty}\left(1+(1-x)^{2^r}\right) \approx \prod_{r=0}^{d}\left(1+(1-x)^{2^r}\right) \tag{4.30}$$

Algorithm 20 $\mathsf{inv}(\mathsf{ct}, \mathsf{rlk}, d)$

Input: ct, an encryption of a real vector with each element $x \in [\frac{1}{2}, \frac{3}{2})$; iteration d; rlk
Output: ct', an encryption of a real vector with each element $1/x$

1: $a_0 = \mathsf{moddown}_{\ell\to\ell-1}(\mathsf{neg}(\mathsf{sub}_{\mathsf{pt}}(\mathsf{ct}, 2)))$ ▷ $a_0 = 2 - x$
2: $b_0 = \mathsf{neg}(\mathsf{sub}_{\mathsf{pt}}(\mathsf{ct}, 1))$ ▷ $b_0 = 1 - x$
3: **for** $r = 0$ to $d - 1$ **do**
4: $\quad b_{r+1} = \mathsf{rs}_{\ell\to\ell-1}(\mathsf{mult}(b_r, b_r, \mathsf{rlk}))$ ▷ $b_{r+1} = b_r^2$
5: $\quad a_{r+1} = \mathsf{rs}_{\ell\to\ell-1}(\mathsf{mult}(a_r, \mathsf{add}_{\mathsf{pt}}(b_{r+1}, 1), \mathsf{rlk}))$ ▷ $a_{r+1} = a_r \cdot (1 + b_{r+1})$
6: **end for**
7: **return** $\mathsf{ct}' = a_d$

Square Root We exploit Wilkes's method [CKK+19] to compute the square root of a positive real $x \in (0, 1)$, implemented in Algorithm 21. The ciphertext level decreases to $\ell - 2d$.

Algorithm 21 $\mathsf{sqrt}(\mathsf{ct}, \mathsf{rlk}, d)$

Input: ct, an encryption of a real vector with each element $x \in (0, 1)$; iteration d; rlk
Output: ct', an encryption of a real vector with each element $\sqrt{x}$

1: $a_0 = \mathsf{ct}$ ▷ $a_0 = x$
2: $b_0 = \mathsf{sub}(\mathsf{ct}, 1)$ ▷ $b_0 = x - 1$
3: **for** $r = 0$ to $d - 1$ **do**
4: $\quad t = \mathsf{neg}(\mathsf{sub}_{\mathsf{pt}}(\mathsf{rs}_{\ell\to\ell-1}(\mathsf{mult}_{\mathsf{pt}}(b_r, 0.5)), 1))$ ▷ $t = 1 - \frac{b_r}{2}$
5: $\quad a_{r+1} = \mathsf{mult}(\mathsf{moddown}_{\ell\to\ell-1}(a_r), t, \mathsf{rlk})$ ▷ $a_{r+1} = a_r \cdot (1 - \frac{b_r}{2})$
6: $\quad t = \mathsf{rs}_{\ell\to\ell-1}(\mathsf{mult}_{\mathsf{pt}}(\mathsf{sub}_{\mathsf{pt}}(b_r, 3), 0.25))$ ▷ $t = \frac{b_r-3}{4}$
7: $\quad b_{r+1} = \mathsf{rs}_{\ell\to\ell-1}(\mathsf{mult}(\mathsf{rs}_{\ell\to\ell-1}(\mathsf{mult}(b_r, b_r, \mathsf{rlk})), t, \mathsf{rlk}))$ ▷ $b_{r+1} = b_r^2(\frac{b_r-3}{4})$
8: **end for**
9: **return** $\mathsf{ct}' = a_d$

By now we are clear that why we calculate $||\mathsf{ct}||_2^2$ instead of $||\mathsf{ct}||_2$, since $||\mathsf{ct}||_2 = \mathsf{sqrt}(||\mathsf{ct}||_2^2, \mathsf{rlk}, d)$, it would kill altogether $2d+2$ levels, which means that the RLWE modulus q must set to be very large. The size of q is bounded by n according to [ACC+18]: only by setting a larger n can we set a larger q. However, large n also makes the system slower. So calculating $||\mathsf{ct}||_2$ is not favorable.

Besides, we can also design an absolute value algorithm and min/max algorithm via sqrt, though they are not implemented in GPQHE. Since $|a-b| = \sqrt{(a-b)^2}$, so $\mathsf{abs}(a-b) = \mathsf{sqrt}((a-b)^2, d)$ (with a, b replaced by ciphertext), which decreases the level to $\ell - 1 - 2d$; $\mathsf{max/min}(a, b) = \frac{1}{2}(a+b) \pm \frac{1}{2}|a-b|$, also have depth $\ell - 1 - 2d$.

Compare We adapt the Algorithm 5 from [CKK+19] for comparison: given two real numbers $a, b \in [\frac{1}{2}, \frac{3}{2})$, the algorithm returns 1_- if $a > b$, and 0_+ if $a < b$:

$$\mathsf{cmp}(a-b) = \begin{cases} 1 & \text{if } a \geq b \\ 0 & \text{if } a < b \end{cases} \tag{4.31}$$

With the previous examples, readers should be familiar with the function interface in practical use; similarly, HE operations are mostly element-wise. So we skip many details in order to concisely reveal the idea, shown in Algorithm 22.

Algorithm 22 $\mathsf{cmp}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rlk}, d, d', t, m = 2)$

Input: $\mathsf{ct}_1, \mathsf{ct}_2$, the encryption of two real vectors a, b with each element $x \in [\frac{1}{2}, \frac{3}{2})$; iteration parameters $d, d', t, m = 2$; relinearization key rlk

Output: ct', an encryption of a real vector with each element either approaches to 1_- or approaches to 0_+

1: $a_0 = \frac{\mathsf{ct}_1}{2} \cdot \mathsf{inv}(\frac{\mathsf{ct}_1+\mathsf{ct}_2}{2}, \mathsf{rlk}, d')$ ▷ a_0's level is $\ell_0 = \ell - d' - 3$
2: $b_0 = 1 - a_0$ ▷ b_0's level is $\ell_0 = \ell - d' - 3$
3: **for** $r = 0$ to $t - 1$ **do**
4: $\quad a_{r+1} = a_r^{m=2} \cdot \mathsf{inv}(a_r^{m=2} + b_r^{m=2}, \mathsf{rlk}, d)$ ▷ $\ell_{r+1} = \ell_r - d - 3$
5: $\quad b_{r+1} = 1 - a_{r+1}$
6: **end for**
7: **return** $\mathsf{ct}' = a_r$

The depth (the level decrease caused by the algorithm) and complexity of the algorithm is $d'+1+t(d+\log m+2)$ and $\Theta(d'+t(d+\log m))$, respectively. To further analyze the algorithm, we need one more parameter: the precision bit α. d and d' can simply set as equal. m should be normally chosen as a 2's power, here we simply choose $m = 2$ and let $c = 1 + 2^{-\alpha}$, this gives $t = \log(\alpha/\log c) \approx \alpha$, and thus the depth is optimized to $\Theta(\log(\alpha/\log c) \cdot \log(\alpha + \log(\alpha/\log c)))$.

5 (Encrypted) Model Predictive Control

5.1 Introduction

Model predictive control (MPC) has its roots in *optimal control problem (OCP)*. The idea of MPC is to use a dynamic model to forecast system behavior, and optimize the forecast to obtain the best decision — the control mode at the current time. There are typically two types of optimization problems — the *regulation problem*, where a model forecast is used to produce the optimal control action, and the *estimation problem*, where the past record of measurements is used to produce an optimal state estimation [RMD17]. In practice, to see how it is used to control a system, we always consider a *close-loop simulation* of an investigated model, for example, inverted pendulum or a *continuous stirred tank reactor (CSTR)*. A closed-loop simulation consists of the *ordinary differential equation (ODE)* dynamics of the *plant*, i.e. the investigated model, and the MPC controller; the MPC controller consisting of the regulator, the state estimator, and the target selector. Their relations are shown in Figure 5.1.

The regulator is an essential component: it performs predictive control for the subsequent actuation. To illustrate the application scenario of the homomorphic encryption, we consider a secure enhanced regulator, illustrated in Figure 5.2. In this setting, the regulator receives the current observed state $\mathsf{enc}(\hat{\boldsymbol{x}}^+(k))$, the control from the last time $\mathsf{enc}(\boldsymbol{u}(k-1))$, the reference state $\mathsf{enc}(\boldsymbol{x}_r(k))$ and reference control $\mathsf{enc}(\boldsymbol{u}_r(k))$, then returns an optimal $\mathsf{enc}(\boldsymbol{u}(k))$ and decrypt it for the next control. Throughout this procedure, the regulator in the controller does not know the actual state and control, so that its security is enhanced in this way. This conception was originally proposed in [KF15] and generalized as *encrypted model predictive controller (Encrypted MPC)* in [SDAQP21].

Section 5.2 explains the mechanism of stiff ODE solver, quadratic programming, and discrete-time algebraic Riccati equation solver — although there are little connections among themselves, they play an inseparable role (namely "*primitive*") in either plant dynamics or regulator and estimator, so we introduce them in the very beginning. Section 5.3 explains the mechanism of plant dynamics as well as the investigated CSTR model. Section 5.4, 5.5, 5.6 explain the mechanism of state estimator, target selector, and model predictive regulator, accordingly. Section 5.7 explains how the encrypted MPC works by studying the CSTR model from [RMD17]. Section 5.8 is the summary of how a closed-loop simulation of MPC/encrypted MPC works.

Although in this project we focus on the regulation problem, the methods developed and the conclusion drawn in this chapter can be generalized to the estimation problem as well.

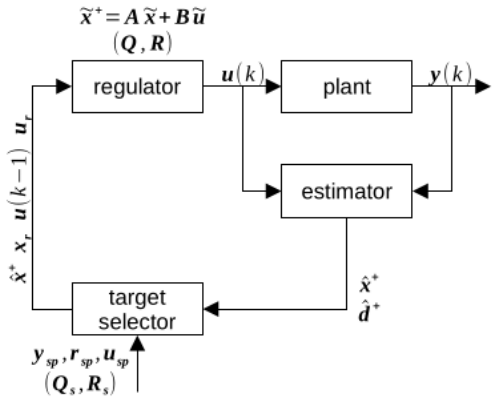

Figure 5.1: The architecture of a closed-loop model predictive control system, which consists of plant, estimator, target selector and regulator. We mainly consider the MPC regulator in this project.

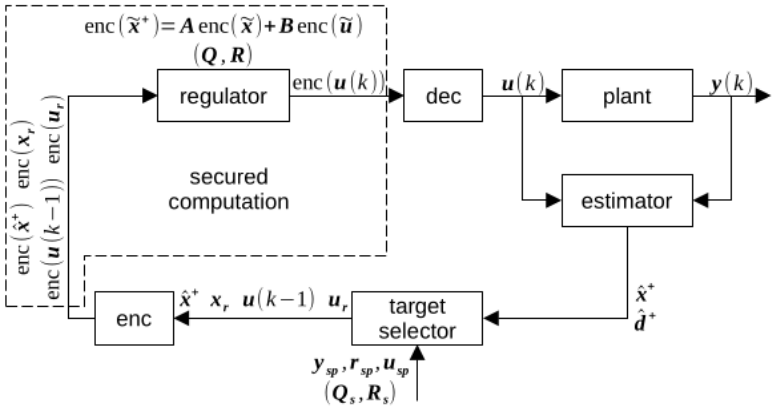

Figure 5.2: Encrypted model predictive controller, in which the regulator receive encrypted signals and calculate the next control.

5.2 Primitive Solvers

5.2.1 Stiff ODE Solver

We briefly talk about the integral method to solve the stiff ODE, since the considered CSTR model should be solved in this way [HS97]. Let $\boldsymbol{x} \in \mathbb{R}^n$, consider a sets of equations

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{x})$$

implicit differencing gives

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{f}(\boldsymbol{x}_{k+1})\Delta t$$

for adjacent time steps k and $k+1$. In general, $\boldsymbol{f}(\boldsymbol{x}_{k+1})$ has to be solved iteratively at each step. However, if we know the Jacobian of the system

$$\boldsymbol{J}(\boldsymbol{x}_k) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_k}$$

then $\boldsymbol{f}(\boldsymbol{x}_{k+1})$ is obtained by Newton's linearization method:

$$\boldsymbol{f}(\boldsymbol{x}_{k+1}) = \boldsymbol{f}(\boldsymbol{x}_k) + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_k} \cdot (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$$

then by rearranging the equation we get

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \left[1 - \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_k} \Delta t\right]^{-1} \boldsymbol{f}(\boldsymbol{x}_k)\Delta t \tag{5.1}$$

The above method is called a *semi-implicit* method.

We use the following symbol to denote the solver interface:

$$\boldsymbol{x}_{k+1} = \mathsf{odes}\left(n, \boldsymbol{x}_k, \boldsymbol{f}, \tfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}, \Delta t\right) \tag{5.2}$$

5.2.2 Discrete-time Algebraic Riccati Equation Solver

We briefly talk about the derivation of *discrete-time algebraic Riccati equation (DARE)*, which is used in the estimator. A typical specification of the discrete-time linear quadratic control problem is to solve the following OCP:

$$\begin{aligned} &\min_{\boldsymbol{u}_0,\cdots,\boldsymbol{u}_{N-1}} \quad V(\mathbf{x},\mathbf{u}) = \textstyle\sum_{k=0}^{N} |\boldsymbol{x}_k|_{\boldsymbol{Q}}^2 + \sum_{k=0}^{N-1} |\boldsymbol{u}_k|_{\boldsymbol{R}}^2 \\ &\text{subject to } \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \text{ for } k = 0, 1, \cdots, N-1 \end{aligned} \tag{5.3}$$

where $\{\boldsymbol{x}_k\}_{k=0}^{N} \in \mathbb{R}^n, \{\boldsymbol{u}_k\}_{k=0}^{N-1} \in \mathbb{R}^m, \boldsymbol{A} \in \mathbb{R}^{n\times n}, \boldsymbol{B} \in \mathbb{R}^{n\times m}$ are system state, the control, state transition matrix and control input matrix, respectively; $\boldsymbol{Q} \in \mathbb{R}^{n\times n}, \boldsymbol{R} \in \mathbb{R}^{m\times m}$ are positive semi-definite state cost matrix and control cost matrix, respectively. The objective function $V(\mathbf{x},\mathbf{u})$ is also called *cost-to-go function* — it is designed in such quadratic form because of its similarity with the "potential V" in physics up to a factor ½.

