Instruction Manual for **HECS**: Homomorphically Encrypted Controller System

Minryoung Kim

June 8, 2020

1 Overview

"**HECS**" library provides C++ codes to build an encrypted control system of a linear SISO controller.

The main intend of this library is to give designers an easy way to implement their control system into encrypted one and simulate its performance.

This library is composed of encrypted system builder, LWE-encrypter, encrypted controller, sensor, actuator, and system simulation codes.

2 Getting Started

2.1 How to build

Windows environment

- Open the project solution file '**EncryptedSystem.sln**' with Visual Studio.
- To build this project, choose **Build Solution**(F7) from the **Build** menu.
- To run the code, on the menu bar, choose **Debug** → **Start without debugging**(Ctrl+F5).

Also, you can run a simulation of a control system operation right away with the prepared example codes.

2.2 Making an input file

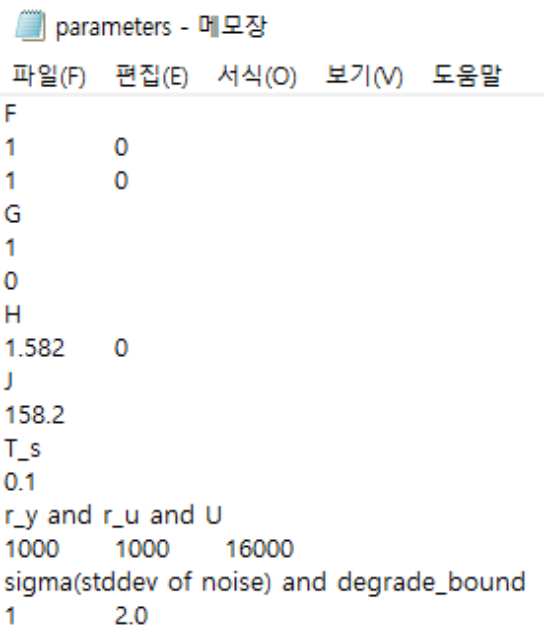
Input file '**parameters.txt**' specify the controller and the LWE-based cryptosystem. Each field in a line must be separated by a tab. Following parameters are entries of the input file 'parameters.txt'.

F, G, H, J: state space matrices in following form of controller

$$x^+ = Fx + Gy \tag{1}$$
$$u = Hx + Jy \tag{2}$$

T_s: sampling time(second)
r_y: plant output sensor resolution
r_u: actuator resolution
U: size of controller range
sigma(σ): standard deviation of Gaussian noise
degrade bound($degrade$): desired upper bound of performance degradation
(ex: if $degrade = 1.8$, then degradation ratio is under $10^{-1.8} = 0.0158$)

The input format of the parameters is shown in the following example.



(a) Input file "parameters.txt"

	A	B	C	D
1	F			
2	1	0		
3	1	0		
4	G			
5	1			
6	0			
7	H			
8	1.582	0		
9	J			
10	158.2			
11	T_s			
12	0.1			
13	r_y and r_u and U			
14	1000	1000	16000	
15	sigma(stddev of noise) and degrade_bound			
16	1	2		

(b) Input file opened with a spreadsheet software(ex: Excel)

Figure 1: Input file form

3 API: Modules

3.1 class Encrypter

Class **Encrypter** is a module for encryption, decryption, and signal scaling. You can create an **Encrypter** object like this:

```
encrypter = new Encrypter(r_y_inverse, s_1_inverse, s_2_inverse, U, L_inverse, sigma,
    n);
```

- **r_y_inverse** is the resolution for plant output signal y.
- **s_1_inverse** and **s_2_inverse** are matrix scaling factors. $1/s_1$ scales matrices G, R, and J. $1/s_2$ scales H and J.
- **U** is the range size of controller output u.
- **L_inverse** is signal scaling factor for controller input signals.
- **sigma**(σ) is standard deviation for Gaussian distribution of noise to inject to the ciphertext.
- **n** is ciphertext dimension.

3.2 class EncryptedController

```
controller = new EncryptedController(encm_FGR, encm_HJ, enc_x_con_init, q, actuator);
```

This function returns an encrypted controller object that was built with following parameters.

- **encm_FGR** and **encm_HJ** are state space matrices of the encrypted controller. They are encrypted matrices of converted and then scaled controller.
- **enc_x_con_init** is a ciphertext which has message of the initial state of controller.
- **q** is the size of cipherspace(ex: q:=2^{48}).
- **actuator** must be an actuator object that will be attached to the plant.

