Fully Homomorphic Encryption Development Ecosystems
Tools, Compilers & Challenges

Alexander Viand*, Patrick Jattke, Anwar Hithnawi

*alexander.viand@inf.ethz.ch
What is FHE?

- Allows data to be processed while remaining encrypted
What is FHE?

- Allows data to be processed while remaining encrypted
What is FHE?

- Allows data to be processed while remaining encrypted

\[ 18 = 3 \times 6 \]
FHE Applications

- Secure Outsourcing

- Private Set Intersection (PSI)

- Private Machine Learning as a Service

Is a gene mutation associated to a disease?

3.63s

Microsoft Edge Password Monitor
40 Years of FHE History

- RSA '78
- GM '82
- El-Gamal '85
- RAD '78

1980

- PHE
- SWHE
- FHE

1990

- Paillier '99
- Benaloh '94
- BGN '05

2000

- SYY '00
- IP '07

2010

- Gentry '09

2020

40 Years of FHE History
The path to adoption

Underlying Math → Development → Deployment

Cryptographic Challenges ↔ Engineering Challenges
### This Work

<table>
<thead>
<tr>
<th>Answer</th>
<th>What makes developing FHE applications hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare</td>
<td>Design decisions in existing FHE compilers and how they address some of FHE’s complexities</td>
</tr>
<tr>
<td>Highlight</td>
<td>Where barriers to entry have been lowered and where they still remain</td>
</tr>
<tr>
<td>Outline</td>
<td>Future directions for FHE tool development</td>
</tr>
</tbody>
</table>
Challenges of Developing FHE Applications

- Underlying Math
- Parameter Selection
- Security

- No If/Else
- No Loops
- No Jumps

- Optimizations
- Approximations
- SIMD Batching

Libraries

Compilers
Cryptographic Challenges

- Underlying Math
- Parameter Selection
- Security
(Ring-)Learning With Errors: It is hard to find $s$ given $c$ and $a$

$$c = a \cdot s + e$$

where $a \leftarrow R_q, e \leftarrow R_q, s \in R_q$

$$R = \mathbb{Z}[X]/(X^n + 1)$$
(Ring-)Learning With Errors: It is hard to find $s$ given $c$ and $a$

$$c = a \cdot s + e$$

where $a \in_{R_q}, e \in_{R_q}, s \in R_q$

$$R = \mathbb{Z}[X]/(X^n + 1)$$

Cryptographic Challenges
(Ring-)Learning With Errors: It is hard to find $s$ given $c$ and $a$

$$c = a \cdot s + e$$

where $a \leftarrow R_q$, $e \leftarrow R_q$, $s \in R_q$

$$R = \mathbb{Z}[X]/(X^n + 1)$$

Encryption

$$c = a \cdot s + \mu + e$$

where $\mu = m \cdot \left\lfloor \frac{q}{t} \right\rfloor$ for $m \in R_t$, $t \ll q$
(Ring-)Learning With Errors: It is hard to find s given c and a

\[ c = a \cdot s + e \]

where \( a \leftarrow R_q, e \leftarrow R_q, s \in R_q \)

\( R = \mathbb{Z}[X]/(X^n + 1) \)

Encryption

\[ c = a \cdot s + \mu + e \]

where \( \mu = m \times \left[ \frac{q}{t} \right] \) for \( m \in R_t, t \ll q \)

Homomorphic Addition:
- Noise increases linearly
(Ring-)Learning With Errors: It is hard to find $s$ given $c$ and $a$

$$c = a \cdot s + e$$

where $a \leftarrow R_q$, $e \leftarrow R_q$, $s \in R_q$

$$R = \mathbb{Z}[X]/(X^n + 1)$$

Encryption

$$c = a \cdot s + \mu + e$$

where $\mu = m \cdot \left\lfloor \frac{q}{t} \right\rfloor$ for $m \in R_t, t \ll q$

Homomorphic Addition / Multiplication

- Noise increases linearly / quadratically

\[
\begin{array}{c}
m \cdot m' \\
(\alpha \cdot \alpha' + \cdots) s \\
\end{array}
\]

\[
\begin{array}{c}
e \cdot e' + \cdots
\end{array}
\]
(Ring-)Learning With Errors: It is hard to find $s$ given $c$ and $a$

$$c = a \cdot s + e \quad \text{where} \quad a \leftarrow R_q, e \leftarrow R_q, s \in R_q$$

$$R = \mathbb{Z}[X]/(X^n + 1)$$

Encryption

$$c = a \cdot s + \mu + e \quad \text{where} \quad \mu = m \ast \left\lfloor \frac{q}{t} \right\rfloor \text{for} \ m \in R_t, t \ll q$$