To solve DARE, we would firstly talk about how to solve a *discrete-time dynamic Riccati equation*. It is solved by *backward dynamic programming (DP)*. To this end we start from the last two time steps $k = N-1, N$:

$$V_{N-1} = |\boldsymbol{x}_N|_{\boldsymbol{Q}}^2 + |\boldsymbol{x}_{N-1}|_{\boldsymbol{Q}}^2 + |\boldsymbol{u}_{N-1}|_{\boldsymbol{R}}^2 = |\boldsymbol{x}_{N-1}|_{\boldsymbol{Q}}^2 + |\boldsymbol{u}_{N-1} - \boldsymbol{v}|_{\boldsymbol{B}^\top\boldsymbol{P}\boldsymbol{B}+\boldsymbol{R}}^2 + |\boldsymbol{x}_{N-1}|_{\boldsymbol{A}^\top\boldsymbol{P}(\boldsymbol{A}+\boldsymbol{B}\boldsymbol{K})}^2$$

where

$$\boldsymbol{v} = -\underbrace{\left(\boldsymbol{B}^\top\boldsymbol{Q}\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\boldsymbol{B}^\top\boldsymbol{Q}\boldsymbol{A}}_{\boldsymbol{K}_{N-1}\in\mathbb{R}^{m\times n}}\boldsymbol{x}_{N-1} = -\boldsymbol{K}_{N-1}\boldsymbol{x}_{N-1}$$

So that V_{N-1} minimized when $\boldsymbol{u}_{N-1}^{\text{op}} = -\boldsymbol{K}_{N-1}\boldsymbol{x}_{N-1}$, i.e. the optimal control law at stage $N-1$ is a linear function of the stage $\boldsymbol{x}_{N-1}$, and the last state is thus $\boldsymbol{x}_N^{\text{op}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}_{N-1})\boldsymbol{x}_{N-1}$. V_{N-1} is minimized to

$$V_{N-1}^{\text{op}} = \boldsymbol{x}_{N-1}^\top \underbrace{\left(\boldsymbol{Q} + \boldsymbol{A}^\top \boldsymbol{P}(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}_{N-1})\right)}_{\boldsymbol{X}_{N-1} \in \mathbb{R}^{n\times n}} \boldsymbol{x}_{N-1} = |\boldsymbol{x}_{N-1}|^2_{\boldsymbol{X}_{N-1}}$$

Now move to the next stage of the DP recursion. Consider $k = N-2, N-1$:

$$V_{N-2} = |\boldsymbol{x}_{N-2}|^2_{\boldsymbol{Q}} + |\boldsymbol{u}_{N-2}|^2_{\boldsymbol{R}} + V_{N-1}^{\text{op}} = |\boldsymbol{x}_{N-2}|^2_{\boldsymbol{Q}} + |\boldsymbol{u}_{N-2}|^2_{\boldsymbol{R}} + |\boldsymbol{x}_{N-1}|^2_{\boldsymbol{X}_{N-1}}$$

Similar to the above procedure, we have $\boldsymbol{u}_{N-2}^{\text{op}} = -\boldsymbol{K}_{N-2}\boldsymbol{x}_{N-2}$, $\boldsymbol{x}_{N-1}^{\text{op}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}_{N-2})\boldsymbol{x}_{N-2}$, $V_{N-2}^{\text{op}} = |\boldsymbol{x}_{N-2}|^2_{\boldsymbol{X}_{N-2}}$, where

$$\begin{aligned}
\boldsymbol{K}_{N-2} &= -\left(\boldsymbol{B}^\top \boldsymbol{X}_{N-1}\boldsymbol{B} + \boldsymbol{R}\right)^{-1} \boldsymbol{B}^\top \boldsymbol{X}_{N-1}\boldsymbol{A} \\
\boldsymbol{X}_{N-2} &= \boldsymbol{Q} + \boldsymbol{A}^\top \boldsymbol{X}_{N-1}\left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}_{N-2}\right)
\end{aligned}$$

The recursion from $\boldsymbol{\Pi}_{N-1}$ to $\boldsymbol{\Pi}_{N-2}$ is known as *backward Riccati iteration*. In general,

$$\begin{aligned}
\boldsymbol{K}_{k-1} &= \left(\boldsymbol{B}^\top \boldsymbol{X}_k \boldsymbol{B} + \boldsymbol{R}\right)^{-1} \boldsymbol{B}^\top \boldsymbol{X}_k \boldsymbol{A}, \ k = N-1, \cdots, 1, 0 \\
\boldsymbol{X}_{k-1} &= \boldsymbol{Q} + \boldsymbol{A}^\top \boldsymbol{X}_k \left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}_{k-1}\right), \ k = N, N-1, \cdots, 1 \\
\boldsymbol{X}_N &= \boldsymbol{Q}
\end{aligned} \tag{5.4}$$

The optimal control policy at stage $k = N-1, \cdots, 0$ is $\boldsymbol{u}_k^{\text{op}} = \boldsymbol{K}_k \boldsymbol{x}_k$, the optimal state in the corresponding next stage is $\boldsymbol{x}_{k+1}^{\text{op}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}_k)\boldsymbol{x}_k$.

If N goes to infinity, then $\boldsymbol{X}_k$ would converge to a constant matrix for small k under the backward Riccati iteration. In this case the time subscript k can be removed from $\boldsymbol{X}_k$. This is the solution of DARE that solves the OCP (5.3).

We use the following symbol to denote the solver interface:

$$\boldsymbol{X} = \mathsf{dare}(n, m, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R}) \tag{5.5}$$

5.2.3 Quadratic Programming

We briefly talk about *quadratic programming (QP)* solved by *active-set method*, which is used in the target selector and MPC regulator.

QP solves the following minimization problem with quadratic objective function $f(\boldsymbol{w})$ and linear constraints:

$$\begin{aligned}
\min_{\boldsymbol{w}\in\mathbb{R}^n} \quad & f(\boldsymbol{w}) = \tfrac{1}{2}\boldsymbol{w}^\top \boldsymbol{H}\boldsymbol{w} + \boldsymbol{c}^\top \boldsymbol{w} \\
\text{subject to} \quad & \boldsymbol{h}(\boldsymbol{w}) = \boldsymbol{A}_{\text{eq}}\boldsymbol{w} + \boldsymbol{b}_{\text{eq}} = \boldsymbol{0} \\
& \boldsymbol{g}(\boldsymbol{w}) = \boldsymbol{A}\boldsymbol{w} + \boldsymbol{b} \le \boldsymbol{0}
\end{aligned} \tag{5.6a}$$

where $\boldsymbol{A}_{\text{eq}} \in \mathbb{R}^{m_{\text{eq}} \times n}, \boldsymbol{b}_{\text{eq}} \in \mathbb{R}^{m_{\text{eq}} \times 1}$ and $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^{m \times 1}$ correspond to equality/inequality constraints, respectively. The bold-face $\boldsymbol{h}, \boldsymbol{g}$ implies that they are vector-type function while the non-bold-face f implies that it's scalar function. Let $\boldsymbol{a}_{\text{eq},i}$ and $\boldsymbol{a}_i$ be the row of $\boldsymbol{A}_{\text{eq}}$ and $\boldsymbol{A}$, respectively. Use $\boldsymbol{a}_{\text{eq},i}$ and $\boldsymbol{a}_i$ to reformulate the problem:

$$\begin{aligned} \min_{\boldsymbol{w} \in \mathbb{R}^n} \quad & f(\boldsymbol{w}) = \tfrac{1}{2} \boldsymbol{w}^\top \boldsymbol{H} \boldsymbol{w} + \boldsymbol{c}^\top \boldsymbol{w} \\ \text{subject to} \quad & h_i(\boldsymbol{w}) = \boldsymbol{a}_{\text{eq},i}^\top \boldsymbol{w} + b_{\text{eq},i} = 0 \ \forall i \in E \\ & g_i(\boldsymbol{w}) = \boldsymbol{a}_i^\top \boldsymbol{w} + b_i \le 0 \ \forall i \in I \end{aligned} \tag{5.6b}$$

A unique solution (the minimum) for the QP problem exists when the first order necessary condition $\nabla_{\boldsymbol{w}} f(\boldsymbol{w}) = 0$ and the second order sufficient condition $\boldsymbol{H} = \nabla^2_{\boldsymbol{w}} f(\boldsymbol{w}) \succ 0$ hold.

Let

$$E = \{1, \cdots, m_{\text{eq}}\} \text{ and } I = \{1, \cdots, m\} \tag{5.7}$$

be the index set of equality/inequality constraints, respectively. The first step to solve it is by introducing Lagrange multipliers $\{\lambda_i\}_{i \in E}$, $\{\mu_i\}_{i \in I}$ and defining the Lagrangian

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{w}) + \sum_{i \in E} \lambda_i h_i(\boldsymbol{w}) + \sum_{i \in I} \mu_i g_i(\boldsymbol{w}) \tag{5.8}$$

The *KKT conditions* claims that if $\boldsymbol{w}^*$ is a global minimum of (5.6b), it satisfies

$$\begin{aligned} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= 0 \\ h_i(\boldsymbol{w}^*) &= 0 \ \forall i \in E \\ g_i(\boldsymbol{w}^*) &\le 0 \ \forall i \in I \\ \mu_i &\ge 0 \ \forall i \in I \\ \mu_i^* g_i(\boldsymbol{w}^*) &= 0 \ \forall i \in I \end{aligned} \tag{5.9}$$

The last three conditions are also called *complementary conditions*: for some $i \in I$,

$$\text{either} \quad \begin{cases} g_i(\boldsymbol{w}^*) = 0 \\ \mu_i > 0 \end{cases} \quad \text{or} \quad \begin{cases} g_i(\boldsymbol{w}^*) < 0 \\ \mu_i = 0 \end{cases} \tag{5.10}$$

holds when $\boldsymbol{w}^*$ is found. Noting this fact, we introduce the *active set*

$$A = \{i \in I | g_i(\boldsymbol{w}) = 0\} \tag{5.11}$$

The complementary conditions (5.10) tells us that either $\{\mu_i\}_{i \in A} > 0$ (active) or $\{\mu_i\}_{i \notin A} = 0$ (inactive). In the optimal state, the inactive inequality constraints vanish and only the active inequality constraints are left:

$$0 = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \nabla_{\boldsymbol{w}} f(\boldsymbol{w}^*) + \sum_{i \in E} \lambda_i \nabla_{\boldsymbol{w}} h_i(\boldsymbol{w}^*) + \sum_{i \in A} \mu_i \nabla_{\boldsymbol{w}} g_i(\boldsymbol{w}^*)$$

The inequality constraints $\{g_i\}_{i \in A}$ can be viewed as equality constraints and they are said to be "*active*". So the problem can be further decomposed to the following steps: (1) determine the active set A; (2) solve a QP with equality constraints only.

QP with only equality constraints

The inequality constraints $\boldsymbol{g}/g_i$ in (5.6a/5.6b) are treated absented in this subsection. The KKT condition (5.9) then remains the first two lines and we yield

$$\begin{aligned} \boldsymbol{H}\boldsymbol{w}^* + \boldsymbol{c} + \boldsymbol{A}_{\text{eq}}^\top \boldsymbol{\lambda} &= 0 \\ \boldsymbol{A}_{\text{eq}}\boldsymbol{w}^* + \boldsymbol{b}_{\text{eq}} &= \boldsymbol{0} \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} \boldsymbol{H} & \boldsymbol{A}_{\text{eq}}^\top \\ \boldsymbol{A}_{\text{eq}} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{w}^* \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{c} \\ -\boldsymbol{b}_{\text{eq}} \end{pmatrix}$$

The block matrix $\left(\begin{smallmatrix} \boldsymbol{H} & \boldsymbol{A}_{\text{eq}}^\top \\ \boldsymbol{A}_{\text{eq}} & \boldsymbol{0} \end{smallmatrix}\right)$ is called *KKT matrix*. With proposition 2.8.7 of [Ber09],

$$\begin{pmatrix} \boldsymbol{H} & \boldsymbol{A}_{\text{eq}}^\top \\ \boldsymbol{A}_{\text{eq}} & \boldsymbol{0} \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{H}^{-1} - \boldsymbol{H}^{-1}\boldsymbol{A}_{\text{eq}}^\top(\boldsymbol{A}_{\text{eq}}\boldsymbol{H}^{-1}\boldsymbol{A}_{\text{eq}}^\top)^{-1}\boldsymbol{A}_{\text{eq}}\boldsymbol{H}^{-1} & \boldsymbol{H}^{-1}\boldsymbol{A}_{\text{eq}}^\top(\boldsymbol{A}_{\text{eq}}\boldsymbol{H}^{-1}\boldsymbol{A}_{\text{eq}}^\top)^{-1} \\ (\boldsymbol{A}_{\text{eq}}\boldsymbol{H}^{-1}\boldsymbol{A}_{\text{eq}}^\top)^{-1}\boldsymbol{A}_{\text{eq}}\boldsymbol{H}^{-1} & -(\boldsymbol{A}_{\text{eq}}\boldsymbol{H}^{-1}\boldsymbol{A}_{\text{eq}}^\top)^{-1} \end{pmatrix}$$

The two off-diagonal matrices are transpose to each other, thanks to the nice property of the Hessian — $(\boldsymbol{H}^{-1})^\top = (\boldsymbol{H}^\top)^{-1}$. A special case worth mentioned is that when the equality constraints are also vanished, we directly yield $\boldsymbol{w}^* = -\boldsymbol{H}^{-1}\boldsymbol{c}$.

QP with inequality constraints

We use $A^{(k)}$ to denote the active set at the iteration stage k since during finding the global minimum, the active inequality constraints $\{g_i\}_{i\in A^{(k)}}$ vary. Since the active inequality constraints $\{g_i\}_{i\in A^{(k)}}$ can be viewed as equality constraints $\{h_i\}_{i\in E}$, we introduce a concept *working set* and/or *effective set*

$$W^{(k)} = E \cup A^{(k)} \tag{5.12}$$

The equality constrained QP problem in this context is

$$\begin{aligned} \min_{w\in\mathbb{R}^n} \quad & \tfrac{1}{2}\boldsymbol{w}^\top \boldsymbol{H}\boldsymbol{w} + \boldsymbol{c}^\top \boldsymbol{w} \\ \text{subject to} \quad & \boldsymbol{a}_{\text{eff},i}^\top \boldsymbol{w} + b_{\text{eff},i} = 0 \ \ \forall i \in W^{(k)} \\ \text{where} \quad & \{\boldsymbol{a}_{\text{eff},i}^\top\}_{i\in W^{(k)}} = \begin{pmatrix} \{\boldsymbol{a}_{\text{eq},i}^\top\}_{i\in E} \\ \{\boldsymbol{a}_i^\top\}_{i\in A^{(k)}} \end{pmatrix}, \ \{b_{\text{eff},i}\}_{i\in W^{(k)}} = \begin{pmatrix} \{b_{\text{eq},i}\}_{i\in E} \\ \{b_i\}_{i\in A^{(k)}} \end{pmatrix} \end{aligned} \tag{5.13}$$

If the initial state $\boldsymbol{w}^{(0)}$ is specified, it must satisfies $\boldsymbol{A}_{\text{eq}}\boldsymbol{w}^{(0)} + \boldsymbol{b}_{\text{eq}} = \boldsymbol{0}$. If it's not provided (w.l.o.g, assumed $\boldsymbol{0} \in \mathbb{R}^n$), it can still be obtained from $\boldsymbol{w}^{(0)} = -\boldsymbol{A}_{\text{eq}}^+ \boldsymbol{b}_{\text{eq}}$, where $\boldsymbol{A}_{\text{eq}}^+ \in \mathbb{R}^{n\times m_{\text{eq}}}$ is the *pseudo-inverse* of $\boldsymbol{A}_{\text{eq}} \in \mathbb{R}^{m_{\text{eq}}\times n}$. Then, we calculate $g_i\left(\boldsymbol{w}^{(0)}\right) = \boldsymbol{a}_i^\top \boldsymbol{w}^{(0)} + b_i$ for $\forall i \in I$. The i's that give $-\epsilon < g_i\left(\boldsymbol{w}^{(0)}\right)$ are added to $A^{(0)}$, thus obtain the initial working set $W^{(0)}$.

