HEFactory
Technical Exemplified Documentation

José Cabrero-Holgueras

February 25, 2023

Contents

1. Introduction

This document aims to describe the basic user side functionality of HEFactory through code snippets.

2. Basic Operations

The basic operations allowed by HEFactory are: Addition (§ 2.1), Subtraction (§ 2.2), Multiplication (§ 2.3), Rotation (§ 2.4), and Negation (§ 2.5).

2.1 Addition Operations

2.1.1 Plaintext Addition

```
plaintext = 5
with CGManager() as cgm:
    encrypted_val = CGSym(cgm, plaintext)
    res = encrypted_val + 5
    cgm.output([res])
```

2.1.2 Ciphertext Addition

```
plaintext_a = 5
plaintext_b = 10
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    encrypted_b = CGSym(cgm, plaintext_b)
    res = encrypted_a + encrypted_b
    cgm.output([res])
```

2.1.3 Plaintext Vector Addition

```
plaintext_a = np.array([1, 2, 3, 4])
plaintext_b = np.array([4, 3, 2, 1])
```

```
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    res = encrypted_a + plaintext_b
    cgm.output([res])
```

2.2 Subtraction Operations

2.2.1 Plaintext Subtraction

```
plaintext = 5
with CGManager() as cgm:
    encrypted_val = CGSym(cgm, plaintext)
    res = encrypted_val - 5
    cgm.output([res])
```

2.2.2 Ciphertext Subtraction

```
plaintext_a = 5
plaintext_b = 10
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    encrypted_b = CGSym(cgm, plaintext_b)
    res = encrypted_a - encrypted_b
    cgm.output([res])
```

2.2.3 Plaintext Vector Subtraction

```
plaintext_a = np.array([1, 2, 3, 4])
plaintext_b = np.array([4, 3, 2, 1])
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    res = encrypted_a - plaintext_b
    cgm.output([res])
```

2.3 Multiplication Operations

2.3.1 Plaintext Multiplication

```
plaintext = 5
with CGManager() as cgm:
    encrypted_val = CGSym(cgm, plaintext)
    res = encrypted_val * 5
    cgm.output([res])
```

2.3.2 Ciphertext Multiplication

```
plaintext_a = 5
plaintext_b = 10
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    encrypted_b = CGSym(cgm, plaintext_b)
    res = encrypted_a * encrypted_b
    cgm.output([res])
```

2.3.3 Plaintext Vector Multiplication

```
plaintext_a = np.array([1, 2, 3, 4])
plaintext_b = np.array([4, 3, 2, 1])
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    res = encrypted_a * plaintext_b
    cgm.output([res])
```

2.4 Rotation Operation

2.4.1 Left Rotation

```
plaintext_a = np.array([1, 2, 3, 4])
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    res = encrypted_a << 4
    cgm.output([res])
```

2.4.2 Right Rotation

```
plaintext_a = np.array([1, 2, 3, 4])
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    res = encrypted_a >> 4
    cgm.output([res])
```

2.5 Negation Operation

2.5.1 Negation

```
plaintext_a = np.array([1, 2, 3, 4])
with CGManager() as cgm:
    encrypted_a = CGSym(cgm, plaintext_a)
    res = - encrypted_a
    cgm.output([res])
```

3. Advanced Ciphertext Operations

This section describes advanced operations supported by the engine such as: Inversion (§ 3.1), Square Root (§ 3.2), Absolute Value (§ 3.3), Convolution (§ 3.4), Matrix-Vector Multiplication (§ 3.5) and Vector Accumulation (§ 3.6).

3.1 Inversion Operation

3.1.1 Inversion

```
x = 5
x_, bits = scale_down(x)
with CGManager() as cgm:
    encrypted_val = CGSym(cgm, x_)
    res = encrypted_val.inv()
    cgm.output([res])
```

3.2 Square Root Operation

3.2.1 Square Root

```
x = np.array([4, 5, 6, 7, 8])
x = x * x
x_, bits = scale_down(x)

with CGManager() as cgm:
    encrypted_val = CGSym(cgm, x_)
    res = encrypted_val.sqrt()
    cgm.output([res])
```

3.3 Absolute Value Operation

3.3.1 Absolute Value

```python
x = [-4, -4.5, 5, -6, -7, 8]
x_, bits = scale_down(x)

with CGManager() as cgm:
    enc_b = CGSym(cgm, x_)
    res = enc_b.abs()
    cgm.output([enc_b, res])
```

3.4 Convolution Operation

3.4.1 Convolution

```python
input_vector = np.arange(1, (1 << 8) + 1, 1).reshape(1 << 4, 1 << 4)
kernel = np.ones(9).reshape(3, 3) * 1/9

with CGManager() as cgm:
    # This code includes encrypted result transformation
    encrypted_vector = CGArray(cgm, input_vector)
    res = encrypted_vector.convolution(kernels = [kernel],
                                        paddings = [(0, 0)],
                                        strides = [(1,1)])
    cgm.output([res])
```

3.5 Matrix-Vector Multiplication Operation

3.5.1 Matrix-Vector Multiplication

```


matrix = np.array([[1, 2, 3, 4],
                    [5, 6, 7, 8],
                    [9, 10, 11, 12]])
vector = np.array([1, 2, 3, 4])


expected_res = matrix.dot(vector)
with CGManager() as cgm:
    encrypted_vector = CGArray(cgm, vector)
    res = encrypted_vector.matrix_vector_mul(matrix)
    cgm.output([res])

```

3.6 Vector Accumulation Operation

3.6.1 Vector Accumulation

```
input_vector = np.arange(1, n + 1, 1)
plaintext_v = np.arange(n, 0, -1)

with CGManager() as cgm:
    encrypted_vector = CGArray(cgm, input_vector)
    a = encrypted_vector * plaintext_v
    res = a.log_accumulate()
    cgm.output([res])

```