Homomorphic Addition / Multiplication

- Noise increases linearly / quadratically

If noise grows too large, decryption fails

- Bootstrapping reduces noise homomorphically
(Ring-)Learning With Errors: It is hard to find $s$ given $c$ and $a$

$$c = a \cdot s + e$$
where $a \overset{\$}{\leftarrow} R_q$, $e \overset{\chi}{\leftarrow} R_q$, $s \in R_q$

$$R = \mathbb{Z}[X]/(X^n + 1)$$

Encryption

$$c = a \cdot s + \mu + e$$
where $\mu = m \cdot \left\lfloor \frac{q}{t} \right\rfloor$ for $m \in R_t$, $t \ll q$

Homomorphic Addition / Multiplication

- Noise increases linearly / quadratically

If noise grows too large, decryption fails

- Bootstrapping reduces noise homomorphically
Cryptographic Challenges

- Parameter Selection
  - Based on current knowledge about Learning With Error (LWE) hardness
  - Careful trade-off between efficiency, security and correctness

\[
R = \mathbb{Z}[X]/(X^n + 1) \quad R_q = \frac{R}{q}
\]
Engineering Challenges

- No If/Else
- No Loops
- No Jumps
- Optimizations
- Approximations
- SIMD Batching
FHE Programming Paradigm

No If/Else

No Loops

SIMD Batching

Standard C++

```cpp
int foo(int a, int b) {
    if(a < b) {
        return a * b;
    } else {
        return a + b;
    }
}
```

FHE

```cpp
int foo(int[] a, int[] b) {
    int[] c = less(a, b);
    int[] i = mult(a, b);
    int[] e = add(a, b);
    int[] r;
    for (k=0; k<len(a); ++k)
        r[k] = c[i[k]] + (1-c)*e[k]
    return r;
}
```
FHE Programming Paradigm

No If/Else

Standard C++

```c
int foo(int a, int b) {
    if(a < b) {
        return a * b;
    } else {
        return a + b;
    }
}
```

FHE

```c
int foo(int[] a, int[] b) {
    int[] c = less(a, b);
    int[] i = mult(a, b);
    int[] e = add(a, b);
    int[] r;
    for (k=0; k<len(a); ++k)
        r[k] = c[i[k]] +
               (1-c)*e[k];
    return r;
}
```

No Loops

Standard C++

```c
int foo(int a, int b) {
    int s = 1;
    for(i = 0; i < a; ++i){
        s = s + i*b;
    }
    return s;
}
```

FHE

```c
int foo(int a, int b){
    int s = 1;
    s = less(a, b) *
        (s + 0*b) +
        (1-c) * s;
    s = less(a, b) *
        (s + 1*b) +
        (1-c) * s;
    return s;
}
```

SIMD Batching
FHE Programming Paradigm

- No If/Else

**Standard C++**

```c
int foo(int a, int b) {
    if(a < b) {
        return a * b;
    } else {
        return a + b;
    }
}
```

**FHE**

```c
int foo(int[] a, int[] b) {
    int[] c = less(a, b);
    int[] i = mult(a, b);
    int[] e = add(a, b);
    int[] r;
    for (k=0; k<len(a); ++k)
        r[k] = c[i[k] + (1-c)*e[k]]
    return r;
}
```

- No Loops

**Standard C++**

```c
int foo(int a, int b){
    int s = 1;
    for(i = 0; i < a; ++i){
        s = s + i*b;
    }
    return s;
}
```

**FHE**

```c
int foo(int[] x, int[] y){
    int[] r;
    for(i = 0; i < 6; ++i){
        r[i] = x[i] * y[i];
    }
    return r;
}
```

- SIMD Batching

**Standard C++**

```c
int foo(int a, int b){
    return a * b;
}
```

**FHE**

```c
int foo(int[] x, int[] y){
    int[] r;
    r[0] = x[0] * y[0];
    r[1] = x[1] * y[1];
    return r;
}
```
Approximation

- FHE is most efficient when used over integers
  - However, here + and * only allow polynomials
  - Boolean emulation is powerful, but expensive

- Alternative: Programmable Bootstrapping
  - Technique used in ZAMA's Concrete library
  - Evaluate a Look-Up Table (LUT) during bootstrapping

- Polynomial Approximation
  - Frequently requires too many chained multiplications
  - Limited to univariate functions (single ctxt as input)
Challenges of Developing FHE Applications

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underlying Math</td>
<td>No If/Else</td>
</tr>
<tr>
<td>Parameter Selection</td>
<td>No Loops</td>
</tr>
<tr>
<td>Security</td>
<td>No Jumps</td>
</tr>
<tr>
<td>Optimizations</td>
<td>Approximations</td>
</tr>
<tr>
<td></td>
<td>SIMD Batching</td>
</tr>
</tbody>
</table>
FHE Compilers