We then talk about how to update from $\boldsymbol{w}^{(k)}$ to $\boldsymbol{w}^{(k+1)}$ using the Newton-type state-update rule. Let $\boldsymbol{p}^{(k)} := \Delta\boldsymbol{w}^{(k)}$ be the step size, and modify the linear terms in (5.13):

$$\begin{aligned} \min_{w\in\mathbb{R}^n} \quad & \tfrac{1}{2}\boldsymbol{p}^{(k)\top} \boldsymbol{H}\boldsymbol{p}^{(k)} + \tilde{\boldsymbol{c}}^\top \boldsymbol{p}^{(k)} \\ \text{subject to} \quad & \boldsymbol{a}_{\text{eff},i}^\top \boldsymbol{p}^{(k)} + \tilde{b}_{\text{eff},i} = 0 \ \ \forall i \in W^{(k)} \\ \text{where} \quad & \tilde{\boldsymbol{c}} = \boldsymbol{H}\boldsymbol{w}^{(k)} + \boldsymbol{c} \\ & \tilde{b}_{\text{eff},i} = \boldsymbol{a}_{\text{eff},i}^\top \boldsymbol{w}^{(k)} + b_{\text{eff},i} \ \ \forall i \in W^{(k)} \end{aligned} \tag{5.14}$$

This is an equality constrained QP encountered before. The resulting $\boldsymbol{p}^{(k)}$ could be 0 (in practice, it's absolute value is smaller than a significantly small number) or else. By solving the equality constrained QP we also get $\{\lambda_i^{(k)}\}_{i\in E}$ and $\{\mu_i^{(k)}\}_{i\in A^{(k)}}$.

$\boldsymbol{p}^{(k)} = 0$ implies it's *possibly* need not to perform state-update — however, we need further check the KKT conditions for the resulting $\{\mu_i^{(k)}\}_{i\in A^{(k)}}$. $\{\mu_i^{(k)}\}_{i\in A^{(k)}} > 0$ implies that the current $\boldsymbol{w}^{(k)}$ is indeed the minimum state and stop iteration; else, we remove the index of the smallest μ_i from $W^{(k)}$, to obtain $W^{(k+1)}$, and begin the next iteration.

$\boldsymbol{p}^{(k)} \neq 0$ naturally implies that a state-update is required, but how to update the state remains to see; generally, let $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \alpha^{(k)}\boldsymbol{p}^{(k)}$ and $\alpha^{(k)} \in [0, 1]$. It turns out that how to treat the "inactive" indices $i \in (E \cup I)\backslash W^{(k)}$ (or say, $i \notin W^{(k)}$). The inactive constraints stay inactive in the $k+1$ iteration are $\boldsymbol{a}_i^\top \boldsymbol{w}^{(k+1)} + b_i < \boldsymbol{a}_i^\top \boldsymbol{w}^{(k)} + b_i < 0$. If $\boldsymbol{a}_i^\top \boldsymbol{p}^{(k)} < 0$, then $\alpha^{(k)} > -(\boldsymbol{a}_i^\top \boldsymbol{w}^{(k)} + b_i)/(\boldsymbol{a}_i^\top \boldsymbol{p}^{(k)})$; however, *larger* than a number would possibly exceed the upper limit 1, implying a "too large" update-size, so gives no advice on $\alpha^{(k)}$. If $\boldsymbol{a}_i^\top \boldsymbol{p}^{(k)} > 0$, then

$$\alpha^{(k)} < -\frac{\boldsymbol{a}_i^\top \boldsymbol{w}^{(k)} + b_i}{\boldsymbol{a}_i^\top \boldsymbol{p}^{(k)}} \quad (i \notin W^{(k)})$$

If the right hand side is smaller than 1, then the Newton-type state-update $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \alpha^{(k)}\boldsymbol{p}^{(k)}$ becomes smaller/smoother than the "stiff" $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \boldsymbol{p}^{(k)}$. In summary, we calculate $\alpha^{(k)}$ as

$$\alpha^{(k)} \overset{\text{def}}{=} \min\left(1, \min_{i\notin W^{(k)}, \boldsymbol{a}_i^\top \boldsymbol{p}^{(k)} > 0} -\frac{\boldsymbol{a}_i^\top \boldsymbol{w}^{(k)} + b_i}{\boldsymbol{a}_i^\top \boldsymbol{p}^{(k)}}\right) \tag{5.15}$$

An extreme case is that when $\alpha^{(k)}$ approaches 0. In this case, we can regard the inequality constraint as "active", and add it to $W^{(k+1)}$.

The algorithm of QP solved by active-set method is shown in Algorithm 23. We use the following symbol to denote the solver interface:

$$\boldsymbol{w}^* = \mathsf{quadprog}\left(n, m, m_{\text{eq}}, \boldsymbol{H}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{A}_{\text{eq}}, \boldsymbol{b}_{\text{eq}}, \boldsymbol{w}^{(0)}\right) \tag{5.16}$$

If the corresponding constraints are absent, use NULL in the corresponding argument input.

5.3 Plant Dynamics: CSTR Model as a Case Study

A typical control system, with system state $\boldsymbol{x} \in \mathbb{R}^{n_x}$, control $\boldsymbol{u} \in \mathbb{R}^{n_u}$ and parameter $\boldsymbol{p} \in \mathbb{R}^{n_p}$, is described by an ODE

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) \tag{5.17}$$

Provided the stationary state $\boldsymbol{x}^s$ and the corresponding control $\boldsymbol{u}^s$ and parameter $\boldsymbol{p}^s$ are known, its Jacobian of the state $\boldsymbol{x}$ is $\partial_{\boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s) \in \mathbb{R}^{n_x \times n_x}$ and the Jacobian of the control $\boldsymbol{u}$ is $\partial_{\boldsymbol{u}}\boldsymbol{f}(\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s) \in \mathbb{R}^{n_x \times n_u}$. Obtaining the state matrix $\boldsymbol{A}$ and control matrix $\boldsymbol{B}$,

Algorithm 23 Active-set method for convex QP

Input: $n, m, m_{\text{eq}}, \boldsymbol{H}, \boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{A}_{\text{eq}}, \boldsymbol{b}_{\text{eq}}$; optional: $\boldsymbol{w}^{(0)}$
Output: $\boldsymbol{w}^*$

Initialize $W^{(0)}$ by $\boldsymbol{w}^{(0)}$
for $k = 0, 1, 2, \cdots$ **do**
  Solve (5.14) to find $\boldsymbol{p}^{(k)}$ and $\{\mu_i^{(k)}\}_{i \in A^{(k)}}$
  if $\boldsymbol{p}^{(k)} = 0$ **then**
    if $\mu_i > 0$ for all $i \in A^{(k)}$ **then**
      Stop with solution $\boldsymbol{w}^{(k)}$
    else
      $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)}$
      $j = \text{argmin}_{i \in A^{(k)}} \mu_i$, $W^{(k+1)} = W^{(k)} \backslash \{j\}$
    end if
  else ($\boldsymbol{p}^{(k)} \neq 0$)
    Calculate $\alpha^{(k)}$ from (5.15) and $j = \text{argmin}_{i \notin W^{(k)}, \boldsymbol{a}_i^\top \boldsymbol{p}^{(k)} > 0} - \frac{\boldsymbol{a}_i^\top \boldsymbol{w}^{(k)} + b_i}{\boldsymbol{a}_i^\top \boldsymbol{p}^{(k)}}$
    $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \alpha^{(k)} \boldsymbol{p}^{(k)}$
    if $\alpha^{(k)} < 1$ **then**
      $W^{(k+1)} = W^{(k)} \cap \{j\}$
    end if
  end if
end for

as that in the previous text, is called *relinearization procedure*. We use *matrix exponential* to do so. To this end, we first construct an augmented matrix

$$\boldsymbol{J}_{\text{aug}} = \begin{bmatrix} \partial_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s) & \boldsymbol{I}_{n_x} \\ \epsilon \mathbb{1}_{n_x \times n_x} & \epsilon \mathbb{1}_{n_x \times n_x} \end{bmatrix} \in \mathbb{R}^{2n_x \times 2n_x} \tag{5.18}$$

Patching $\epsilon \mathbb{1}_{n_x \times n_x}$ is in order to make $\boldsymbol{J}_{\text{aug}}$ full rank when calculating matrix exponential. The linearized model matrices are then obtained by slicing

$$\begin{aligned} \boldsymbol{A} &= \text{expm}\,(\boldsymbol{J}_{\text{aug}} \Delta t)\,[1 : n_x, 1 : n_x], \\ \boldsymbol{B} &= \text{expm}\,(\boldsymbol{J}_{\text{aug}} \Delta t)\,[1 : n_x, n_x + 1 : n_x + n_u] \partial_{\boldsymbol{u}} \boldsymbol{f}(\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s) \end{aligned} \tag{5.19}$$

With the current state $\boldsymbol{x}(k)$ to get the next state $\boldsymbol{x}(k+1)$ via (5.2) is called *actuate*:

$$\boldsymbol{x}(k+1) = \mathsf{actuate}\,(n_x, n_u, n_p, \boldsymbol{x}(k), \boldsymbol{u}(k), \boldsymbol{p}(k), \boldsymbol{f}, \partial_{\boldsymbol{x}} \boldsymbol{f}) \tag{5.20}$$

As a case study of a plant dynamics, we consider a CSTR model as in Pannocchia and Rawlings (2003) [PR03] and [RMD17]. The model is described by the ODE:

$$\begin{aligned}\frac{\mathrm{d}c}{\mathrm{d}t} &= \frac{F_0(c_0 - c)}{\pi r^2 h} - k_0 \exp\left(-\frac{E}{RT}\right) c \\ \frac{\mathrm{d}T}{\mathrm{d}t} &= \frac{F_0(T_0 - T)}{\pi r^2 h} + \frac{-\Delta H}{\rho c_p} k_0 \exp\left(-\frac{E}{RT}\right) c + \frac{2U}{r\rho c_p}(T_c - T) \\ \frac{\mathrm{d}h}{\mathrm{d}t} &= \frac{F_0 - F}{\pi r^2}\end{aligned} \tag{5.21}$$

The state variables are the molar concentration c of species, the reactor temperature T, the level h of the tank. c and h is the controlled variables and observable, while T is not observable, which is classified as "additional state variable". The manipulated variables are the coolant liquid temperature T_c and the outlet flow rate F. The inlet flow rate F_0 acts as an unmeasured disturbance. The model parameters in nominal conditions are shown in Table 5.1. The open-loop stable steady state operating conditions are the following:

$$\begin{gathered} c^s = 0.878 \text{ kmol/m}^3 \quad T^s = 324.5\text{K} \quad h^s = 0.659 \text{ m} \\ T_c^s = 300 \text{ K} \quad F^s = 0.1 \text{ m}^3/\text{min} \end{gathered}$$

Using a sampling time of 1 min, a linearized discrete state space model is obtain near the stationary points

$$\boldsymbol{x} = \begin{bmatrix} c - c^s \\ T - T^s \\ h - h^s \end{bmatrix}, \boldsymbol{u} = \begin{bmatrix} T_c - T_c^s \\ F - F^s \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} c - c^s \\ T - T^s \\ h - h^s \end{bmatrix}, p = F_0 - F_0^s$$

Table 5.1: Parameters of the well-stirred reactor.

Parameter	Nominal value	Units	Meaning
F_0	0.1	m^3/min	inlet flow rate
T_0	350	K^3/min	feed temperature
c_0	1	kmol/m^3	feed concentration
r	0.219	m	radius of the container
k_0	7.2×10^{10}	min^{-1}	Pre-exponential factor
E/R	8750	K	the activation energy E devided by the universal ideal gas constant R
U	54.94	$\text{kJ/(min·m}^2\text{·K)}$	overall heat transfer coefficient
ρ	1000	kg/m^3	density of the species
c_p	0.239	kJ/(kg·K)	heat capacity of the species
ΔH	-5×10^4	kJ/kmol	heat of the reaction

5.4 Estimator

The estimator will be only briefly discussed as we mainly focus on the MPC regulation problem instead of the estimation problem. However, this is necessary so that we can

compare our result with [RMD17]. This section is mainly organized from Section 1.5.2 in [RMD17] and the CSTR model being studied and implemented. The estimator experiences observation and unmeasured disturbance, usually with offset. To model the influence of the offset we consider the following augmented system for the state estimator:

$$\begin{aligned}\begin{bmatrix}\hat{\boldsymbol{x}}\\ \hat{\boldsymbol{d}}\end{bmatrix}^{+} &= \underbrace{\begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}_d\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}}_{\boldsymbol{A}_{\text{aug}}\in\mathbb{R}^{n_a\times n_a}}\begin{bmatrix}\hat{\boldsymbol{x}}\\ \hat{\boldsymbol{d}}\end{bmatrix} + \underbrace{\begin{bmatrix}\boldsymbol{B}\\ \boldsymbol{0}\end{bmatrix}}_{\boldsymbol{B}_{\text{aug}}\in\mathbb{R}^{n_a\times n_u}}\boldsymbol{u} + \boldsymbol{w}_d\\ \boldsymbol{y} &= \underbrace{\begin{bmatrix}\boldsymbol{C} & \boldsymbol{C}_d\end{bmatrix}}_{\boldsymbol{C}_{\text{aug}}\in\mathbb{R}^{n_y\times n_a}}\begin{bmatrix}\hat{\boldsymbol{x}}\\ \hat{\boldsymbol{d}}\end{bmatrix} + \boldsymbol{v}_d\end{aligned} \tag{5.22}$$

Here, $n_y = n_x$, as in many cases do, $\boldsymbol{C} = \boldsymbol{I}_{n_y}$, $\boldsymbol{y}$ is the sloppy of $\boldsymbol{y}(k)$ and $\boldsymbol{y}(k) = \boldsymbol{C}\boldsymbol{x}(k)$, with $\boldsymbol{x}(k)$ is obtained from (5.20). $\hat{\boldsymbol{d}}$ is the modeled disturbance, and $\boldsymbol{B}_d, \boldsymbol{C}_d$ depend on the disturbance model; $\boldsymbol{w}_d$ and $\boldsymbol{v}_d$ are white noise, which can be modeled by Kalman filter; the augmented state dimension is $n_a = n_x + n_d$. We use hat over $\boldsymbol{x}$ and $\boldsymbol{d}$ to denote the "measurement" nature, since *the situation of applying estimator is performing the measurement to the plant status* (recall Figure 5.1). $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{d}}$ are the pure measurement value, $\hat{\boldsymbol{x}}^{+}$ and $\hat{\boldsymbol{d}}^{+}$ is the counterpart after the Kalman filter. Note that measurement would introduce disturbance, so *the central task of the estimator is to remove the disturbance during measurement.* This task is accomplished by Kalman filter.

