The 13th Asian Control Conference (ASCC 2022)

Tutorial: Homomorphic Encryption and Its Application to Feedback Control

ECLib: Encrypted Control Library for Python

Kaoru Teranishi

The University of Electro-Communications
Research Fellow of Japan Society for the Promotion of Science

ECLib

Encrypted control requires the knowledge of control theory and cryptography.

For control engineers, studying cryptography is challenging.

ECLib (**E**ncrypted **C**ontrol **Lib**rary)

- **Python** library for numerical simulation of encrypted control
- **Open source software**
- Easy to use for researchers and students

https://github.com/KaoruTeranishi/EncryptedControl

Why Python?

In control society, MATLAB is a popular tool for design and analysis of control systems. Unfortunately, MATLAB does not support **bignum arithmetic**.

Cryptography is typically based on very big integers, and so MATLAB is not suitable for encrypted control.

Python

- The most popular programming language in 2021 [1]
- Built-in support for bignums
- Many OSS packages for developing codes
- **Interactive coding and computing**

[1] https://spectrum.ieee.org/top-programming-languages/

Jupyter Notebook

Jupyter notebook is a web application for interactive computing.

https://jupyter.org

Jupyter notebook documents can contain codes, markdown texts, equations, and plots.
(≒ MATLAB live editor)

Visual Studio Code supports Jupyter Notebook.

Goal of This Presentation

This presentation is a brief introduction to Python and ECLib to develop numerical simulations of encrypted control.

The usage of ECLib will be explained through two applications.

- ElGamal encryption
- Encrypted PI control

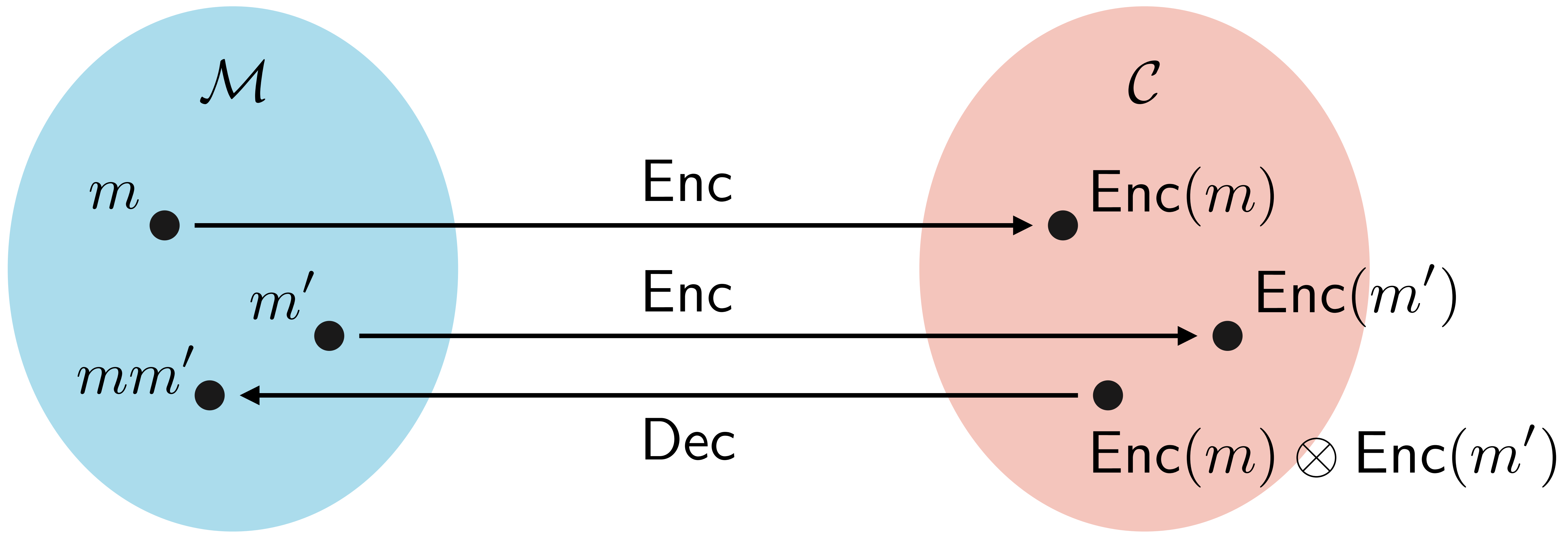
Slides and codes are available here:

https://github.com/KaoruTeranishi/EncryptedControl

Example 1: ElGamal encryption

Algorithms

$$\begin{aligned}
\mathsf{KeyGen} &: 1^\lambda \mapsto (\mathsf{pk}, \mathsf{sk}) = ((\mathbb{G}, g, p, q, h), s) \\
\mathsf{Enc} &: (\mathsf{pk}, m) \mapsto c = (c_1, c_2) = (g^r \bmod p, mh^r \bmod p) \\
\mathsf{Dec} &: (\mathsf{sk}, c) \mapsto m = {c_1}^{-s} c_2 \bmod p \\
\mathsf{Eval} &: (c, c') \mapsto (c_1 c_1' \bmod p, c_2 c_2' \bmod p)
\end{aligned}$$