3.3 class Actuator

```
actuator = new Actuator(encrypter);
actuator->SetPlant(plant);
actuator->SetController(controller);
```

- **encrypter** must be an **Encrypter** object that will decrypt the controller output signal, and also re-encrypt the signal.
- **plant** is expected to be an **Plant** object that will receive actuator signal from this actuator.
- **controller** is expected to be an **EncryptedController** object that will send an output **u** to this actuator and receive re-encrypted signal $\mathbf{u}'$ in return.

3.4 class Sensor

```
sensor = new Sensor(controller, encrypter);
```

- **controller** is expected to be an **EncryptedController** object that will be get the sensor signal.
- **encrypter** must be an **Encrypter** object that will encrypt the plant output signal.

3.5 class Plant

```
plant = new Plant(sensor);
```

- **sensor** must be a **Sensor** object that will get the plant output signal and process it.

4 Building An Encrypted Controller System

4.1 class SystemBuilder

Class **SystemBuilder** provides default codes for encrypted system construction.

```
SystemBuilder* controlSystem = new SystemBuilder();
```

The following is the process of work that **SystemBuilder** does during above line of code.

1. Read the input file.
   - Get the state-space representation of original controller.
   - Get cryptosystem settings.
2. Build an **EncryptedController** object.
   - Build a controller with parameters read from input file.

     ```
     BuildController(T_s, F_precision, G_precision, H_precision, J_precision);
     ```

     (X_precision: precision of the matrix X from the input file)
   - Convert the controller to have integer state matrix, and then scale it by powers of 10 to quantize other matrices(process explained in 5.1).
   - Encrypt the controller.
3. Simulate one step control routine and compare that time cost with sampling time $\boldsymbol{T_s}$. Adjust $\boldsymbol{n}$ so that it has the largest possible value while guaranteeing control time constraint.
4. Construct an **Encrypter** object with $\boldsymbol{n}$ which was determined above.
5. Construct an **Actuator**, a **Sensor**, and a **Plant** objects and connect each of them.

4.2 Control loop

You can start control loop with following line of codes and the loop will continue forever until forced termination.

```
SystemBuilder* controlSystem = new SystemBuilder();
controlSystem->ControlLoop();
```

5 Brief Explanations For Processes

5.1 Converting process of controller

To encrypt a controller by LWE-based cryptosystem, its state-space representation must be composed of only integer matrices to be encrypted. Further, its state matrix(ex: F) needs to be integer matrix without scaling, because otherwise you can't construct infinite operations.
Following routine shows the process of building an encrypted controller from the original controller model.

Colors of matrix symbols represent:
- real matrix
- integer matrix
- encrypted matrix

$$x(t+1) = F'x(t) + G'y(t)$$
$$u(t) = H'x(t) + J'y(t)$$

$\Downarrow$
Convert state-space to the observable canonical form
$\Downarrow$

$$x(t+1) = F''x(t) + G''y(t)$$
$$u(t) = Hx(t) + Jy(t)$$
$$x(t+1) = \begin{bmatrix} -a_1 & 1 & 0 & \cdots & 0 \\ -a_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_{l_1-1} & 0 & 0 & \cdots & 1 \\ -a_{l_1} & 0 & 0 & \cdots & 0 \end{bmatrix} x(t) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{l_1} \end{bmatrix} y(t)$$
$$u(t) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix} x(t) + \begin{bmatrix} b_0 \end{bmatrix} y(t)$$

$\Downarrow$
Introduce u as a new input to make state matrix F an integer matrix
$\Downarrow$

$$x(t+1) = Fx(t) + Gy(t) + Ru(t)$$
$$u(t) = Hx(t) + Jy(t)$$
$$x(t+1) = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} x(t) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{l_1} \end{bmatrix} y(t) + \begin{bmatrix} -a_1 \\ -a_2 \\ \vdots \\ -a_{l_1} \end{bmatrix} u(t)$$

$\Downarrow$

Scale matrices except for F, with $1/s_1$ and $1/s_2$,
so that they become integer matrices

$\Downarrow$

$$\bar{x}(t+1) = F \cdot \bar{x}(t) + \bar{G} \cdot \bar{y}(t) + \bar{R} \cdot \bar{u}(t)$$
$$\bar{u}(t) = \bar{H} \cdot \bar{x}(t) + \bar{J} \cdot \bar{y}(t)$$

$\Downarrow$

Encrypt the matrices and signals

$\Downarrow$

$$\mathbf{x}(t+1) = \mathbf{F} \cdot \mathbf{x}(t) + \mathbf{G} \cdot \mathbf{y}(t) + \mathbf{R} \cdot \mathbf{u}(t)$$
$$\mathbf{u}(t) = \mathbf{H} \cdot \mathbf{x}(t) + \mathbf{J} \cdot \mathbf{y}(t)$$