We only give the result of the Kalman filter without derivation under the simplest case. For the CSTR model we design the weighting matrices motivated by the covariance matrix in probability theory:

$$\begin{aligned}\boldsymbol{Q}_{\boldsymbol{w}} &= \text{diag}(\epsilon, \cdots, \epsilon, 1) \in \mathbb{R}^{(n_x+n_d)\times(n_x+n_d)}\\ \boldsymbol{R}_{\boldsymbol{v}} &= \text{diag}(\epsilon\boldsymbol{x}_1^{s2}, \cdots, \epsilon\boldsymbol{x}_{n_x}^{s2}) \in \mathbb{R}^{n_y\times n_y}\end{aligned} \tag{5.23}$$

It's reasonable that the diagonal elements are all zero since c, T, h are different quantities in the physics sense. We firstly solve the DARE to obtain $\boldsymbol{X} \in \mathbb{R}^{n_a\times n_a}$:

$$\boldsymbol{X} = \mathsf{dare}(n_a, n_y, \boldsymbol{A}_{\text{aug}}^{\top}, \boldsymbol{C}_{\text{aug}}^{\top}, \boldsymbol{Q}_{\boldsymbol{w}}, \boldsymbol{R}_{\boldsymbol{v}})$$

Then the Kalman filter in our simplest case is

$$\boldsymbol{L} = \begin{bmatrix}\boldsymbol{L}_{\boldsymbol{x}}\\ \boldsymbol{L}_{\boldsymbol{d}}\end{bmatrix} = \boldsymbol{X}\boldsymbol{C}_{\text{aug}}^{\top}\left(\boldsymbol{C}_{\text{aug}}X\boldsymbol{C}_{\text{aug}}^{\top} + \boldsymbol{R}\right)^{-1} \in \mathbb{R}^{n_a\times n_y} \tag{5.24}$$

This is the simplest case of a linear-quadratic estimator. We use the following symbol to denote the solver interface:

$$\begin{bmatrix}\boldsymbol{L}_{\boldsymbol{x}}\\ \boldsymbol{L}_{\boldsymbol{d}}\end{bmatrix} = \mathsf{dlqe}\left(n_y, n_x + n_d, \boldsymbol{A}_{\text{aug}}, \boldsymbol{C}_{\text{aug}}, \boldsymbol{Q}_{\boldsymbol{w}}, \boldsymbol{R}_{\boldsymbol{v}}\right) \tag{5.25}$$

By applying the Kalman filter $\left[\begin{smallmatrix}\boldsymbol{L}_{\boldsymbol{x}}\\ \boldsymbol{L}_{\boldsymbol{d}}\end{smallmatrix}\right]$ to the measurement disturbance $\boldsymbol{w}_d$ in (5.22) we get

$$\begin{bmatrix}\hat{\boldsymbol{x}}\\ \hat{\boldsymbol{d}}\end{bmatrix}^{+} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}_d\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\hat{\boldsymbol{x}}\\ \hat{\boldsymbol{d}}\end{bmatrix} + \begin{bmatrix}\boldsymbol{B}\\ \boldsymbol{0}\end{bmatrix}\boldsymbol{u} + \begin{bmatrix}\boldsymbol{L}_{\boldsymbol{x}}\\ \boldsymbol{L}_{\boldsymbol{d}}\end{bmatrix}\left(\boldsymbol{y} - \begin{bmatrix}\boldsymbol{C} & \boldsymbol{C}_d\end{bmatrix}\begin{bmatrix}\hat{\boldsymbol{x}}\\ \hat{\boldsymbol{d}}\end{bmatrix}\right) \tag{5.26}$$

We use the following symbol to denote the estimation procedure:

$$\begin{bmatrix} \hat{\boldsymbol{x}}^+ \\ \hat{\boldsymbol{d}}^+ \end{bmatrix} = \mathsf{estimate}\left(n_x, n_u, n_d, n_y, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{B}_d, \boldsymbol{C}_d, \begin{bmatrix} \boldsymbol{L}_{\boldsymbol{x}} \\ \boldsymbol{L}_{\boldsymbol{d}} \end{bmatrix}, \boldsymbol{y}, \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{d}} \end{bmatrix} \right) \tag{5.27}$$

The time step argument "(k)" after each state is ignored since all of them are at the same time step k.

5.5 Target Selector

Target selection is used to select the reference trajectory / steady-state target for MPC regulation. We use $(\boldsymbol{x}_r, \boldsymbol{u}_r)$ to denote the reference trajectory instead of using $(\boldsymbol{x}_s, \boldsymbol{u}_s)$ in case of symbol abusing. It's expected that in the stationary state,

$$\begin{bmatrix} \boldsymbol{I} - \boldsymbol{A} & -\boldsymbol{B} \\ \boldsymbol{H}_r \boldsymbol{C} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_r \\ \boldsymbol{u}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{r}_{\text{sp}} \end{bmatrix} \tag{5.28}$$

where $\boldsymbol{r}_{\text{sp}}$ is the *set-point* of the observable controlled variables. In our CSTR model, since c and h are controlled variable while T is not, so to get $\boldsymbol{r}_{\text{sp}} = \boldsymbol{H}_r \boldsymbol{y}_r$, we just simply design

$$\boldsymbol{H}_r = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.29}$$

This consideration is useful when treating the *constrained problem*, though we do not consider this case for simplicity in this project.

We then turn to the problem of determining $\boldsymbol{x}_r$ and $\boldsymbol{u}_r$. This is called *steady-state target problem*. In the unconstrained case, this is simply solved by inverting the matrix $\left[\begin{smallmatrix} \boldsymbol{I}-\boldsymbol{A} & -\boldsymbol{B} \\ \boldsymbol{H}_r\boldsymbol{C} & \boldsymbol{0} \end{smallmatrix}\right]$ in (5.28). Due to the fact that in most cases we set $\boldsymbol{r}_{\text{sp}} = \boldsymbol{0}$ and thus $\boldsymbol{x}_r, \boldsymbol{u}_r = 0$, considering target selection is a trivial problem, to some extent. However, if consider the disturbance in the estimator, problem would become slightly complex.

If disturbance is included into the consideration, the equation that describing the stationary state (5.28) would also changed to

$$\begin{bmatrix} \boldsymbol{I} - \boldsymbol{A} & -\boldsymbol{B} \\ \boldsymbol{H}_r \boldsymbol{C} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_r \\ \boldsymbol{u}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_d \hat{\boldsymbol{d}}^+ \\ \boldsymbol{r}_{\text{sp}} - \boldsymbol{H}_r \boldsymbol{C}_d \hat{\boldsymbol{d}}^+ \end{bmatrix} \tag{5.30}$$

In the unconstrained case, we need not consider the quadratic optimization problem, and thus the reference trajectory is simply determined by invert the matrix $\left[\begin{smallmatrix} \boldsymbol{I}-\boldsymbol{A} & -\boldsymbol{B} \\ \boldsymbol{H}_r\boldsymbol{C} & \boldsymbol{0} \end{smallmatrix}\right]$. We use the following symbol to denote the target selection procedure:

$$\begin{bmatrix} \boldsymbol{x}_r \\ \boldsymbol{u}_r \end{bmatrix} = \mathsf{select}\left(n_x, n_u, n_d, n_y, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{B}_d, \boldsymbol{C}_d, \boldsymbol{H}_r, \boldsymbol{r}_{\text{sp}}, \hat{\boldsymbol{d}}^+ \right) \tag{5.31}$$

5.6 MPC Regulator

The MPC regulator receives $\boldsymbol{x}_r(k), \boldsymbol{u}_r(k), \hat{\boldsymbol{x}}^+(k), \boldsymbol{u}(k-1)$ as input to decide the optimal $\boldsymbol{u}(k)$. As discussed in Section 5.4, $\hat{\boldsymbol{x}}^+$ is interpreted as the measured and (Kalman) filtered state at simulation time k. In the context of regulator, during which the measurement disturbance is not the focusing point, we can remove the superscript "+" in this section.

Let $\hat{\boldsymbol{x}}(k+n|k)$ be the prediction of value $\hat{\boldsymbol{x}}(k)$ at time instance $k+n$ starting at current time step k, and $n = 0, 1, \cdots, N$, $N < \infty$ being the predictive horizon. From the predictive sequence

$$\begin{aligned} \hat{\boldsymbol{x}}(k+n|k) =& \boldsymbol{A}\hat{\boldsymbol{x}}(k+n-1) + \boldsymbol{B}\boldsymbol{u}(k+n-1|k) && n = 1, \cdots, N \\ \boldsymbol{u}(k+n|k) =& \boldsymbol{u}(k+n-1|k) + \Delta\boldsymbol{u}(k+n|k) && n = 1, \cdots, N-1 \\ \boldsymbol{u}(k|k) =& \boldsymbol{u}(k-1) + \Delta\boldsymbol{u}(k|k) && n = 0 \end{aligned} \tag{5.32}$$

we find that the control increment $\{\Delta\boldsymbol{u}(k+n|k)\}_{n=0}^{N-1}$ are independent and adjustable quantities, while the current state $\hat{\boldsymbol{x}}(k)$ and the last control $\boldsymbol{u}(k-1)$ are immutable, so (5.32) can be reformulated as

$$\begin{aligned} \hat{\boldsymbol{x}}(\cdot|k) =& \begin{bmatrix} \hat{\boldsymbol{x}}(k|k) \\ \hat{\boldsymbol{x}}(k+1|k) \\ \vdots \\ \hat{\boldsymbol{x}}(k+N|k) \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^N \end{bmatrix}}_{\mathbf{A}\in\mathbb{R}^{n_x(N+1)\times n_x}} \hat{\boldsymbol{x}}(k) + \underbrace{\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{B} \\ \boldsymbol{A}\boldsymbol{B} \\ \vdots \\ \sum_{n=0}^{N-1}\boldsymbol{A}^n\boldsymbol{B} \end{bmatrix}}_{\mathbf{B}\in\mathbb{R}^{n_x(N+1)\times n_u}} \boldsymbol{u}(k-1) \\ &+ \underbrace{\begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{B} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \sum_{n=0}^{N-1}\boldsymbol{A}^n\boldsymbol{B} & \sum_{n=0}^{N-2}\boldsymbol{A}^n\boldsymbol{B} & \cdots & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} \end{bmatrix}}_{\boldsymbol{\Theta}\in\mathbb{R}^{n_x(N+1)\times n_u N}} \underbrace{\begin{bmatrix} \Delta\boldsymbol{u}(k|k) \\ \Delta\boldsymbol{u}(k+1|k) \\ \vdots \\ \Delta\boldsymbol{u}(k+N-1|k) \end{bmatrix}}_{\Delta\hat{\boldsymbol{u}}(\cdot|k)\in\mathbb{R}^{n_u N}} \\ =& \mathbf{A}\hat{\boldsymbol{x}}(k) + \mathbf{B}\boldsymbol{u}(k-1) + \boldsymbol{\Theta}\Delta\boldsymbol{u}(\cdot|k) \in \mathbb{R}^{n_x(N+1)} \\ \boldsymbol{y}(\cdot|k) =& \mathbf{C}\hat{\boldsymbol{x}}(\cdot|k) \in \mathbb{R}^{n_y(N+1)}, \quad \mathbf{C} \coloneqq \operatorname{diag}(\boldsymbol{C}, \boldsymbol{C}, \cdots, \boldsymbol{C}) \in \mathbb{R}^{n_y(N+1)\times n_x(N+1)} \end{aligned} \tag{5.33}$$

(5.33) utilizes $\hat{\boldsymbol{x}}(k), \boldsymbol{u}(k-1)$ in the inputs, and leaves $\{\Delta\boldsymbol{u}(k+n|k)\}_{n=0}^{N-1}$ to be optimized. We now turn to the reference state $\boldsymbol{x}_r(k), \boldsymbol{u}_r(k)$, which are also immutable over the predictive horizon. With $\mathbf{A}, \mathbf{B}$ defined in (5.33), we have an immutable the predictive reference state

$$\boldsymbol{x}_r(\cdot|k) = \mathbf{A}\boldsymbol{x}_r(k) + \mathbf{B}\boldsymbol{u}_r(k) \in \mathbb{R}^{n_x(N+1)}, \ \boldsymbol{y}_r(\cdot|k) = \mathbf{C}\boldsymbol{x}_r(\cdot|k) \in \mathbb{R}^{n_y(N+1)}$$

so the deviation of $\boldsymbol{y}(\cdot|k)$ from the reference trajectory $\boldsymbol{y}_r(\cdot|k)$ is

$$\begin{aligned} \boldsymbol{y}(\cdot|k) - \boldsymbol{y}_r(\cdot|k) =& \underbrace{\mathbf{C}\left(\mathbf{A}(\hat{\boldsymbol{x}}(k) - \boldsymbol{x}_r(k)) + \mathbf{B}(\boldsymbol{u}(k-1) - \boldsymbol{u}_r(k))\right)}_{\boldsymbol{e}(\cdot|k)\in\mathbb{R}^{n_y(N+1)}} + \mathbf{C}\Theta\Delta\boldsymbol{u}(\cdot|k) \\ =& \boldsymbol{e}(\cdot|k) + \mathbf{C}\Theta\Delta\boldsymbol{u}(\cdot|k) \end{aligned} \tag{5.34}$$

$\boldsymbol{e}(\cdot|k)$ is so-called *free error*, meaning that it's constant over the predictive horizon. So to minimize the deviation $\boldsymbol{y}(\cdot|k) - \boldsymbol{y}_r(\cdot|k)$ turns out to optimize $\Delta\boldsymbol{u}(\cdot|k)$, implying that an objective function should be set up to solve this OCP. To this end, define weighting matrices

$$\begin{aligned}\boldsymbol{Q} =& \operatorname{diag}\left((\boldsymbol{x}_1^s)^{-2}, \cdots, (\boldsymbol{x}_{n_x}^s)^{-2}\right) \in \mathbb{R}^{n_x \times n_x},\ \mathbf{Q} \coloneqq \operatorname{diag}(\boldsymbol{Q}, \boldsymbol{Q}, \cdots, \boldsymbol{Q}) \in \mathbb{R}^{n_y(N+1) \times n_y(N+1)} \\ \boldsymbol{R} =& \operatorname{diag}\left((\boldsymbol{u}_1^s)^{-2}, \cdots, (\boldsymbol{u}_{n_u}^s)^{-2}\right) \in \mathbb{R}^{n_u \times n_u},\ \mathbf{R} \coloneqq \operatorname{diag}(\boldsymbol{R}, \boldsymbol{R}, \cdots, \boldsymbol{R}) \in \mathbb{R}^{n_u N \times n_u N}\end{aligned} \tag{5.35}$$

here, $\mathbf{Q}, \mathbf{R}$ are horizon extension of $\boldsymbol{Q}, \boldsymbol{R}$. The objective function is

$$\begin{aligned}J =& \|\boldsymbol{y}(\cdot|k) - \boldsymbol{y}_r(\cdot|k)\|_{\mathbf{Q}}^2 + \|\Delta\boldsymbol{u}(\cdot|k)\|_{\mathbf{R}}^2 \\ =& \|\boldsymbol{e}(\cdot|k) + \mathbf{C\Theta}\Delta\boldsymbol{u}(\cdot|k)\|_{\mathbf{Q}}^2 + \|\Delta\boldsymbol{u}(\cdot|k)\|_{\mathbf{R}}^2 \\ =& \Delta\boldsymbol{u}(\cdot|k)^\top \underbrace{\left(\mathbf{\Theta}^\top \mathbf{C}^\top \mathbf{Q} \mathbf{C} \mathbf{\Theta} + \mathbf{R}\right)}_{\boldsymbol{H} \in \mathbf{R}^{n_u N \times n_u N}} \Delta\boldsymbol{u}(\cdot|k) + 2\Delta\boldsymbol{u}(\cdot|k)^\top \underbrace{\left(\mathbf{\Theta}^\top \mathbf{C}^\top \mathbf{Q} \boldsymbol{e}(\cdot|k)\right)}_{\boldsymbol{c} \in \mathbf{R}^{n_u N}} + \underbrace{\boldsymbol{e}(\cdot|k)^\top \mathbf{Q} \boldsymbol{e}(\cdot|k)}_{\text{const}} \\ =& 2\left[\tfrac{1}{2}\Delta\boldsymbol{u}(\cdot|k)^\top \boldsymbol{H} \Delta\boldsymbol{u}(\cdot|k) + \boldsymbol{c}^\top \Delta\hat{\boldsymbol{u}}(\cdot|k) + \text{const}\right]\end{aligned}$$

We only need to minimize the non-constant terms:

$$\min_{\Delta\boldsymbol{u}(\cdot|k)} J = \tfrac{1}{2}\Delta\boldsymbol{u}(\cdot|k)^\top \boldsymbol{H} \Delta\boldsymbol{u}(\cdot|k) + \boldsymbol{c}^\top \Delta\boldsymbol{u}(\cdot|k) \tag{5.36}$$

The advantage of separating $\Delta\boldsymbol{u}(\cdot|k)$ out is that we can cast the MPC regulation problem into a QP problem. By solving the QP problem we obtained the optimal $\Delta\boldsymbol{u}(\cdot|k)^*$, i.e. the sequence $\Delta\boldsymbol{u}(k|k)^*, \Delta\boldsymbol{u}(k+1|k)^*, \cdots, \Delta\boldsymbol{u}(k+N-1|k)^*$. However, we only take the first one - $\Delta\boldsymbol{u}(k|k)^*$, then $\boldsymbol{u}(k) = \boldsymbol{u}(k|k) = \boldsymbol{u}(k-1) + \Delta\boldsymbol{u}(k|k)$. $\boldsymbol{u}(k)$ together with $\hat{\boldsymbol{x}}(k)$ serve as the input of (5.20) to actuate the system.