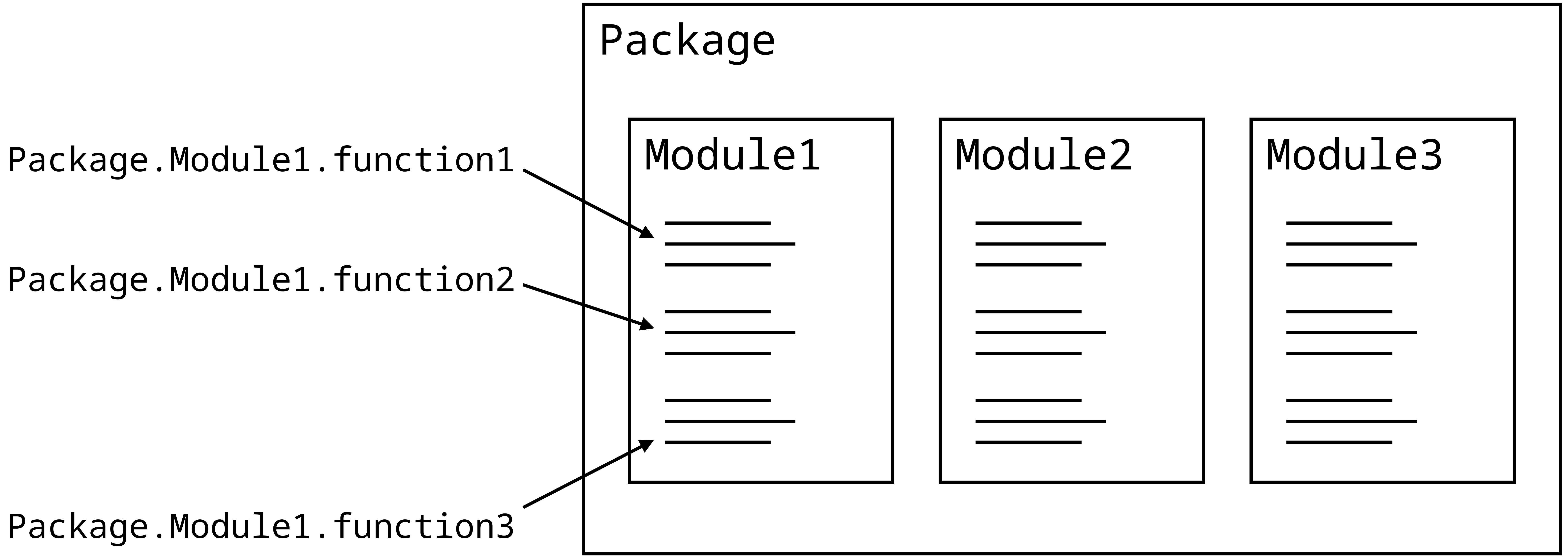
Import Packages

```python
from eclib.elgamal import *
```

Import all functions from `elgamal` module in `eclib` package.

Module: Script file consisting of some commands. Module can be used from external codes.

Package: A way structuring multiple modules.


Package
Module1
Module2
Module3
Package.Module1.function1
Package.Module1.function2
Package.Module1.function3

Package Installation

We can use many OSS packages for developing codes.

NumPy SciPy matplotlib

pandas ECLib CVXPY

If you want to install ECLib, run `pip` command on your terminal.

```
$ pip install eclib
```

python™ Package Index

pip

- De facto standard of package management system in Python.
- The system installs packages from online repository, Python Package Index (PyPI).
- The system can also update and uninstall packages.

Key Generation

```
 3 key_length = 20
 4 params, pk, sk = keygen(key_length)
 5
 6 print(f'\u03BB = {key_length}')
 7 print(f'p = {params.p}')
 8 print(f'q = {params.q}')
 9 print(f'g = {params.g}')
10 print(f'h = {pk}')
11 print(f's = {sk}')
```

Result:

```
λ = 20
p = 1049963
q = 524981
g = 3
h = 33790
s = 110369
```

Line 3: Assign `20` to `key_length`.

Line 4: Generate public and secret keys.

`params`: public parameters p, q, g

`pk`: public key h

`sk`: secret key s

Each public parameter can be referred by the dot operator.

Encryption and Decryption (1/2)

```
13 delta = 0.01
14 x1 = 1.23
15 m1 = encode(params, x1, delta)
16 c1 = encrypt(params, pk, m1)
17 n1 = decrypt(params, sk, c1)
18 y1 = decode(params, n1, delta)
19
20 print(f'x1 = {x1}')
21 print(f'm1 = Ecd(x1) = {m1}')
22 print(f'c1 = Enc(m1) = {c1}')
23 print(f'n1 = Dec(c1) = {n1}')
24 print(f'y1 = Dcd(n1) = {y1}')
```

Result:

```
x1 = 1.23
m1 = Ecd(x1) = 122
c1 = Enc(m1) = [40345, 874487]
n1 = Dec(c1) = 122
y1 = Dcd(n1) = 1.22
```

Line 15: Encode `x1 = 1.23` using `delta = 0.01`.

$$m_1 = \text{Ecd}_\Delta(x_1), \ x_1 = 1.23, \ \Delta = 0.01$$

Line 16: Encrypt `m1`.

$$c_1 = \text{Enc}(m_1)$$

Line 17: Decrypt `c1`.

$$n_1 = \text{Dec}(c_1)$$

Line 18: Decode `n1` using `delta`.

$$y_1 = \text{Dcd}_\Delta(n_1)$$

Encryption and Decryption (2/2)

```
26 x2 = -4.56
27 c2 = enc(params, pk, x2, delta)
28 y2 = dec(params, sk, c2, delta)
29
30 print(f'x2 = {x2}')
31 print(f'c2 = Enc(Ecd(x2)) = {c2}')
32 print(f'y2 = Dcd(Dec(c2)) = {y2}')
```

Result:

```
x2 = -4.56
c2 = Enc(Ecd(x2)) = [424381, 935668]
y2 = Dcd(Dec(c2)) = -4.57
```

Line 27: We can encrypt float numbers by enc instead of using encode and encrypt.

$$c_2 = \mathrm{Enc}(\mathrm{Ecd}_\Delta(x_2)),\ x_2 = -4.56$$

Line 28: Similarly, we can use dec for decryption.

$$y_2 = \mathrm{Dcd}_\Delta(\mathrm{Dec}(c_2))$$

Homomorphic Computation

```
34 x3 = x1 * x2
35 c3 = mult(params, c1, c2)
36 y3 = dec(params, sk, c3, delta ** 2)
37
38 print(f'x3 = x1 * x2 = {x3}')
39 print(f'c3 = Mult(c1, c2) = {c3}')
40 print(f'y3 = Dcd(Dec(c3)) = {y3}')
```

Result:

```
x3 = x1 * x2 = -5.6088
c3 = Mult(c1, c2) = [954767, 686157]
y3 = Dcd(Dec(c3)) = -5.5754
```

Line 34: Assign x1 multiplied by x2 to x3.