- FHE “Compilers” are frequently not actually technically compilers
  - Term used more generally for tools doing high-level to low-level translation
```python
# First Dense Layer
DenseLayer d1(32, input_size);
seal=Ciphertext result;
mvp(*galoisKeys; *encoder, d1.units());
result.d1.weights_as_diags();
imagectxt(result);
seal=Plaintext b1;
encoder->encode(d1.bias());
result.parms_id();
result.scale();
b1;
evaluator->add_plain_inplace(result);:
resultscale_to_next_inplace(result);

// Activation, x -> x^2
evaluator->square_inplace(result);
evaluator->relinearize_inplace(result, *relinKeys);
evaluator->scale_to_next_inplace(result);

def compile():
mpl = EvaProgram('NN (MLP)', vec_size=32 * 32)
with mpl:
image = Input('input_0')
d1 = mvp(weights_1, image)
d1 = d1 + bias_1.toliet()
act1 = d1 * d1
act2 = d2 + bias_2.toliet()
act2 = act2 * d2
Output('output', act2)
Output('output', d1)
```

---

```python
# Second Dense Layer
DenseLayer d2(16, d1.units());
mvp(*galoisKeys; *encoder, d2.units());
d2.weights_as_diags();
result.scale();
seal=Plaintext b2;
encoder->encode(d2.bias());
result.parms_id();
result.scale();
b2;
evaluator->add_plain_inplace(result, b2);
evaluator->scale_to_next_inplace(result);

// Activation, x -> x^2
evaluator->square_inplace(result);
```
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense, Activation

def mnist_mlp_model(input):
    def square_activation(x):
        return x * x

    known_shape = input.get_shape()[1]
    size = np.prod(known_shape).print("size", size)
    y = tf.reshape(input, [-1, size])
    y = Dense(input_shape=[1, 784, units=30, use_bias=True](y)
    y = Activation(square_activation)(y)
    y = Dense(units=10, use_bias=True, name="output")
    known_shape = y.get_shape()[1]
    size = np.prod(known_shape)
    return y

with mlp:
    image = Input('input_0')
    d1 = mlp(weights_1, image)
    act1 = d1 * d1
    d2 = mlp(weights_2, act1)
    act2 = d2 * d2
    Output('output', act2)
    Output('output', d1)```
Input Language

### SEAL (library)

```python
// First Dense Layer
DenseLayer d1(32, input_size);
seal::Ciphertext result;
mvp("galoisKeys", *evaluator, *encoder, d1.input_size(),
    d1.weights_as_diags(), image_ctxt, result);

// Activation, x -> x^2
evaluator->sqr_inplace(result);

def diag(matrix, m, n, mat_r, r_k) {
    size_t r = 0;
    for (k in range(log2_n_div_m)) {
        return r_k * m;
    }
}

def mvp(ptxt, m, n, ptx, n_div_m) {
    if (m * n || diagonals.size() || n_div_m) {
        throw invalid_argument("Matrix must not be empty, and diagonals vector must have size m! ");
    }

    // Activation, x -> x^2
    evaluator->sqr_inplace(ciphertext);

    // Duplication of certain values necessary for the SIMD layout
    size_t t
    evaluator->sqr_inplace(ciphertext);

    // Second Dense Layer
    DenseLayer d2(16, d1.units());
    mvp("galoisKeys", *evaluator, *encoder, d2.input_size(),
        d2.weights_as_diags(), image_ctxt, result);
    evaluate->sqr_inplace(ciphertext);

    // Activation, x -> x^2
    evaluator->sqr_inplace(result);
}
```

```c
void mvp(const seal::GaloisKeys &galois_keys, seal::Evaluator &evaluator, seal::CKKSEncoder &encoder,
    size_t m, size_t n, std::vector<vec> diagonals, const seal::Ciphertext &ctxt, seal::Ciphertext &enc_result) {
    if (m == 0 || m == diagonals.size()) {
        throw invalid_argument("Matrix must not be empty, and diagonals vector must have size m!");
    }

    // Activation, x -> x^2
    evaluator->rotate_vector_inplace(ctxt_rotated_v, m, galois_keys);

    // Duplication of certain values necessary for the SIMD layout
    size_t t
    evaluator->rotate_vector_inplace(ctxt_rotated_v, m, galois_keys);

    // Second Dense Layer
    DenseLayer d2(16, d1.units());
    mvp("galoisKeys", *evaluator, *encoder, d2.input_size(),
        d2.weights_as_diags(), image_ctxt, result);
    evaluate->rotate_vector_inplace(ctxt_rotated_v, m, galois_keys);
    evaluator->rotate_vector_inplace(ctxt_rotated_v, m, galois_keys);
    evaluator->add_inplace(ctxt_rotated_r, ctxt_rotated_r);
}
```
### SEAL (library)