In the unconstrained case, the optimal solution is directly: $\Delta\boldsymbol{u}(\cdot|k)^* = -\boldsymbol{H}^{-1}\boldsymbol{c}$. But the developed QP formalism can also solve the case with constraints. Bounds of $\Delta\boldsymbol{u}(\cdot|k)$, $\boldsymbol{u}(\cdot|k)$ and $\hat{\boldsymbol{x}}(\cdot|k)$ are three typical inequality constraints. The following is the *condensing formalism* to represent the constraints in MPC regulator.

The bound of control increment

$$\Delta\boldsymbol{u}_{\min} \le \Delta\boldsymbol{u}(\cdot|k) \le \Delta\boldsymbol{u}_{\max} \tag{5.37a}$$

can be converted into

$$\underbrace{\begin{bmatrix} -\boldsymbol{I} & & \\ & \ddots & \\ & & -\boldsymbol{I} \\ \boldsymbol{I} & & \\ & \ddots & \\ & & \boldsymbol{I} \end{bmatrix}}_{\in \mathbb{R}^{2n_u N \times n_u N}} \Delta\boldsymbol{u}(\cdot|k) + \underbrace{\begin{bmatrix} \Delta\boldsymbol{u}_{\min} \\ \vdots \\ \Delta\boldsymbol{u}_{\min} \\ -\Delta\boldsymbol{u}_{\max} \\ \vdots \\ -\Delta\boldsymbol{u}_{\max} \end{bmatrix}}_{\in \mathbb{R}^{2n_u N}} \le \mathbf{0} \tag{5.37b}$$

The bound of control

$$\boldsymbol{u}_{\min} \le \boldsymbol{u}(\cdot|k) \le \boldsymbol{u}_{\max} \tag{5.38a}$$

with $\boldsymbol{u}(\cdot|k) = \boldsymbol{u}(k-1) + \Delta\boldsymbol{u}(\cdot|k)$, can be converted into

$$\underbrace{\begin{bmatrix} -\boldsymbol{I} & & \\ & \ddots & \\ & & -\boldsymbol{I} \\ \boldsymbol{I} & & \\ & \ddots & \\ & & \boldsymbol{I} \end{bmatrix}}_{\in\mathbb{R}^{2n_u N \times n_u N}} \Delta\boldsymbol{u}(\cdot|k) + \underbrace{\begin{bmatrix} \boldsymbol{u}_{\min} \\ \vdots \\ \boldsymbol{u}_{\min} \\ -\boldsymbol{u}_{\max} \\ \vdots \\ -\boldsymbol{u}_{\max} \end{bmatrix}}_{\in\mathbb{R}^{2n_u N}} + \begin{bmatrix} -\boldsymbol{u}(k-1) \\ \vdots \\ -\boldsymbol{u}(k-1) \\ \boldsymbol{u}(k-1) \\ \vdots \\ \boldsymbol{u}(k-1) \end{bmatrix} \leq \boldsymbol{0} \tag{5.38b}$$

The bound of state

$$\hat{\boldsymbol{x}}_{\min} \leq \hat{\boldsymbol{x}}(\cdot|k) \leq \hat{\boldsymbol{x}}_{\max} \tag{5.39a}$$

with $\hat{\boldsymbol{x}}(\cdot|k)$ defined in (5.33), can be converted into

$$\underbrace{\begin{bmatrix} -\Theta \\ \Theta \end{bmatrix}}_{\in\mathbb{R}^{2n_x(N+1)\times n_u N}} \Delta\hat{\boldsymbol{u}}(\cdot|k) + \underbrace{\begin{bmatrix} \hat{\boldsymbol{x}}_{\min} \\ \vdots \\ \hat{\boldsymbol{x}}_{\min} \\ -\hat{\boldsymbol{x}}_{\max} \\ \vdots \\ -\hat{\boldsymbol{x}}_{\max} \end{bmatrix}}_{\in\mathbb{R}^{2n_x(N+1)}} + \underbrace{\begin{bmatrix} -\left(\mathbf{A}\hat{\boldsymbol{x}}(k) + \mathbf{B}\hat{\boldsymbol{u}}(k-1)\right) \\ \left(\mathbf{A}\hat{\boldsymbol{x}}(k) + \mathbf{B}\hat{\boldsymbol{u}}(k-1)\right) \end{bmatrix}}_{\in\mathbb{R}^{2n_x(N+1)}} \leq \boldsymbol{0} \tag{5.39b}$$

We use the following symbol to denote the MPC regulation procedure:

$$\begin{aligned} \boldsymbol{u}(k) = \mathsf{regulate}(&N, n_x, n_u, n_y, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \hat{\boldsymbol{x}}^+(k), \boldsymbol{u}(k-1), \boldsymbol{x}_r(k), \boldsymbol{u}_r(k), \\ &\Delta\boldsymbol{u}_{\min}, \Delta\boldsymbol{u}_{\max}, \boldsymbol{u}_{\min}, \boldsymbol{u}_{\max}, \hat{\boldsymbol{x}}_{\min}, \hat{\boldsymbol{x}}_{\max}) \end{aligned} \tag{5.40}$$

5.7 Encrypted MPC: CSTR Model as a Case Study

In the CSTR model settings of [RMD17], There is no state and control constraints. With the results in Section 5.6, the increment of the control is (we drop argument "k" for simplicity):

$$\begin{aligned} \Delta\boldsymbol{u}(\cdot|k)^* &= -\boldsymbol{H}^{-1}\boldsymbol{c} = -\left(\Theta^\top\mathbf{C}^\top\mathbf{Q}\mathbf{C}\Theta + \mathbf{R}\right)^{-1}\Theta^\top\mathbf{C}^\top\mathbf{Q}e(\cdot|k) \\ &= -\underbrace{\left(\Theta^\top\mathbf{C}^\top\mathbf{Q}\mathbf{C}\Theta + \mathbf{R}\right)^{-1}\Theta^\top\mathbf{C}^\top\mathbf{Q}\mathbf{C}}_{\mathbf{M}\in\mathbb{R}^{n_u N\times n_x(N+1)}}\left(\mathbf{A}(\hat{\boldsymbol{x}} - \boldsymbol{x}_r) + \mathbf{B}(\boldsymbol{u} - \boldsymbol{u}_r)\right) \\ &= -\left(\underbrace{\mathbf{MA}}_{\mathbb{R}^{n_u N\times n_x}}(\hat{\boldsymbol{x}} - \boldsymbol{x}_r) + \underbrace{\mathbf{MB}}_{\mathbb{R}^{n_u N\times n_u}}(\boldsymbol{u} - \boldsymbol{u}_r)\right) \end{aligned}$$

The predictive control is $\boldsymbol{u}(\cdot|k) = \boldsymbol{u}(k-1) + \Delta\boldsymbol{u}(\cdot|k)$. In the encrypted setting, it becomes

$$\begin{aligned} \mathsf{ct}_{\Delta\boldsymbol{u}(\cdot|k)} = \mathsf{neg}(&\mathsf{add}(\mathsf{gemv}(\mathbf{MA}, \mathsf{sub}(\mathsf{ct}_{\hat{\boldsymbol{x}}}, \mathsf{ct}_{\boldsymbol{x}_r}), \mathsf{rk}), \\ &\mathsf{gemv}(\mathbf{MB}, \mathsf{sub}(\mathsf{ct}_{\boldsymbol{u}}, \mathsf{ct}_{\boldsymbol{u}_r}), \mathsf{rk}))) \end{aligned} \tag{5.41}$$

with the predictive control being

$$\mathsf{ct}_{\boldsymbol{u}(\cdot|k)} = \mathsf{add}\left(\mathsf{moddown}_{\ell\to\ell'}(\mathsf{ct}_{\boldsymbol{u}}), \mathsf{ct}_{\Delta\boldsymbol{u}(\cdot|k)}\right) \tag{5.42}$$

Although gemv's matrix input should be in $\mathbb{C}^{n_{\text{slots}}\times n_{\text{slots}}}$. Indeed, usually the inputted message is of $\mathbb{R}$ type (e.g. **MA** and **MB**), and has dimension less than $n_{\text{slots}} \times n_{\text{slots}}$. To tackle with this, we naturally embedded the $\mathbb{R}$ messages to $\mathbb{C}$ types and patch zeros to it to have the length equal to $n_{\text{slots}} \times n_{\text{slots}}$.

5.8 Summary

We summarize the full procedure of the model setup together with a close-loop simulation for both MPC and encrypted MPC.

The following quantities should be specified depending on the investigated model: the ODE $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p})$ of the plant (5.17), the stationary states $(\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s)$, the Jacobian $\partial_{\boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s)$, and specify the dimension of the state n_x and control n_u. With these, we linearize the ODE to obtain the state transition matrix $\boldsymbol{A}$ and control matrix $\boldsymbol{B}$. The output matrix $\boldsymbol{C} = \boldsymbol{I}_{n_y}$ where $n_y = n_x$ as most cases do. Also, design $\boldsymbol{H}_r$ (5.29) according to which quantities are observable.

Beside the ODE model itself, there is a disturbance model for the estimator. The corresponding matrices are $\boldsymbol{B}_d$ and $\boldsymbol{C}_d$. With them, we use (5.25) to calculate the Kalman filter matrices $\boldsymbol{L}_x$ and $\boldsymbol{L}_d$.

The "state $\boldsymbol{x}$" and "control $\boldsymbol{u}$" for the whole controller is understood as the deviation from the stationary state $(\boldsymbol{x}^s, \boldsymbol{u}^s)$. So we define the initial state $\boldsymbol{x}(0) = 0$ and $\boldsymbol{u}(-1) = 0$. The disturbance $\hat{\boldsymbol{d}}(0)$ is chosen as white noise, or zero for simplicity.

If we investigate the encrypted MPC, we should also initialize the ctx (4.1) for the HE settings, in which the n_{slots} should guaranteed larger than $n_u N$, where N is the predictive horizon. Also generate the keys pk, sk, and n_{slots} pieces of rotation keys rk.

Then we enter the close-loop simulation $k = 0, 1, \cdots, N_{\text{simulation}}$. Firstly, perform the estimation (5.27)

$$\begin{bmatrix}\hat{\boldsymbol{x}}^+(k)\\ \hat{\boldsymbol{d}}^+(k)\end{bmatrix} = \mathsf{estimate}\left(n_x, n_u, n_d, n_y, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{B}_d, \boldsymbol{C}_d, \begin{bmatrix}\boldsymbol{L}_x\\ \boldsymbol{L}_d\end{bmatrix}, \boldsymbol{y}(k) = \boldsymbol{C}\boldsymbol{x}(k), \begin{bmatrix}\hat{\boldsymbol{x}}(k)\\ \hat{\boldsymbol{d}}(k)\end{bmatrix}\right)$$

where $\hat{\boldsymbol{x}}(0) = \boldsymbol{x}(0)$, as the initial settings. Secondly, perform the target selection (5.31)

$$\begin{bmatrix}\boldsymbol{x}_r(k)\\ \boldsymbol{u}_r(k)\end{bmatrix} = \mathsf{select}\left(n_x, n_u, n_d, n_y, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{B}_d, \boldsymbol{C}_d, \boldsymbol{H}_r, \boldsymbol{r}_{\text{sp}}, \hat{\boldsymbol{d}}^+(k)\right)$$

Then, perform MPC regulation (5.40)

$$\begin{aligned}\boldsymbol{u}(k) = \mathsf{regulate}(&N, n_x, n_u, n_y, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \hat{\boldsymbol{x}}^+(k), \boldsymbol{u}(k-1), \boldsymbol{x}_r(k), \boldsymbol{u}_r(k),\\ &\Delta\boldsymbol{u}_{\min}, \Delta\boldsymbol{u}_{\max}, \boldsymbol{u}_{\min}, \boldsymbol{u}_{\max}, \hat{\boldsymbol{x}}_{\min}, \hat{\boldsymbol{x}}_{\max})\end{aligned}$$

Finally, actuate the plant (5.20) (we plus stationary state $(\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s)$ to convert to the "position unit" instead of the deviation)

$$\boldsymbol{x}(k+1) = \mathsf{actuate}\left(n_x, n_u, n_p, \boldsymbol{x}(k) + \boldsymbol{x}^s, \boldsymbol{u}(k) + \boldsymbol{u}^s, \boldsymbol{p}(k) + \boldsymbol{p}^s, \boldsymbol{f}, \partial_{\boldsymbol{x}} \boldsymbol{f}\right) - \boldsymbol{x}^s$$

Note that the actuate should no longer implemented when $k = N_{\text{simulation}}$. The above is the complete procedure of a close-loop simulation for a typical MPC.