Line 35: Compute homomorphic multiplication.

$$c_3 = c_1 \otimes c_2$$

Line 36: Decrypt and decode c3 using `delta` squared because `delta` is accumulated every homomorphic multiplication.

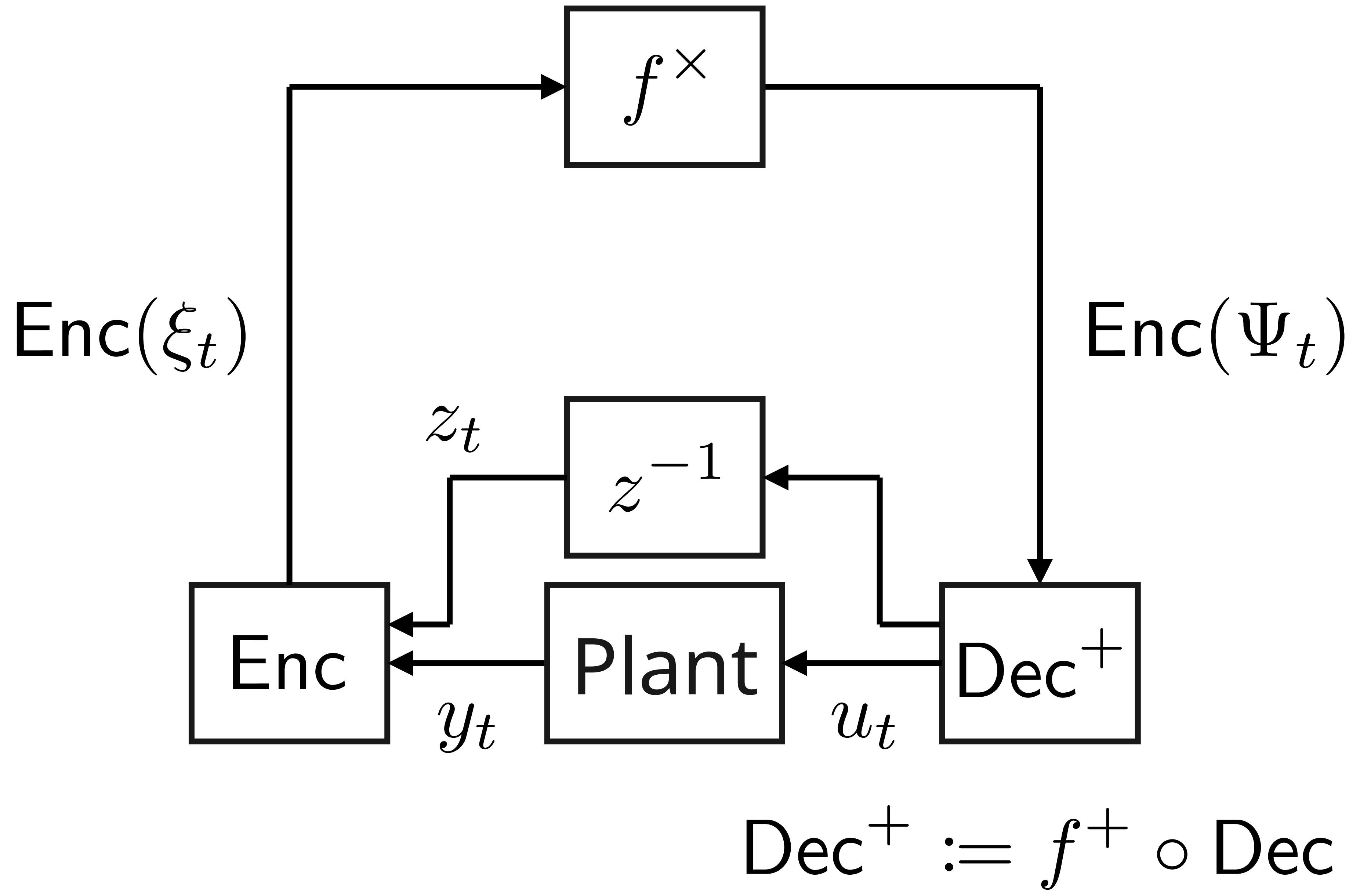
$$y_3 = \mathsf{Dcd}_{\Delta^2}(\mathsf{Dec}(c_3))$$