- First Dense Layer
  - DenseLayer d1(32, input_size);
  - seal=Ciphertext result;
  - mvp=galoisKeys, evaluator, encode(d1 weights_as_diags), imasPlaintext b1;
  - encoder=encode(d1 bias!), result;
  - evaluator=decode_plain_inplace(result);
  - evaluator=decode_to_next_inplace(result);

- Activation: x \rightarrow x^2
  - evaluator=square_inplace(result);
  - evaluator=rotate_vector(result);
  - evaluator=rescale_to_next_inplace(result);

- Duplication of certain values needed:
  - seal=Plaintext mask; encoder=encode(mask);
  - evaluator=decode_plain_inplace(result, mask);
  - evaluator=decode_to_next_inplace(result, mask);

### EVA

- def diag(matrix_d):
  - 

### nGraph-HE

- import numpy as np

### SEALion

- Libraries and compilers target very different audiences
- Consider UX beyond programming

---

### DISCONNECT BETWEEN LOW- AND HIGH-LEVEL LANGUAGES

<table>
<thead>
<tr>
<th>SEAL (library)</th>
<th>EVA</th>
<th>nGraph-HE</th>
</tr>
</thead>
<tbody>
<tr>
<td>// First Dense Layer DenseLayer d1(32, input_size); seal=Ciphertext result; mvp=galoisKeys, evaluator, encode(d1 weights_as_diags), imasPlaintext b1; encoder=encode(d1 bias!), result; evaluator=decode_plain_inplace(result); evaluator=decode_to_next_inplace(result);</td>
<td>def diag(matrix_d):</td>
<td>import numpy as np</td>
</tr>
<tr>
<td>// Activation: x \rightarrow x^2 evaluator=decode_plain_inplace(result); evaluator=rotate_vector(result); evaluator=rescale_to_next_inplace(result);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>// Duplication of certain values needed: seal=Plaintext mask; encoder=encode(mask); evaluator=decode_plain_inplace(result, mask); evaluator=decode_to_next_inplace(result, mask);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Libraries and compilers target very different audiences

### Consider UX beyond programming
FHE Compilers

- FHE “Compilers” are frequently not actually technically compilers
  - Term used more generally for tools doing high-level to low-level translation
Tool Space

- Where do compilers apply optimizations?

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Prog Opt</th>
<th>Circuit Gen</th>
<th>Circuit Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALCHEMY</td>
<td>![Image]</td>
<td>![Image]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cingulata</td>
<td>![Image]</td>
<td></td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>E³</td>
<td>![Image]</td>
<td></td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td>EVA</td>
<td>![Image]</td>
<td></td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>Marble</td>
<td>![Image]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ramparts</td>
<td>![Image]</td>
<td>![Image]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHET</td>
<td>![Image]</td>
<td></td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td>nGraph-HE</td>
<td>![Image]</td>
<td></td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>SEALion</td>
<td>![Image]</td>
<td></td>
<td></td>
<td>![Image]</td>
</tr>
</tbody>
</table>
Chi-squared tests are common statistical tests
- Here: Pearson’s Goodness-of-Fit test
- Used in Genome-Wide Association Studies (GWAS) [LLN14]
- Can be rearranged to require only + and x

\[ \alpha = (4N_0 N_2 - N_1^2)^2 \]
\[ \beta_1 = 2(2N_0+N_1)^2 \]
\[ \beta_2 = (2N_0+N_1)(2N_2+N_1) \]
\[ \beta_3 = 2(2N_2+N_1)^2 \]
Evaluation: Machine Learning Inference

- MNIST – handwritten digit recognition
  - Simple task used as a reference benchmark in ML
  - Given an image of a digit, recognize a number in 0-9

- Approximate Activation Function with $x^2$ [GDK+16]

Current State of the Art

Performance: Wide gap between naïve implementations & expert solutions

Current Tools: Solve individual aspects, but fall short of the wider vision

Open Issues: Hardware Accelerators, Intermediate Representation
## Future Directions for FHE Tool Development

<table>
<thead>
<tr>
<th>E2E</th>
<th>Toolchains must address all aspects of FHE development</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automate</strong></td>
<td>Translation between imperative paradigm and FHE</td>
</tr>
<tr>
<td><strong>Optimize</strong></td>
<td>Identify &amp; exploit opportunities for SIMD parallelism</td>
</tr>
</tbody>
</table>