In the close-loop simulation for the encrypted CSTR model, before performing the MPC regulation (5.40), we should do encryption

$$\begin{aligned} \hat{\boldsymbol{x}}^+(k) &\to \mathsf{enc}_{\mathsf{pk}}(\hat{\boldsymbol{x}}^+(k), \mathsf{pk}) = \mathsf{ct}_{\hat{\boldsymbol{x}}^+(k)} \\ \boldsymbol{u}(k-1) &\to \mathsf{enc}_{\mathsf{pk}}(\boldsymbol{u}(k-1), \mathsf{pk}) = \mathsf{ct}_{\boldsymbol{u}(k-1)} \\ \boldsymbol{x}_r(k) &\to \mathsf{enc}_{\mathsf{pk}}(\boldsymbol{x}_r(k), \mathsf{pk}) = \mathsf{ct}_{\boldsymbol{x}_r(k)} \\ \boldsymbol{u}_r(k) &\to \mathsf{enc}_{\mathsf{pk}}(\boldsymbol{u}_r(k), \mathsf{pk}) = \mathsf{ct}_{\boldsymbol{u}_r(k)} \end{aligned} \tag{5.43}$$

(also patch zeros to these quantities before encryption so that the "message length" is n_{slots}) then calculate $\mathsf{ct}_{\boldsymbol{u}(\cdot|k)}$ via (5.41) and (5.42). This is analogue to do the following

$$\begin{aligned} \mathsf{ct}_{\boldsymbol{u}(\cdot|k)} = \mathsf{regulate}(N, n_x, n_u, n_y, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \mathsf{ct}_{\hat{\boldsymbol{x}}^+(k)}, \mathsf{ct}_{\boldsymbol{u}(k-1)}, \mathsf{ct}_{\boldsymbol{x}_r(k)}, \mathsf{ct}_{\boldsymbol{u}_r(k)}, \{\mathsf{rk}_r\}_{r=0}^{n_{\text{slots}}-1} \\ \mathsf{NULL}, \mathsf{NULL}, \mathsf{NULL}, \mathsf{NULL}, \mathsf{NULL}, \mathsf{NULL}) \end{aligned} \tag{5.44}$$

then decrypt:

$$\boldsymbol{u}(k) = \mathsf{dec}(\mathsf{ct}_{\boldsymbol{u}(\cdot|k)}, \mathsf{sk})[0]$$

and use this $\boldsymbol{u}(k)$ to actuate the plant. The complete procedure is shown in Algorithm 24.

Algorithm 24 Closed-loop simulation of encrypted controller (Figure 5.2)

Input: $\boldsymbol{x}^s, \boldsymbol{u}^s, \boldsymbol{p}^s, n_x, n_u, n_p, n_y = n_x, n_d, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} = \boldsymbol{I}_{n_y}, \boldsymbol{B}_d, \boldsymbol{C}_d, \boldsymbol{H}_r, \boldsymbol{f}, \partial_{\boldsymbol{x}} \boldsymbol{f}, N_{\text{simulation}}, N$. The encryption parameters are $\lambda, n, q, n_{\text{slots}}, \Delta, h, \sigma$.

Output: $\{\boldsymbol{x}(k)\}_{k=0}^{N_{\text{simulation}}}, \{\boldsymbol{u}(k)\}_{k=0}^{N_{\text{simulation}}-1}$

1: Calculate $\boldsymbol{L}_x, \boldsymbol{L}_d$ by (5.25), $\mathbf{Q}, \mathbf{R}$ by (5.35) and $\begin{bmatrix} \boldsymbol{I}-\boldsymbol{A} & -\boldsymbol{B} \\ \boldsymbol{H}_r\boldsymbol{C} & \mathbf{0} \end{bmatrix}^{-1}$.
2: Let $\boldsymbol{x}(0) = \mathbf{0}, \boldsymbol{u}(0) = \mathbf{0}, \boldsymbol{x}_r(0) = \mathbf{0}, \boldsymbol{u}_r(0) = \mathbf{0}, \boldsymbol{r}_{\text{sp}} = \mathbf{0}, \hat{\boldsymbol{x}}(0) = \boldsymbol{x}(0), \hat{\boldsymbol{d}}(0) = \mathbf{0}$.
3: Initialize $\mathsf{ctx} = \mathsf{init}(\lambda, n, q, n_{\text{slots}}, \Delta, h, \sigma)$, $\mathsf{sk}, \mathsf{pk} = \mathsf{keypair}(q)$
4: Initialize rotation keys $\mathsf{rk}_r = \mathsf{genswk}(\tau_{5^r \bmod 2n}(\mathsf{sk}), \mathsf{sk})$ for $0 \le r < \cdots n_{\text{slots}}$.
5: **for** $k = 0$ to $N_{\text{simulation}} - 1$ **do**
6: Perform the estimation (5.27) to get $\hat{\boldsymbol{x}}^+(k), \hat{\boldsymbol{d}}^+(k)$.
7: Perform the target selection (5.31) to get $\boldsymbol{x}_r(k), \boldsymbol{u}_r(k)$.
8: Encrypt $\hat{\boldsymbol{x}}^+(k), \boldsymbol{u}(k-1), \boldsymbol{x}_r(k), \boldsymbol{u}_r(k)$ to $\mathsf{ct}_{\hat{\boldsymbol{x}}^+(k)}, \mathsf{ct}_{\boldsymbol{u}(k-1)}, \mathsf{ct}_{\boldsymbol{x}_r(k)}, \mathsf{ct}_{\boldsymbol{u}_r(k)}$ via (5.43).
9: Perform regulation (5.44) to get $\mathsf{ct}_{\boldsymbol{u}(\cdot|k)}$.
10: Decrypt $\boldsymbol{u}(k) = \mathsf{dec}(\mathsf{ct}_{\boldsymbol{u}(\cdot|k)}, \mathsf{sk})[0]$
11: Perform actuation (5.20) to get $\boldsymbol{x}(k+1)$.
12: **end for**

6 Implementations and Results Analysis

6.1 Introduction

We contribute three software in this project: GPQHE, HECTR and Libpmu. GPQHE is the software that implements CKKS homomorphic encryption scheme, which has been discussed in depth in Chapter 4. Its external library dependence is Libgcrypt 1.10, used for multi-precision integer management. Libgcrypt is chosen because its core components constitute the cryptographic module in the Linux kernel. HECTR is the software that implements encrypted model predictive control, it uses the function interfaces from GPQHE and LAPACK. LAPACK is the natural choice for low-level numerical computing, which is also used in NumPy. Section 6.2 explains necessary low-leveled implementation details of both software. Libpmu is used for performance test.

The correctness/precision, time usage, RAM usage are three main indicators to illustrate how well homomorphic encryption works. Their analysis to GPQHE is shown in Section 6.3. Section 6.4 is the closed-loop simulation of the unencrypted MPC and encrypted MPC using the CSTR model.

6.2 More Details on Low-Leveled Implementations

6.2.1 Obtaining Primes for RNS and its Linked-list Management

We here discuss the bookkeeping questions in Section 3.4.4. GPQHE follows the scheme of HEAAN, which requires that $\min \log \mathfrak{p}_k = 59$. By starting from $\mathfrak{p}_1 = (1 << 59) + 1$ and check whether $\mathfrak{p}_1$ is prime and satisfies NTT existence condition $\mathfrak{p}_k \equiv 1 \bmod m$ we get a series of $\mathfrak{p}_k$. Concretely shown in Algorithm 25, where both q_ℓ $(0 < \ell \leq L)$ and P are generally stored as `MPI` type.

As an important remark, the RNS P in NewHope and Kyber is the ring modulus q itself, i.e. $P = \mathfrak{p}_1 = q$, so that there is no explicit MPI-RNS conversion as that in Section 3.4.4 does. As a reference implementation submitted to NIST, they use small q that can be stored as a builtin type in C.

Due to the fact that the modulus level decreases during evaluations, so for each $q_\ell, 0 < \ell \leq L$, there is a corresponding minimum dimension to accommodate the current q_ℓ, schematically shown in Figure 6.1. Corresponding to different dimension, there are a series of corresponding parameters, e.g. $\{\hat{\mathfrak{p}}_k\}_{k=1}^{\mathsf{dim}}$, $\{\mathfrak{b}_k\}_{k=1}^{\mathsf{dim}}$, $\mathsf{Table}_{\omega_{m,k}}$, etc, so using arrays to store is not a clever idea

Algorithm 25 Obtain dim, $\{\mathfrak{p}_k\}_{k=1}^{\mathsf{dim}}$ and $P = \prod_{k=1}^{\mathsf{dim}} \mathfrak{p}_k$

Output: dim, $\{\mathfrak{p}_k\}_{k=1}^{\mathsf{dim}}$, $\prod_{k=1}^{\mathsf{dim}} \mathfrak{p}_k$

Initialize $k = 1$
Initialize $\mathfrak{p}_1 = (1 << 59) + 1$ ▷ This is obviously not a prime
Initialize $P = 1$ ▷ P is the product of all primes
while $P < q_\ell$ **do**
    while $\mathfrak{p}_k$ is not prime **do**
        $\mathfrak{p}_k += 2n$
    end while
    $P = P \cdot \mathfrak{p}_k$, $k++$
end while
return $\mathsf{dim} = k$, $\{\mathfrak{p}_k\}_{k=1}^{\mathsf{dim}}$, $P(= \prod_{k=1}^{\mathsf{dim}} \mathfrak{p}_k)$

q_0	$\mathfrak{p}_1$						$\prod_{k=1}^{1} \mathfrak{p}_k = P_1$
$q_1 = q_0\mathfrak{p}$	$\mathfrak{p}_1$	$\mathfrak{p}_2$					$\prod_{k=1}^{2} \mathfrak{p}_k = P_2$
$q_2 = q_0\mathfrak{p}^2$	$\mathfrak{p}_1$	$\mathfrak{p}_2$	$\mathfrak{p}_3$				$\prod_{k=1}^{3} \mathfrak{p}_k = P_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$			$\vdots$
$q_{L-1} = q_0\mathfrak{p}^{L-1}$	$\mathfrak{p}_1$	$\mathfrak{p}_2$	$\mathfrak{p}_3$	$\cdots$	$\mathfrak{p}_{\mathsf{dim}-1}$		$\prod_{k=1}^{\mathsf{dim}-1} \mathfrak{p}_k = P_{\mathsf{dim}-1}$
$q_L = q_0\mathfrak{p}^{L}$	$\mathfrak{p}_1$	$\mathfrak{p}_2$	$\mathfrak{p}_3$	$\cdots$	$\mathfrak{p}_{\mathsf{dim}-1}$	$\mathfrak{p}_{\mathsf{dim}}$	$\prod_{k=1}^{\mathsf{dim}} \mathfrak{p}_k = P_{\mathsf{dim}}$

Figure 6.1: Linked-list management of q_ℓ and P_{dim}

because arrays occupy a piece of continuous memory. GPQHE uses linked-list to optimize the storage of RNS related parameters, as SEAL does. In this scheme, each dimension serves as a node of linked-list, and each linked-list stores following quantities

$$\begin{aligned}\mathsf{rns\_ctx}_k = \{ & k, \mathfrak{p}_k, \mathfrak{p}_{\mathsf{mont},k}^{-1}, \mathfrak{p}_{\mathsf{barr},k}^{-1}, n^{-1} \bmod \mathfrak{p}_k, \mathsf{Table}_{\omega_{m,k}}, \mathsf{Table}_{\omega_{m,k}^{-1}} \\ & P_k, \{\hat{\mathfrak{p}}_l\}_{l=1}^{k}, \{\mathfrak{b}_l\}_{l=1}^{k} \}\end{aligned}$$

where k is the current dim. The quantities in the first line are relevant to NTT while that in the second line are relevant to MPI-RNS conversion. By using linked-list, $\mathsf{Table}_{\omega_{m,k}}$, $\mathsf{Table}_{\omega_{m,k}^{-1}}$, $\{\hat{\mathfrak{p}}_l\}_{l=1}^{k}$, $\{\mathfrak{b}_l\}_{l=1}^{k}$ are stored as $\mathtt{u64}[n], \mathtt{u64}[n], \mathtt{MPI}[k], \mathtt{MPI}[k]$ arrays, while others are numbers of corresponding types.

We need to point out that there is no strict relation between L and dim since we impose no additional theoretical conditions on the choice of each $\mathfrak{p}_k$ except for $\mathfrak{p}_k \equiv 1 \bmod 2n$. For example, in the settings (4.1) $\mathsf{ctx} = \mathsf{init}(128, 2^{15}, 2^{881}, *, \mathfrak{p} = \Delta = 2^{30}, h = 64, \sigma = 3.2)$ (where "$*$" stands for placeholder, here is n_{slots}), we have $\mathsf{dim} = 16$ while $L = 29$. So the indices in Figure 6.1 is only a special case for illustration.

We finally remark that the possibility of using multi-thread to accelerate the MPI-RNS conversion. In Algorithm 6,7,8, the blocks within the for-loop "$k = 1$ to dim" can be combined into one for-loop and parallelize by thread; similarly, the for-loop "$i = 0$ to $n-1$" in Algorithm 8 can also perform thread acceleration in this way. However, GPQHE has not implemented thread acceleration.

6.2.2 eps Function: Floating-point relative accuracy

The floating-point relative accuracy function eps in Matlab and Octave is implemented via memcpy (memory copy) in C, shown in Algorithm 26. Figure 6.2 is an illustration of $x = 1$ as input. Its memory copy to an `u64` integer is 4607182418800017408. After $a+ = 1$ and copying a to x', x' would "seems like 1", however, x' is not 1: $x' - x = 2.220446 \times 10^{-16}$.

Algorithm 26 Floating-point relative accuracy function

Input: `double` x
Output: ϵ, the relative accuracy of x in machine's floating-point representation.

1: $x = |x|$ ▷ $\mathsf{sizeof}(x) = 8$
2: Declare `u64` a, `double` x' ▷ both are of size 8
3: $\mathsf{memcpy}(\&a,\ \&x,\ 8)$
4: $a+ = 1$
5: $\mathsf{memcpy}(\&x',\ \&a,\ 8)$
6: **return** $x' - x$

address	a	a+1
0xb0c028	0x00	0x01
0xb0c029	0x00	0x00
0xb0c02a	0x00	0x00
0xb0c02b	0x00	0x00
0xb0c02c	0x00	0x00
0xb0c02d	0x00	0x00
0xb0c02e	0xf0	0xf0
0xb0c02f	0x3f	0x3f
0xb0c030	⋮	⋮

Figure 6.2: Illustration of $\mathsf{eps}(1)$

6.2.3 Matrix Calculus via LAPACK

We talk about three frequently used matrix calculus in HECTR:

Matrix Inverse It is implemented by two steps: first do getrf, the LU factorization, then getri, the inverse of an LU-factored matrix.

Pseudo-Inverse It is used in calculating the initial $\boldsymbol{w}^{(0)} = -\boldsymbol{A}_{\text{eq}}^{+}\boldsymbol{b}_{\text{eq}}$ in quadprog (Algorithm 23) if equality constraints are provided. Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. $\boldsymbol{A}^{+} = \mathsf{pinv}(\boldsymbol{A}) \in \mathbb{R}^{n\times m} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^{\dagger}$, where $\boldsymbol{U} \in \mathbb{R}^{m\times l}, \boldsymbol{\Sigma} \in \mathbb{R}^{l\times l}, \boldsymbol{V}^{\dagger} \in \mathbb{R}^{l\times n}$ are returns of $\boldsymbol{A}$'s singular value decomposition. Due to the fact that small singular value $\boldsymbol{\Sigma}$ causes a huge inverse $\boldsymbol{\Sigma}^{-1}$, Matlab and Octave use a cutoff scheme: the singular values that smaller than $\max(m,n)\cdot\mathsf{eps}(\boldsymbol{\Sigma}_0)$ are treated as zeros, then trim the dimension l to l' of three matrices accordingly.