Example 2: Encrypted PI Control

Plant ($T_s = 0.1$ s)

$$x_{t+1} = \begin{bmatrix} 0.35 & -0.16 \\ 0.13 & 0.98 \end{bmatrix} x_t + \begin{bmatrix} 0.031 \\ 0.004 \end{bmatrix} u_t$$

$$y_t = \begin{bmatrix} 0 & 1 \end{bmatrix} x_t$$



PI controller ($K_p = 15.34$, $K_i = 15.62$)

$$w_{t+1} = w_t + \begin{bmatrix} T_s & -T_s \end{bmatrix} \begin{bmatrix} r_t \\ y_t \end{bmatrix}$$

$$u_t = K_i w_t + \begin{bmatrix} K_p & -K_p \end{bmatrix} \begin{bmatrix} r_t \\ y_t \end{bmatrix}$$

$$\iff \begin{bmatrix} w_{t+1} \\ u_t \end{bmatrix} = \begin{bmatrix} 1 & T_s & -T_s \\ K_i & K_p & -K_p \end{bmatrix} \begin{bmatrix} w_t \\ r_t \\ y_t \end{bmatrix}$$

$$\psi_t = \begin{bmatrix} w_{t+1} \\ u_t \end{bmatrix}, \ \Phi = \begin{bmatrix} 1 & T_s & -T_s \\ K_i & K_p & -K_p \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & -0.1 \\ 15.62 & 15.34 & -15.34 \end{bmatrix}, \ \xi_t = \begin{bmatrix} w_t \\ r_t \\ y_t \end{bmatrix}$$

Import Packages

```python
1 from eclib.elgamal import *
2 from eclib.colors import *
3 import eclib.figsetup
4 import numpy as np
5 import numpy.linalg as la
6 from control.matlab import *
7 import matplotlib.pyplot as plt
```

Import modules in ECLib, NumPy, Python-control, and matplotlib.

Line 4: Import numpy module as the short name, np.
After this line, we can refer numpy module as np.

Lines 5, 7: Import `linalg` and `pyplot` modules from numpy and `matplotlib` packages as `la` and `plt`, respectively.

Simulation Settings (1/2)

```
 9 # sampling time
10 Ts = 0.1
11
12 # simulation setting
13 simulation_time = 10
14 t = np.linspace(0, simulation_time - Ts, int(simulation_time / Ts))
```

In this simulation, a sampling time is 0.1 s, and the total simulation time is 10 s.

Line 14: Generate an array including sampling points.

`linspace(start, stop, num)` returns an array consisting of num elements, which are equally spaced in the interval from `start` to `stop`.

Simulation Settings (2/2)

Example:

```
a = np.linspace(0, 10, 5)
print(f'a = {a}’)
```

Result:

```
a = [ 0.   2.5  5.   7.5 10. ]
```

```
t = np.linspace(0, simulation_time - Ts, int(simulation_time / Ts))
```

`t` is an array consisting of `int(simulation_time / Ts)` elements

in the interval from `0` to `simulation_time` - `Ts`.

Here, `simulation_time` is 10, and `Ts` is 0.1. Hence, `t` is an array,

`t = [0, 0.1, 0.2, 0.3, ... 9.7, 9.8, 9.9]`

Plant

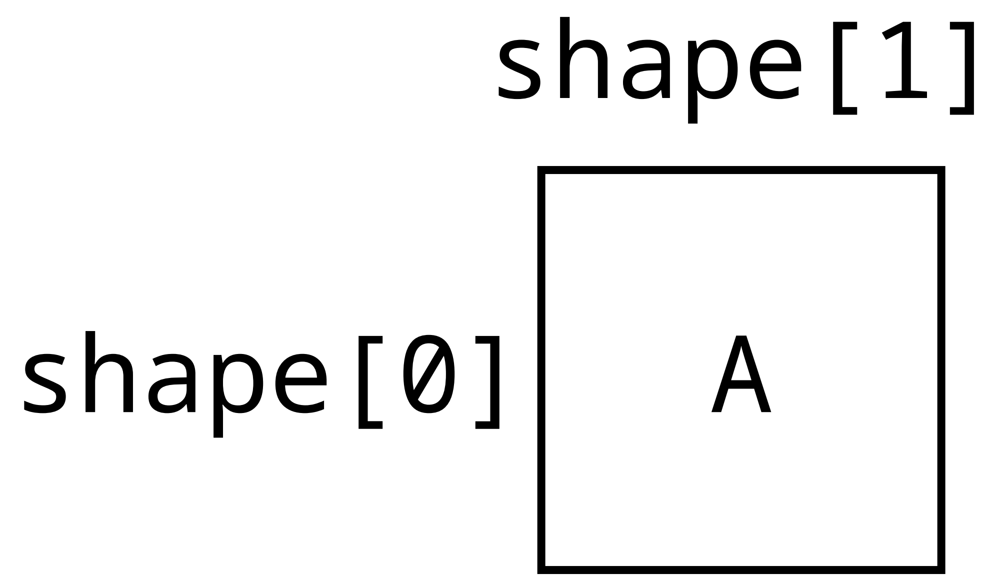
```
16 # plant (continuous time)
17 A = np.array([[-10, -2.5],
18               [  2,  0  ]])
19 B = np.array([[0.5],
20               [0  ]])
21 C = np.array([0, 1])
22 D = np.array(0)
23
24 # plant (discrete time)
25 sys = c2d(ss(A, B, C, D), Ts)
26 A = sys.A
27 B = sys.B
28 C = sys.C
29 D = sys.D
30
31 # dimension
32 n = A.shape[0]
33 m = B.shape[1]
34 l = C.shape[0]
```

Lines 17-22: Define plant parameters.

$$\dot{x}(\tau) = \begin{bmatrix} -10 & -2.5 \\ 2 & 0 \end{bmatrix} x(\tau) + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} u(\tau)$$
$$y(\tau) = \begin{bmatrix} 0 & 1 \end{bmatrix} x(\tau)$$
$$A = \begin{bmatrix} -10 & -2.5 \\ 2 & 0 \end{bmatrix},\ B = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix},\ C = \begin{bmatrix} 0 & 1 \end{bmatrix},\ D = 0$$

Lines 25-29: Discretize plant parameters with `Ts`.

