

HEFactory

Technical Exemplified Documentation

José Cabrero-Holgueras

February 25, 2023

Contents

1. Introduction	2
2. Basic Operations	2
2.1 Addition Operations	2
2.1.1 Plaintext Addition	2
2.1.2 Ciphertext Addition	2
2.1.3 Plaintext Vector Addition	2
2.2 Subtraction Operations	3
2.2.1 Plaintext Subtraction	3
2.2.2 Ciphertext Subtraction	3
2.2.3 Plaintext Vector Subtraction	3
2.3 Multiplication Operations	4
2.3.1 Plaintext Multiplication	4
2.3.2 Ciphertext Multiplication	4
2.3.3 Plaintext Vector Multiplication	4
2.4 Rotation Operation	5
2.4.1 Left Rotation	5
2.4.2 Right Rotation	5
2.5 Negation Operation	5
2.5.1 Negation	5
3. Advanced Ciphertext Operations	6
3.1 Inversion Operation	6
3.1.1 Inversion	6
3.2 Square Root Operation	6
3.2.1 Square Root	6
3.3 Absolute Value Operation	7
3.3.1 Absolute Value	7
3.4 Convolution Operation	7
3.4.1 Convolution	7
3.5 Matrix-Vector Multiplication Operation	7
3.5.1 Matrix-Vector Multiplication	8
3.6 Vector Accumulation Operation	8
3.6.1 Vector Accumulation	8

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

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

2.1.2 Ciphertext Addition

```
1 plaintext_a = 5
2 plaintext_b = 10
3 with CGManager() as cgm:
4     encrypted_a = CGSym(cgm, plaintext_a)
5     encrypted_b = CGSym(cgm, plaintext_b)
6     res = encrypted_a + encrypted_b
7     cgm.output([res])
```

2.1.3 Plaintext Vector Addition

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

```

3 with CGManager() as cgm:
4     encrypted_a = CGSym(cgm, plaintext_a)
5     res = encrypted_a + plaintext_b
6     cgm.output([res])

```

2.2 Subtraction Operations

2.2.1 Plaintext Subtraction

```

1 plaintext = 5
2 with CGManager() as cgm:
3     encrypted_val = CGSym(cgm, plaintext)
4     res = encrypted_val - 5
5     cgm.output([res])

```

2.2.2 Ciphertext Subtraction

```

1 plaintext_a = 5
2 plaintext_b = 10
3 with CGManager() as cgm:
4     encrypted_a = CGSym(cgm, plaintext_a)
5     encrypted_b = CGSym(cgm, plaintext_b)
6     res = encrypted_a - encrypted_b
7     cgm.output([res])

```

2.2.3 Plaintext Vector Subtraction

```

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

```

```
5     res = encrypted_a - plaintext_b
6     cgm.output([res])
```

2.3 Multiplication Operations

2.3.1 Plaintext Multiplication

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

2.3.2 Ciphertext Multiplication

```
1 plaintext_a = 5
2 plaintext_b = 10
3 with CGManager() as cgm:
4     encrypted_a = CGSym(cgm, plaintext_a)
5     encrypted_b = CGSym(cgm, plaintext_b)
6     res = encrypted_a * encrypted_b
7     cgm.output([res])
```

2.3.3 Plaintext Vector Multiplication

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

```
6     cgm.output([res])
```

2.4 Rotation Operation

2.4.1 Left Rotation

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

2.4.2 Right Rotation

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

2.5 Negation Operation

2.5.1 Negation

```
1 plaintext_a = np.array([1, 2, 3, 4])
2 with CGManager() as cgm:
3     encrypted_a = CGSym(cgm, plaintext_a)
4     res = - encrypted_a
5     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

```
1 x = 5
2 x_, bits = scale_down(x)
3 with CGManager() as cgm:
4     encrypted_val = CGSym(cgm, x_)
5     res = encrypted_val.inv()
6     cgm.output([res])
7
```

3.2 Square Root Operation

3.2.1 Square Root

```
1 x = np.array([4, 5, 6, 7, 8])
2 x = x * x
3 x_, bits = scale_down(x)
4
5 with CGManager() as cgm:
6     encrypted_val = CGSym(cgm, x_)
7     res = encrypted_val.sqrt()
8     cgm.output([res])
9
```

3.3 Absolute Value Operation

3.3.1 Absolute Value

```
1 x = [-4, -4.5, 5, -6, -7, 8]
2 x_, bits = scale_down(x)
3
4 with CGManager() as cgm:
5     enc_b = CGSym(cgm, x_)
6     res = enc_b.abs()
7     cgm.output([enc_b, res])
8
```

3.4 Convolution Operation

3.4.1 Convolution

```
1 input_vector = np.arange(1, (1 << 8) + 1, 1).reshape(1 << 4, 1 << 4)
2 kernel = np.ones(9).reshape(3, 3) * 1/9
3
4 with CGManager() as cgm:
5     # This code includes encrypted result transformation
6     encrypted_vector = CGArray(cgm, input_vector)
7     res = encrypted_vector.convolution(kernels = [kernel],
8                                       paddings = [(0, 0)],
9                                       strides = [(1,1)])
10    cgm.output([res])
11
```

3.5 Matrix-Vector Multiplication Operation

3.5.1 Matrix-Vector Multiplication

```
1
2
3 matrix = np.array([[1, 2, 3, 4],
4                     [5, 6, 7, 8],
5                     [9, 10, 11, 12]])
6 vector = np.array([1, 2, 3, 4])
7
8
9 expected_res = matrix.dot(vector)
10 with CGManager() as cgm:
11     encrypted_vector = CGArray(cgm, vector)
12     res = encrypted_vector.matrix_vector_mul(matrix)
13     cgm.output([res])
14
```

3.6 Vector Accumulation Operation

3.6.1 Vector Accumulation

```
1 input_vector = np.arange(1, n + 1, 1)
2 plaintext_v = np.arange(n, 0, -1)
3
4 with CGManager() as cgm:
5     encrypted_vector = CGArray(cgm, input_vector)
6     a = encrypted_vector * plaintext_v
7     res = a.log_accumulate()
8     cgm.output([res])
9
```