Matrix Exponential It is used in linearization of a model ODE, shown in Section 5.3. Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. $\mathsf{expm}(\boldsymbol{A}) = \boldsymbol{V}\exp(\boldsymbol{D})\boldsymbol{V}^{-1}$, where $\boldsymbol{V}, \boldsymbol{D}$ are returns of $\boldsymbol{A}$'s eigenvalue decomposition.

6.3 Performance and Precision Analysis

We provide three functions to monitor the performance of GPQHE: rdtsc to read (CPU) time-stamp counter in the unit CPU cycles per tick, which is a de facto measurement of time usage; rdclk, to read the real time (user time + system time) in seconds; rdmaxrss, to read the maximum resident set size (RSS), a measurement of RAM usage. Experiments are performed on an Intel i5-6300U CPU with 3GHz being the maximum frequency that can achieve. Though by dividing the CPU frequency one yields the real time in seconds, however, the CPU frequency varies with the presenting operation system states, so we provide rdclk which invokes a system call to clock to read the time counts in seconds.

We use classical security with $\lambda = 128$, $h = 64$, $\sigma = 3.2$ and pack ctx = $(\log n, \log q, n_{\text{slots}}, \log \Delta)$ as the interested parameters. We use pk for experiment since both pk- and sk- encrypted schemes give the same results based on the previous experiments.

HE KEM The result is shown in Table.6.1. Reducing n and q can reduce the time and memory usage in KEM. n_{slots} only shows its impact when generating rk since there are a bunch (n_{slots}) of keys: the time and RAM consumption of $(14, 438, 16, 50)$'s rk generation is indeed approximately 4 times of that of $(14, 438, 4, 50)$.

Table 6.1: Performance of HE KEM.

Key	ctx	rdclk (ms)	rdmaxrss
generate pk & sk pair	$(14, 438, 16, 50)$	333.891100	6044 Kb
	$(14, 438, 4, 50)$	352.519741	6044 Kb
	$(14, 109, 16, 50)$	111.067702	4116 Kb
	$(12, 109, 16, 50)$	27.783834	1128 Kb
generate rlk	$(14, 438, 16, 50)$	1,528.751585	1.332 Mb
	$(14, 438, 4, 50)$	1,486.125385	1.337 Mb
	$(14, 109, 16, 50)$	377.952248	3580 Kb
	$(12, 109, 16, 50)$	96.634911	736 Kb
generate rk	$(14, 438, 16, 50)$	23,435.378576	12.561 Mb
	$(14, 438, 4, 50)$	5,577.383948	3.796 Mb
	$(14, 109, 16, 50)$	6,012.241115	3.821 Mb
	$(12, 109, 16, 50)$	1,366.171140	7916 Kb
generate ck	$(14, 438, 16, 50)$	1,356.186569	1.293 Mb
	$(14, 438, 4, 50)$	1,337.211423	1.306 Mb
	$(14, 109, 16, 50)$	334.038474	3608 Kb
	$(12, 109, 16, 50)$	82.268925	948 Kb

HE Primitives The result is shown in Table 6.2 (in the column rdmaxrss, some of them are 0's, perhaps because their memory cost is too tiny to be monitored). Reducing n can always significantly reduce the resource consumption, however, reducing q does not guarantee this goal, e.g. in additions. Though generating rk is a resource exhaustive procedure, performing rotation to an individual slot costs little. A noticeable observation is that both rotation and

conjugation cost almost the same time and RAM, this is because both of them leverage the automorphism property. The most expensive operation is the ciphertext multiplication.

Δ does not affect the resource consumption; instead, it affects the precision. When $\log \Delta = 50$, the precision is of order $\sim 10^{-12}$ if encrypted by pk, as that shows in Table 6.2; when $\log \Delta = 30$, the precision is of order $\sim 10^{-6}$, as that shown at the last two rows in Table 6.3. Though a higher precision is always desired, however, in nonlinear function evaluations, large Δ would deplete the level soon. So there is a trade-off between choosing Δ and q.

Table 6.2: Performance of HE primitives.

Operation	ctx	Precision	rdtsc	rdclk (ms)	rdmaxrss
enc	$(14, 438, 16, 50)$	2.96×10^{-14}	540004079	216.700719	6600 Kb
	$(14, 438, 4, 50)$	1.44×10^{-14}	535522603	219.740140	6600 Kb
	$(14, 109, 16, 50)$	2.93×10^{-14}	221505784	104.559621	5544 Kb
	$(12, 109, 16, 50)$	2.59×10^{-14}	51821219	25.092819	1584 Kb
dec	$(14, 438, 16, 50)$	2.96×10^{-14}	379762226	136.750465	0 Kb
	$(14, 438, 4, 50)$	1.44×10^{-14}	338258729	132.828728	0 Kb
	$(14, 109, 16, 50)$	2.93×10^{-14}	75636711	32.506267	0 Kb
	$(12, 109, 16, 50)$	2.59×10^{-14}	18625049	8.522551	0 Kb
$\mathsf{add}_{\mathsf{pt}}$	$(14, 438, 16, 50)$	2.92×10^{-12}	21216049	9.195485	0 Kb
	$(14, 438, 4, 50)$	8.57×10^{-13}	21332606	9.412839	0 Kb
	$(14, 109, 16, 50)$	2.94×10^{-12}	20593622	10.449749	0 Kb
	$(12, 109, 16, 50)$	1.90×10^{-12}	5716239	2.111140	0 Kb
add	$(14, 438, 16, 50)$	3.53×10^{-12}	24127340	10.626529	0 Kb
	$(14, 438, 4, 50)$	1.43×10^{-12}	22881326	9.833877	0 Kb
	$(14, 109, 16, 50)$	3.96×10^{-12}	22255153	9.793495	0 Kb
	$(12, 109, 16, 50)$	2.32×10^{-12}	5611365	2.328595	0 Kb
$\mathsf{mult}_{\mathsf{pt}}(*, *) + \mathsf{rs}$	$(14, 438, 16, 50)$	2.47×10^{-12}	1069233425	404.782807	0 Kb
	$(14, 438, 4, 50)$	1.11×10^{-12}	1057432069	531.732099	0 Kb
	$(14, 109, 16, 50)$	1.43×10^{-12}	330983931	141.510758	0 Kb
	$(12, 109, 16, 50)$	1.27×10^{-12}	80030113	32.206144	0 Kb
$\mathsf{mult}(*, *, \mathsf{rlk}) + \mathsf{rs}$	$(14, 438, 16, 50)$	3.15×10^{-12}	5612994869	2,303.951218	2.319 Mb
	$(14, 438, 4, 50)$	1.63×10^{-12}	5605811372	2,204.419684	2.314 Mb
	$(14, 109, 16, 50)$	3.81×10^{-12}	1294199457	557.964015	5016 Kb
	$(12, 109, 16, 50)$	1.80×10^{-12}	311214965	127.011819	1320 Kb
$\mathsf{rot}(*, n_{\text{slots}}/2, \mathsf{rk})$	$(14, 438, 16, 50)$	2.68×10^{-12}	2387535826	969.516122	5088 Kb
	$(14, 438, 4, 50)$	1.36×10^{-12}	2579428077	1,060.778946	5072 Kb
	$(14, 109, 16, 50)$	2.14×10^{-12}	606428212	235.696431	208 Kb
	$(12, 109, 16, 50)$	1.61×10^{-12}	144760627	63.111918	0 Kb
$\mathsf{conj}(*, \mathsf{ck})$	$(14, 438, 16, 50)$	2.71×10^{-12}	2517294202	972.978089	7008 Kb
	$(14, 438, 4, 50)$	1.33×10^{-12}	2347252608	903.261426	6968 Kb
	$(14, 109, 16, 50)$	2.15×10^{-12}	611319037	235.354460	1056 Kb
	$(12, 109, 16, 50)$	1.61×10^{-12}	139869806	58.931308	264 Kb

HE Linear Transformations The result is shown in Table 6.3, in which we remove the rdtsc column since it is only interested for primitive operations; the column "level" means level change. The resource consumption are almost the same among gemv, sum and idx in the corresponding parameter settings since the latter two are derived from a gemv algorithm according to Section 4.6.1. The time usage of $(14, 438, 16, 50)$'s gemv is indeed approximately 4 times of that of $(14, 438, 4, 50)$, however, their RAM usage are almost the same — this observation is different from that of Table 6.1.

Table 6.3: Performance of HE linear transformation.

Operation	ctx	Precision	Level	rdclk (ms)	rdmaxrss
gemv$(*, \mathsf{rk})$	$(14, 438, 16, 50)$	8.71×10^{-12}	$8 \to 7$	23,728.783353	2.357 Mb
	$(14, 438, 4, 50)$	4.35×10^{-13}	$8 \to 7$	6,964.191927	2.343 Mb
	$(14, 109, 16, 50)$	6.58×10^{-12}	$2 \to 1$	6,366.409786	1.098 Mb
	$(12, 109, 16, 50)$	3.40×10^{-12}	$2 \to 1$	1,529.178110	2232 Kb
sum$(*, \mathsf{rk})$	$(14, 438, 16, 50)$	8.06×10^{-12}	$8 \to 7$	23,324.799436	2.647 Mb
	$(14, 438, 4, 50)$	2.15×10^{-13}	$8 \to 7$	6,956.085209	2.665 Mb
	$(14, 109, 16, 50)$	4.09×10^{-12}	$2 \to 1$	7,003.026923	1.293 Mb
	$(12, 109, 16, 50)$	2.34×10^{-12}	$2 \to 1$	1,749.960873	2688 Kb
idx$(*, \mathsf{rk})$	$(14, 438, 16, 50)$	2.05×10^{-12}	$8 \to 7$	22,304.672866	2.685 Mb
	$(14, 438, 4, 50)$	5.34×10^{-13}	$8 \to 7$	6,047.578471	2.657 Mb
	$(14, 109, 16, 50)$	1.11×10^{-12}	$2 \to 1$	5,441.236965	1.293 Mb
	$(12, 109, 16, 50)$	5.58×10^{-13}	$2 \to 1$	1,320.167525	2688 Kb
nrm22$(*, \mathsf{rlk}, \mathsf{rk}, \mathsf{ck})$	$(14, 438, 16, 50)$	3.84×10^{-12}	$8 \to 6$	25,608.547131	4.281 Mb
	$(14, 438, 4, 50)$	1.03×10^{-12}	$8 \to 6$	9,451.586554	4.287 Mb
	$(14, 109, 16, 30)$	6.86×10^{-06}	$3 \to 1$	7,064.632972	2.077 Mb
	$(12, 109, 16, 30)$	9.87×10^{-07}	$3 \to 1$	1,684.678907	4556 Kb

HE Nonlinear Functions and Number Comparison The result is shown in Table 6.4, where we use $n_{\text{slots}} = 4, \log \Delta = 30$ throughout the test so that ctx $= (\log n, \log q)$. 5 and/or 6 after rlk is iteration, while the last "2" in cmp is the precision bit, c.f. Section 4.6.2 and 4.6.3. The nonlinear functions and comparison kill many levels, so we must set a large n (and thus permits larger q, c.f. the relation [ACC+18]) and small Δ to achieve the correct results: the former one increase the resource consumption, the latter one sacrifice the precision.

Table 6.4: Performance of HE nonlinear functions and comparison. $n_{\text{slots}} = 4, \log \Delta = 30$

Operation	ctx	Precision	Level	rdclk (ms)	rdmaxrss
$\mathsf{exp}(\frac{2\pi i}{\Delta}\mathsf{ct}, \mathsf{rlk}, 5)$	$(14, 438)$	7.89×10^{-9}	$14 \to 5$	9,504.579667	1.903 Mb
$\mathsf{ln}(1 + \frac{\mathsf{ct}}{100000}, \mathsf{rlk})$	$(14, 438)$	8.19×10^{-7}	$14 \to 10$	15,988.186611	7.014 Mb
$\mathsf{sigmoid}(\mathsf{ct}, \mathsf{rlk})$	$(14, 438)$	2.09×10^{-7}	$14 \to 10$	15,474.620148	8.092 Mb
$\mathsf{inv}(\mathsf{ct}, \mathsf{rlk}, 5)$	$(14, 438)$	1.94×10^{-6}	$14 \to 8$	21,571.355721	4.175 Mb
$\mathsf{sqrt}(\mathsf{ct}, \mathsf{rlk}, 6)$	$(14, 438)$	2.07×10^{-6}	$14 \to 2$	32,003.074994	4.552 Mb
$\mathsf{cmp}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rlk}, 5, 2)$	$(15, 881)$	-	$29 \to 5$	316,065.779455	20.711 Mb

The return of HE cmp is a vector with each element in $(0, 1)$. In our experiment,

$$\begin{aligned}
\mathsf{ct}_1 &\xleftarrow{\mathsf{enc}} (0.683594, 1.457031, 1.117188, 1.265625) \\
\mathsf{ct}_2 &\xleftarrow{\mathsf{enc}} (0.929688, 1.257812, 1.156250, 0.585938) \\
\mathsf{cmp}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rlk}, d = 5, \alpha = 2) &\xrightarrow{\mathsf{dec}} (0.226193, 0.642931, 0.465686, 0.956078) \xrightarrow{\lfloor\cdot\rceil} (0, 1, 0, 1)
\end{aligned}$$

However, the final $\lfloor\cdot\rceil$ is carried on the decrypted vector $(0.226193, 0.642931, 0.465686, 0.956078)$; in other words, given the compared ciphertext $\mathsf{cmp}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rlk}, 5, 2)$, we still not know which one is larger. There is, however, no such "homomorphic rounding" so far.

6.4 Closed-loop Simulation of (Encrypted) MPC

Besides the parameters settings in Section 5.3, we use the disturbance model in [RMD17] Example 1.11 (a): $n_d = 2$,

$$\boldsymbol{C}_d = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{n_y \times n_d}, \quad \boldsymbol{B}_d = \mathbf{0}_{n_x \times n_d}$$

In the sample code, the authors calculate $\boldsymbol{K}$ from dlqr solver in Octave. Our HECTR re-implements dlqr in C as well. We provide the results implemented via dlqr and mpc solvers (the simulation time $N_{\text{simulation}} = 40$ and predictive horizon $N = N_{\text{simulation}}/10 = 4$), shown in Figure 6.3 and 6.4.

Then we implement the comparison of the unencrypted and encrypted MPC in Section 5.8. We use $\mathsf{ctx} = (\log n = 12, \log q = 109, n_{\text{slots}} = 16, \log \Delta = 50)$ (since $n_{\text{slots}} \geq n_u N$). The results are shown in Figure 6.5 and 6.6 and their resource consumption is shown in Table 6.5. It's within our expectation that both lines are overlapped. The corresponding error comparison is shown in Figure 6.7 and 6.8, which is of order $\sim 10^{-12}$.

The parameter $\mathsf{ctx} = (12, 109, 16, 50)$ is chosen based on the previous study in the last section: it costs least resources. In Section 5.7 and 5.8 we have simplified the encrypted system settings to a purely linear case that can be accomplished by a HE linear transformation gemv which costs only one level to make the optimal control $\boldsymbol{u}(k)$. However, despite this simplification, the encrypted controller uses 2 minutes 16.5 seconds to run the simulation; in contrast, the time usage of unencrypted controller is negligible.