Lines 32-34: Assign signal dimensions to `n`, `m`, and `l`.



Controller

```
36 # controller
37 Kp = 15.34
38 Ki = 15.62
39
40 Phi = np.array([[ 1, Ts, -Ts],
41                 [Ki, Kp, -Kp]])
```

Line 40: Define controller parameter matrix.

$$w_{t+1} = w_t + \begin{bmatrix} T_s & -T_s \end{bmatrix} \begin{bmatrix} r_t \\ y_t \end{bmatrix}$$

$$u_t = K_i w_t + \begin{bmatrix} K_p & -K_p \end{bmatrix} \begin{bmatrix} r_t \\ y_t \end{bmatrix}$$

$$\iff \quad \begin{bmatrix} w_{t+1} \\ u_t \end{bmatrix} = \underset{\texttt{Phi}}{\begin{bmatrix} 1 & T_s & -T_s \\ K_i & K_p & -K_p \end{bmatrix}} \begin{bmatrix} w_t \\ r_t \\ y_t \end{bmatrix}$$

Controller Encryption

```
43 # cryptosystem
44 key_length = 256
45 params, pk, sk = keygen(key_length)
46
47 # scaling parameter
48 delta = 0.0001
49
50 # controller encryption
51 Phi_enc = enc(params, pk, Phi, delta)
```

Line 51: Encrypt controller parameter matrix.
enc can be used for a scalar, vector, and matrix.
The dimensions of return value are decided automatically.

```
Phi_enc =                                    [c1, c2]
                                            ↙
┌                                                                                                            ┐
  enc(params,pk,Phi[0,0],delta)   enc(params,pk,Phi[0,1],delta)   enc(params,pk,Phi[0,2],delta)

  enc(params,pk,Phi[1,0],delta)   enc(params,pk,Phi[1,1],delta)   enc(params,pk,Phi[1,2],delta)
└                                                                                                            ┘
```

Signals (1/3)

```
# state
x = np.zeros([len(t) + 1, n])
x_ = np.zeros([len(t) + 1, n])

# input
u = np.zeros([len(t), m])
u_ = np.zeros([len(t), m])

# output
y = np.zeros([len(t), l])
y_ = np.zeros([len(t), l])

# reference
r = np.zeros([len(t), l])
r_ = np.zeros([len(t), l])

# controller state
w = np.zeros([len(t) + 1, l])
w_ = np.zeros([len(t) + 1, l])
```

Define arrays for saving signal data.

`x`, `u`, `y`, `r`, and `w` are used for unencrypted control.

Variables with underscore are used for encrypted control.

`np.zeros(shape)` returns an array consisting of zeros. `shape` is an array of integers.

`len(t)` is a length of `t`.

Example:

```
a = np.zeros([2, 3])
print(a)
```

Result:

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
b = len(np.zeros(5))
print(b)
```

```
5
```

Signals (2/3)

```
73 # controller input
74 xi = np.zeros([len(t), 3 * l])
75 xi_ = np.zeros([len(t), 3 * l])
76 xi_enc = [[[0, 0] for j in range(3 * l)] for i in range(len(t))]
77 
78 # controller output
79 psi = np.zeros([len(t), l + m])
80 psi_ = np.zeros([len(t), l + m])
81 psi_enc = [[[[0, 0] for k in range(3 * l)] for j in range(l + m)] for i in range(len(t))]
```

Line 76: Define 2-dimensional array of which elements are `[0, 0]` by using list comprehension.

`range(stop)` represents an immutable sequence numbers from `0` to `stop` - 1.

Example:

```
a = range(5)
print(list(a))
```

Result:

```
[0, 1, 2, 3, 4]
```

Signals (3/3)

List comprehension is a way to create various lists.

Example:

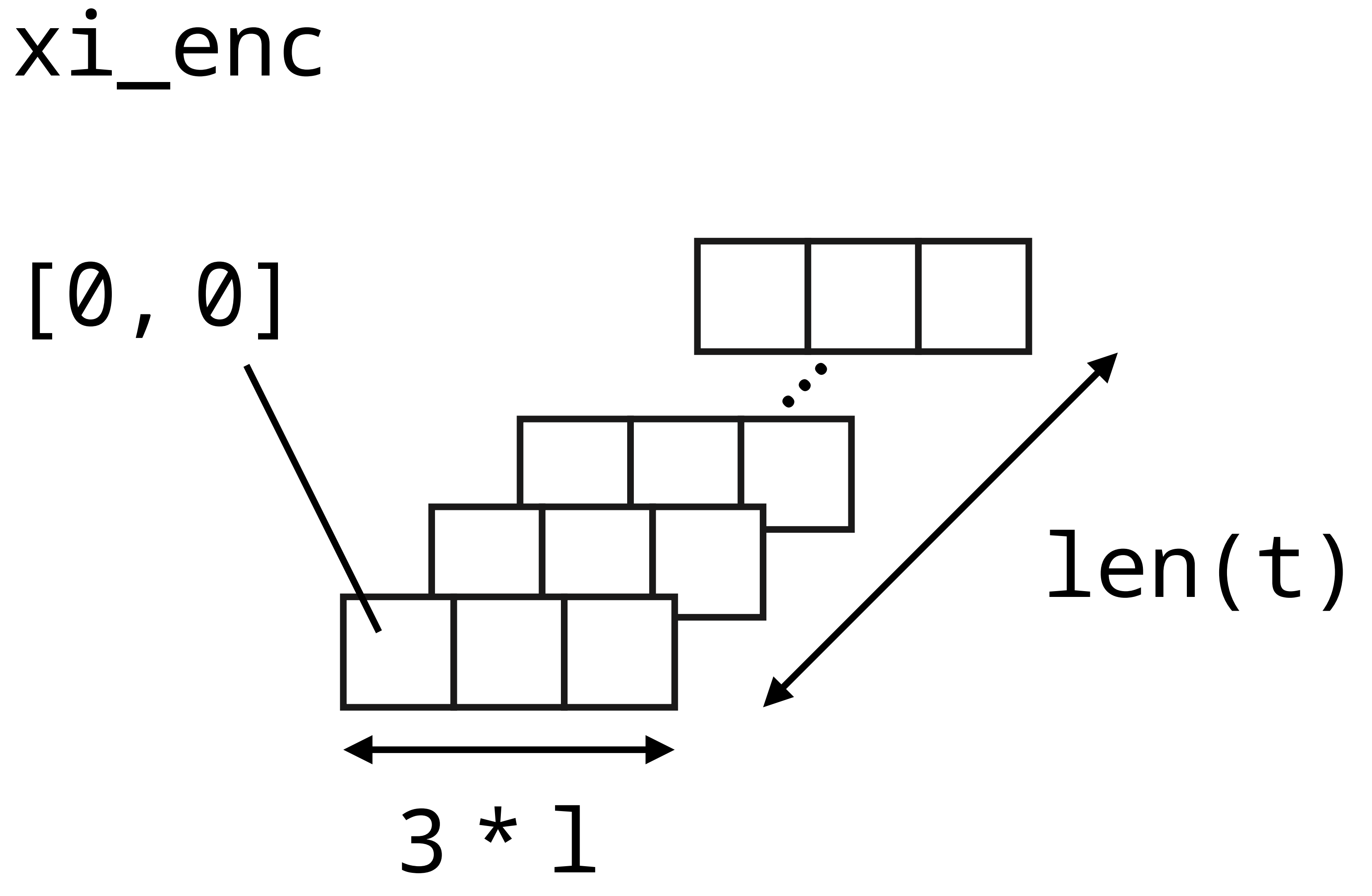
```python
a = [x ** 2 for x in range(5)]
print(f'a = {a}')
```

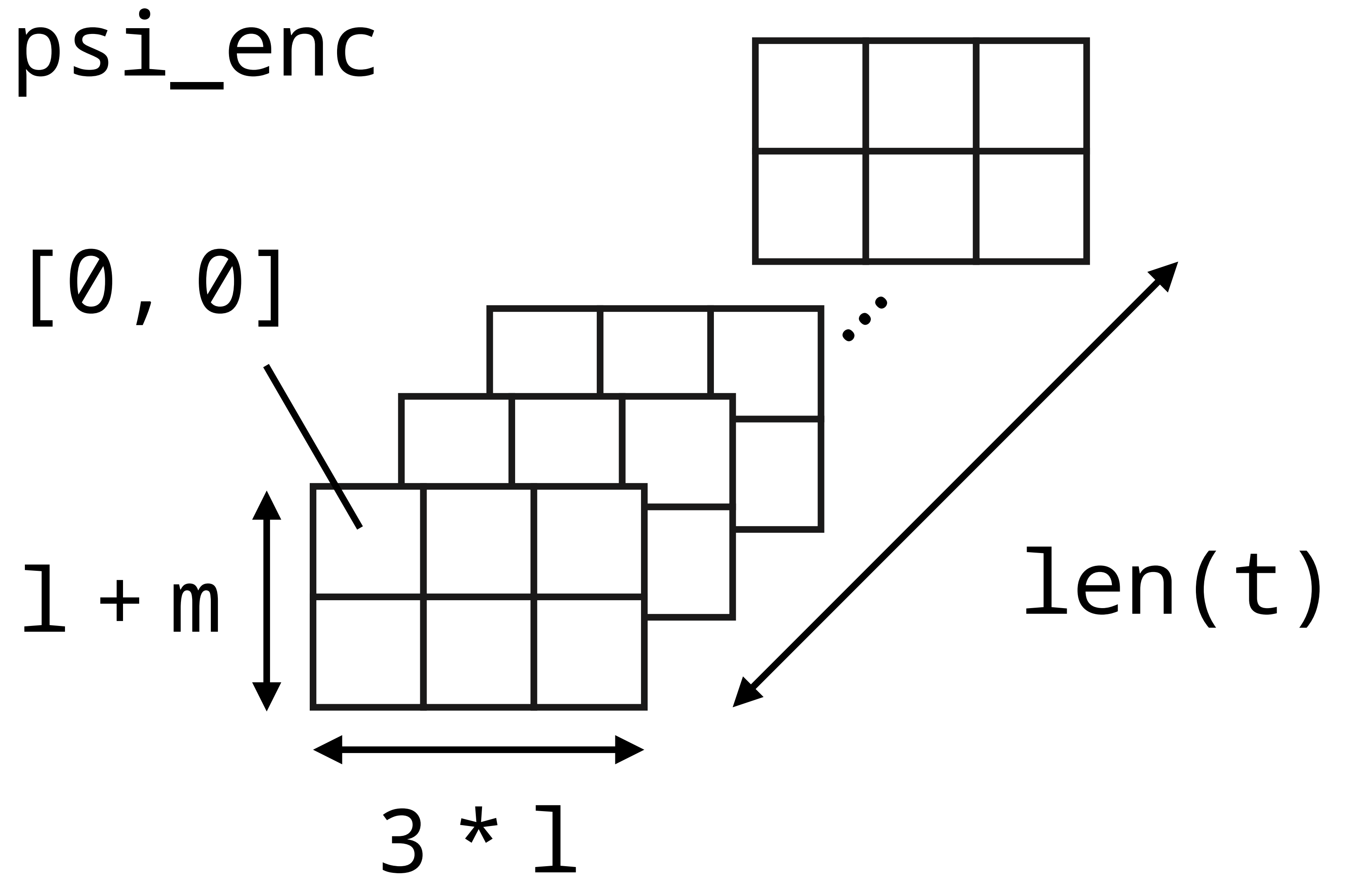
Result:

```
a = [0, 1, 4, 9, 16]
```

```python
xi_enc = [[[0, 0] for j in range(3 * l)] for i in range(len(t))]
```

```python
psi_enc = [[[[0, 0] for k in range(3 * l)] for j in range(l + m)] for i in range(len(t))]
```





Simulation of Unencrypted Control (1/2)

```
83 # unencrypted control
84 for k in range(len(t)):
85     # reference
86     r[k] = 1
87     # sensor measurement
88     y[k] = C @ x[k]
89     # controller input
90     xi[k,0:l] = w[k]
91     xi[k,l:2*l] = r[k]
92     xi[k,2*l:3*l] = y[k]
93     # controller computation
94     psi[k] = Phi @ xi[k]
95     # controller output
96     w[k+1] = psi[k,0:l]
97     u[k] = psi[k,l:l+m]
98     # plant update
99     x[k+1] = A @ x[k] + B @ u[k]
```

Line 84: For loop from k = `0` to k = `len(t) - 1`.

Line 88: Compute plant output. `@` is the operator for matrix product.

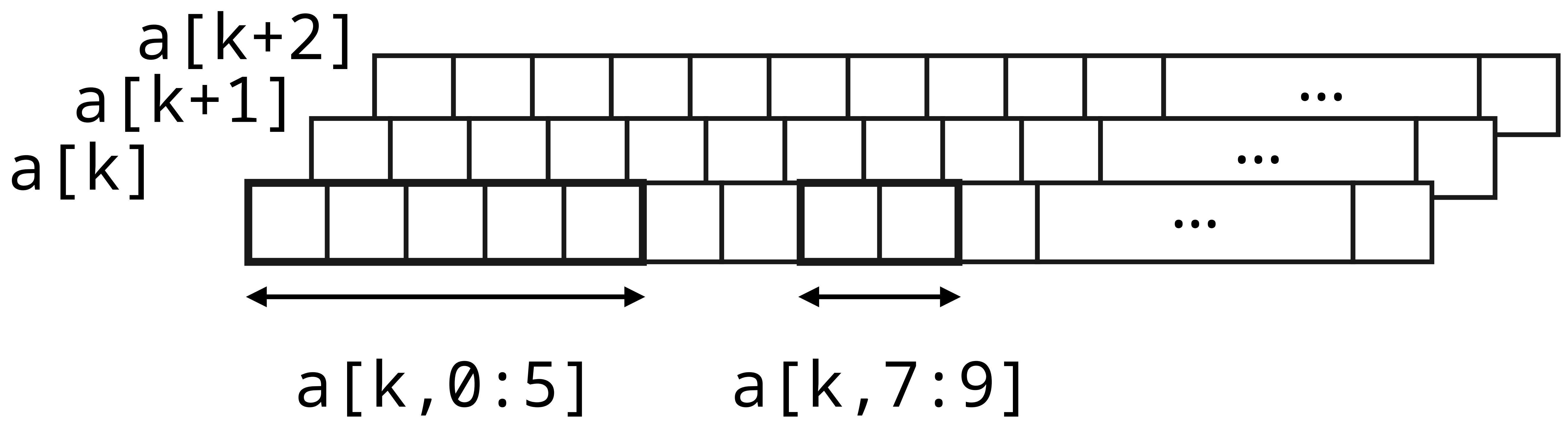
$$y_k = C x_k$$

Lines 90-92: Construct a controller input.

$$\xi_k = \begin{bmatrix} w_k^\top & r_k^\top & y_k^\top \end{bmatrix}^\top$$

Colon is an operator for slicing an array.

Example (2-dimensional array):



Simulation of Unencrypted Control (2/2)

```
83 # unencrypted control
84 for k in range(len(t)):
85     # reference
86     r[k] = 1
87     # sensor measurement
88     y[k] = C @ x[k]
89     # controller input
90     xi[k,0:l] = w[k]
91     xi[k,l:2*l] = r[k]
92     xi[k,2*l:3*l] = y[k]
93     # controller computation
94     psi[k] = Phi @ xi[k]
95     # controller output
96     w[k+1] = psi[k,0:l]
97     u[k] = psi[k,l:l+m]
98     # plant update
99     x[k+1] = A @ x[k] + B @ u[k]
```

Line 94: Compute a controller output.

$$\psi_k = \Phi\xi_k$$

Lines 96-97: Decompose a controller output.

$$\begin{bmatrix} w_{k+1} \\ u_k \end{bmatrix} = \psi_k$$

Line 99: Update a plant state.

Simulation of Encrypted Control

```python
101 # encrypted control
102 for k in range(len(t)):
103     # reference
104     r_[k] = 1
105     # sensor measurement
106     y_[k] = C @ x_[k]
107     # controller input
108     xi_[k,0:l] = w_[k]
109     xi_[k,l:2*l] = r_[k]
110     xi_[k,2*l:3*l] = y_[k]
111     xi_enc[k] = enc(params, pk, xi_[k], delta)
112     # encrypted controller computation
113     psi_enc[k] = mult(params, Phi_enc, xi_enc[k])
114     # controller output
115     psi_[k] = dec_add(params, sk, psi_enc[k], delta ** 2)
116     w_[k+1] = psi_[k,0:l]
117     u_[k] = psi_[k,l:l+m]
118     # plant update
119     x_[k+1] = A @ x_[k] + B @ u_[k]
```

Line 111: Encrypt a controller input.

$$\xi_{\mathsf{Enc}} = \mathsf{Enc}(\mathsf{Ecd}_{\Delta}(\xi_k))$$

Line 113: Homomorphic computation

$$\Psi_{\mathsf{Enc}} = f^{\times}(\Phi_{\mathsf{Enc}}, \xi_{\mathsf{Enc}})$$

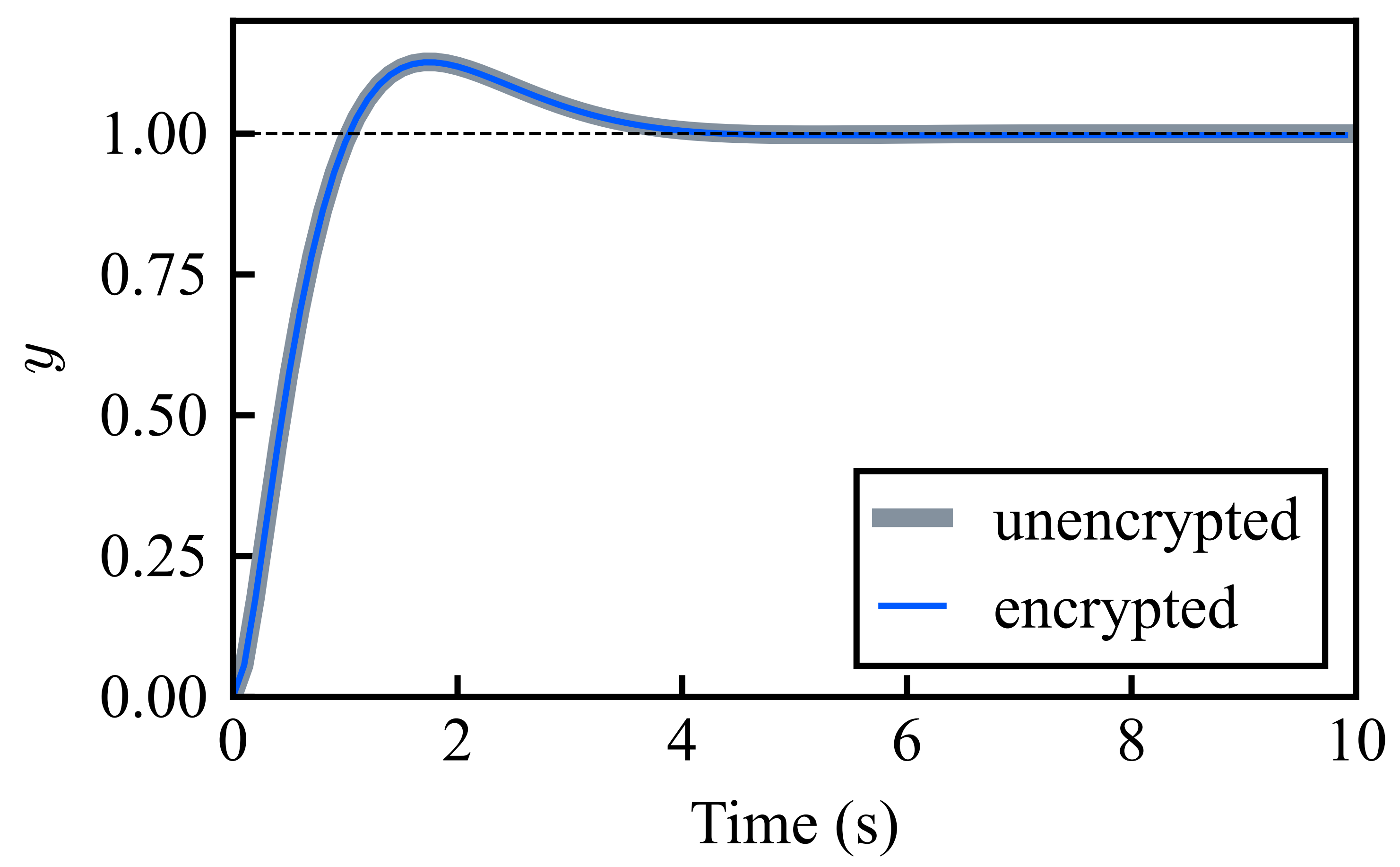
Line 115: Decrypt a controller output.

$$\psi_k = \mathsf{Dcd}_{\Delta^2}(\mathsf{Dec}^{+}(\Psi_{\mathsf{Enc}}))$$

Figures

```
130 plt.figure()
131 plt.plot(t, y, linestyle='-', color=gray, linewidth=3.0, label='unencrypted')
132 plt.plot(t, y_, linestyle='-', color=blue, linewidth=1.0, label='encrypted')
133 plt.plot(t, r, linestyle='--', color=black, linewidth=0.5)
134 plt.xlabel('Time (s)')
135 plt.ylabel(r'$y$')
136 plt.xlim(0, simulation_time)
137 plt.ylim(0, 1.2)
138 plt.legend(loc='lower right')
```

```
266 plt.show()
```



Line 130: Create a new figure.

Lines 131-133: Plot y, y_, and `r`.

Lines 134-137: Set labels and ranges of x- and y-axises.

Lines 138: Set a legend position.

Lines 266: Show the figure.

Conclusion

- ECLib is an OSS Python library for researchers and students.
- ECLib provides functions of homomorphic encryption to develop numerical simulation codes of encrypted control.
- Supported cryptosystems in ECLib are ElGamal encryption and Paillier encryption.
- We welcome your suggestions, bug reports, and contributions.
  https://github.com/KaoruTeranishi/EncryptedControl

ECLib