Table 6.5: Performance of unencrypted and encrypted model predictive controller.

	rdclk (ms)	rdmaxrss
Unencrypted MPC	4.120997	0 Kb
Encrypted MPC	136,558.752570	1.148 Mb

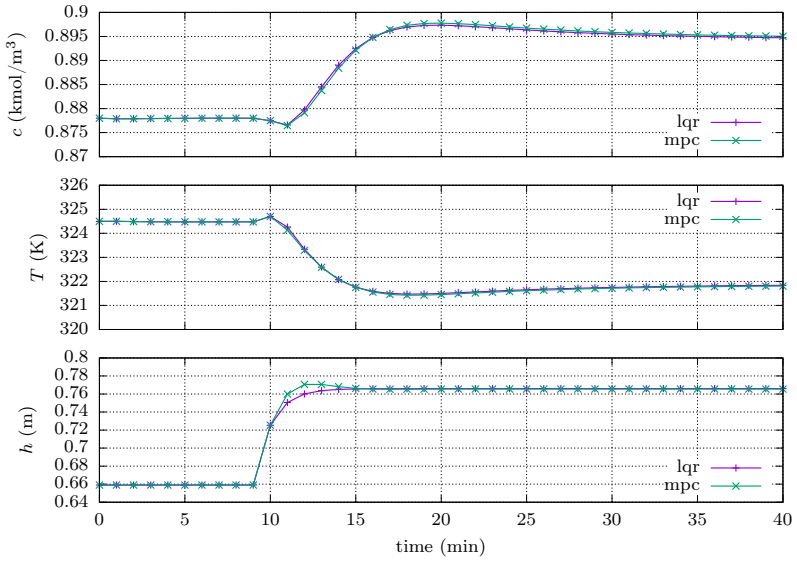

Figure 6.3: Three measured outputs versus time after a step change in inlet flow rate at 10 min. This is a comparative illustration between dlqr and mpc solvers.

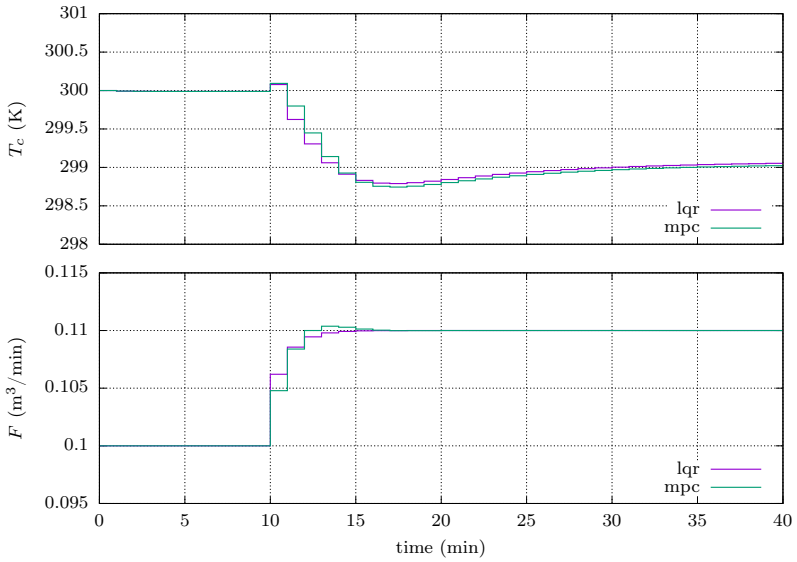

Figure 6.4: Two manipulated inputs versus time after a step change in inlet flow rate at 10 min. This is a comparative illustration between dlqr and mpc solvers.

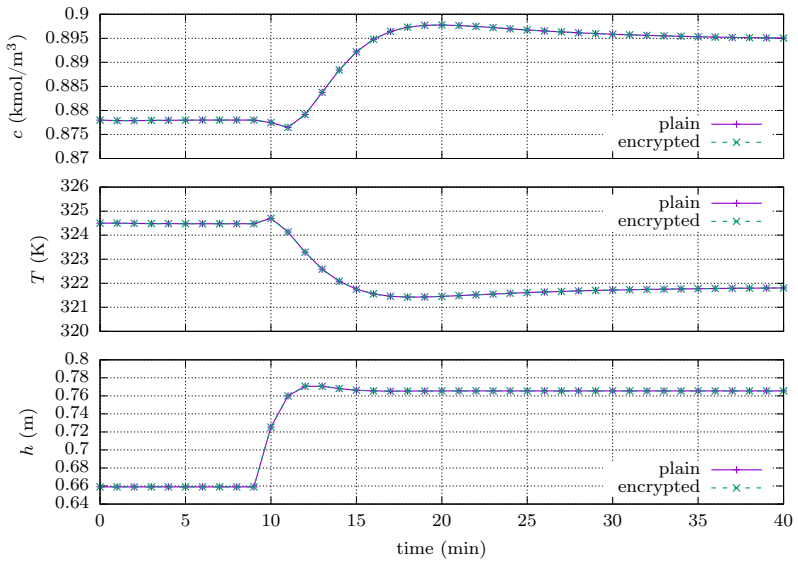

Figure 6.5: Three measured outputs versus time after a step change in inlet flow rate at 10 min. This is a comparative illustration between plain (unencrypted) and encrypted MPC.

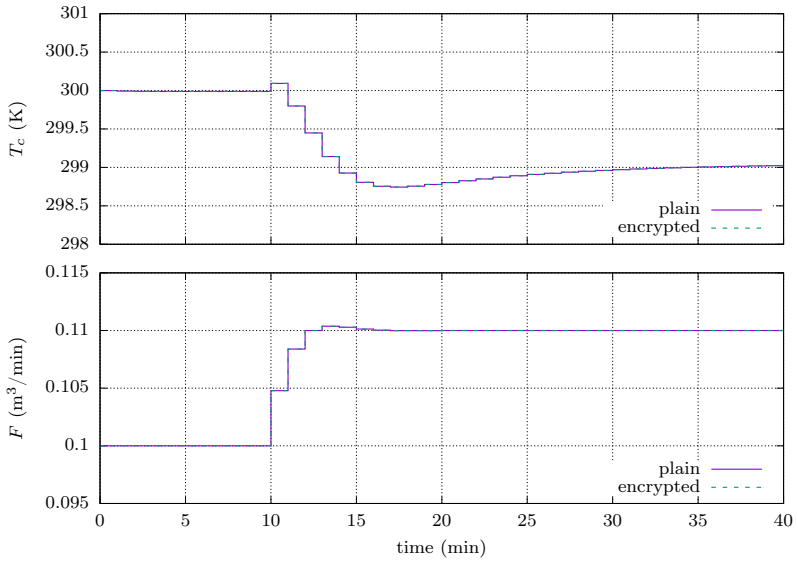

Figure 6.6: Two manipulated inputs versus time after a step change in inlet flow rate at 10 min. This is a comparative illustration between plain (unencrypted) and encrypted MPC.

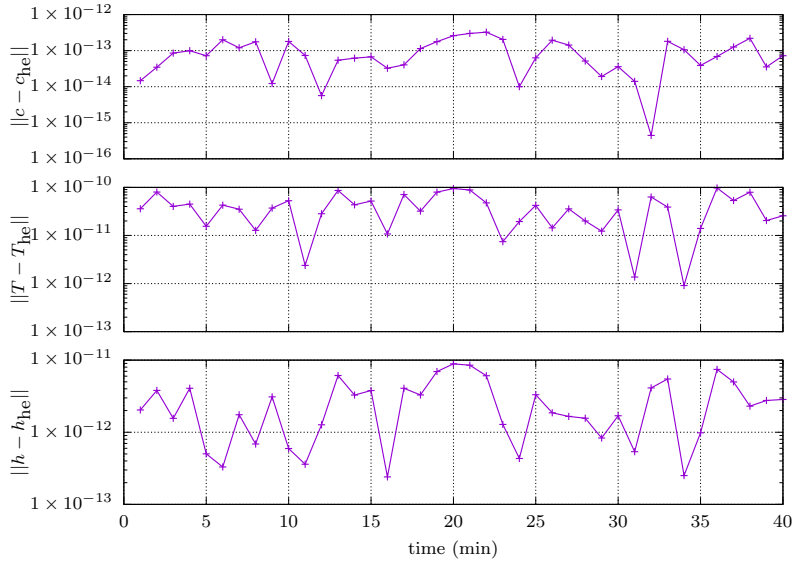

Figure 6.7: The relative error comparison between plain and encrypted MPC.

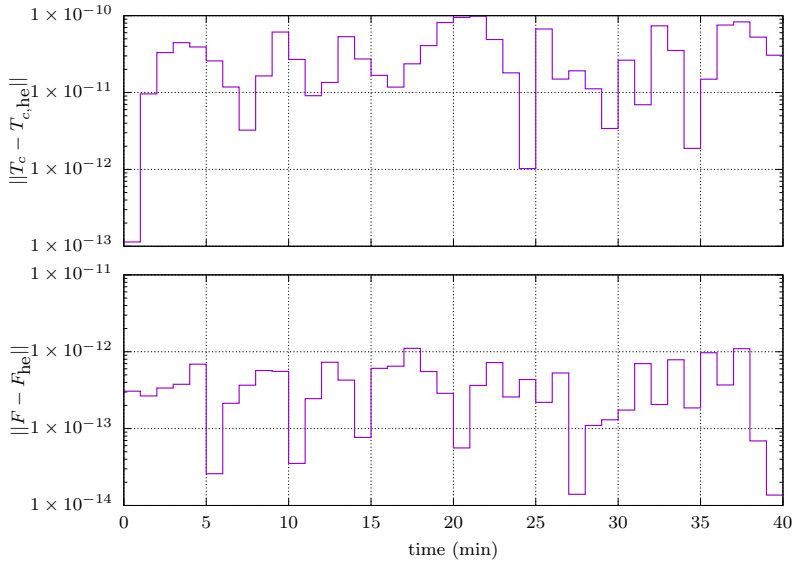

Figure 6.8: The relative error comparison between plain and encrypted MPC.

7 Conclusion and Future Works

Based on CKKS's homomorphic encryption scheme, we designed an encrypted MPC, which can perform homomorphic calculations to the encrypted signal. Since the CKKS scheme is RLWE-based, the encrypted MPC can avoid signal leakage risk as well as is post-quantum secure. We developed our HE software GPQHE to achieve this goal, GPQHE supports HE primitives, linear transformation, and some nonlinear functions.

With the CSTR model as a benchmark system, we verified the correctness of the encrypted MPC with the unencrypted counterpart. To make a comparison between the encrypted and unencrypted MPC, we tested many parameter combinations in GPQHE and select the one that occupies the least system resources as well as is capable to accomplish the control task as the parameter settings for the encrypted MPC. Both systems are compared under the fully simplified settings and the results show that an encrypted MPC costs much more time and RAM than the unencrypted one.

Although using RLWE-based HE to design a controller provides a seemly promising solution of avoiding signal leakage risk as well as is post-quantum secure, we still want to emphasize the drawback of the performance of encrypted MPC. In the fully simplified settings, in which the encrypted MPC actually runs a linear transformation task and only costs one level, it still costs 10K more time and 100M more RAM to accomplish the task. This raises a high risk of being side-channel attacked.

There are two possible directions for future works. The first one is focusing on hardware acceleration to HE. The second one is to use MLWE instead of RLWE. Both the polynomial degree n and the modulus q of MLWE are smaller than that of RLWE so it costs less time and RAM usage. It seemly a contradiction that we need a larger modulus to perform homomorphic operations but MLWE goes the opposite, however, if the computation/query/control goal can be achieved by linear operations, which only costs one or two levels — for example, in our fully simplified CSTR model, the control task is achieved by only one gemv operation — then the endeavor of exploring the solution of reducing n and q is meaningful.

Bibliography

[AAB+17] Erdem Alkim, Roberto Maria Avanzi, Joppe W. Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. Newhope algorithm specifications and supporting documentation. 2017.

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange: A new hope. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 327343, USA, 2016. USENIX Association.

[Ber09] Dennis S. Bernstein. *Basic Matrix Properties*, pages 85–178. Princeton University Press, 2009.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309325, New York, NY, USA, 2012. Association for Computing Machinery.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 360–384, Cham, 2018. Springer International Publishing.

[CKK+19] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 415–445, Cham, 2019. Springer International Publishing.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and

Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[Con22a] Keith Conrad. Expository papers, 2022.

[Con22b] Keith Conrad. Expository papers, 2022.

[D'A21] Jan-Pieter D'Anvers. Design and security analysis of lattice-based post-quantum encryption, 2021.

[Dar20] Moritz Schulze Darup. Encrypted mpc based on admm real-time iterationssupport by the german research foundation (dfg) under the grant schu 2940/4-1 is gratefully acknowledged. *IFAC-PapersOnLine*, 53(2):3508–3514, 2020. 21th IFAC World Congress.

[DM02] Shafi Goldwasser Daniele Micciancio. *Complexity of Lattice Problems*. The Springer International Series in Engineering and Computer Science. Springer, Boston, MA, 1 edition, 2002.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. *the 41st ACM Symposium on Theory of Computing (STOC)*, 2009.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[HS97] M.A. Henson and D.E. Seborg. *Nonlinear Process Control*. Prentice Hall PTR, 1997.

[HS21] Shai Halevi and Victor Shoup. Bootstrapping for helib. *Journal of Cryptology*, 34(1):7, Jan 2021.

[KF15] Kiminao Kogiso and Takahiro Fujita. Cyber-security enhancement of networked control systems using homomorphic encryption. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 6836–6843, 2015.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients. *MATH. ANN*, 261:515–534, 1982.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 35–54, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[MH78] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Comput. Complex.*, 16(4):365411, dec 2007.

[MKvOV96] A.J. Menezes, J. Katz, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press, 1996.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Pei16] Chris Peikert. A decade of lattice cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283424, mar 2016.

[PR03] Gabriele Pannocchia and James B. Rawlings. Disturbance models for offset-free model-predictive control. *AIChE Journal*, 49(2):426–437, 2003.

[Qiu03] Weisheng Qiu. *Abstract Algebra (in Chinese)*. Higher Education Press, 2003.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[Reg10] Oded Regev. The learning with errors problem (invited survey). In *2010 IEEE 25th Annual Conference on Computational Complexity*, pages 191–204, 2010.

[RMD17] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[Rot15] Joseph J. Rotman. *Advanced Modern Algebra: Third Edition, Part 1*. Graduate studies in mathematics 165 180. American Mathematical Society, 3ed. edition, 2015.

[SDAQP21] Moritz Schulze Darup, Andreea B. Alexandru, Daniel E. Quevedo, and George J. Pappas. Encrypted control for networked systems: An illustrative introduction and current challenges. *IEEE Control Systems Magazine*, 41(3):58–78, June 2021.

[SDRS+18] Moritz Schulze Darup, Adrian Redder, Iman Shames, Farhad Farokhi, and Daniel Quevedo. Towards encrypted mpc for linear constrained systems. *IEEE Control Systems Letters*, 2(2):195–200, April 2018.

[Ste18] Douglas Stebila. Introduction to post-quantum cryptography and learning with errors, 2018.

[THC11] Charles E Leiserson;Ronald L Rivest; Clifford Stein Thomas H Cormen. *Introduction to Algorithms*. Third edition edition, 2011.