Fully Homomorphic Encryption Development Ecosystems

Tools, Compilers & Challenges

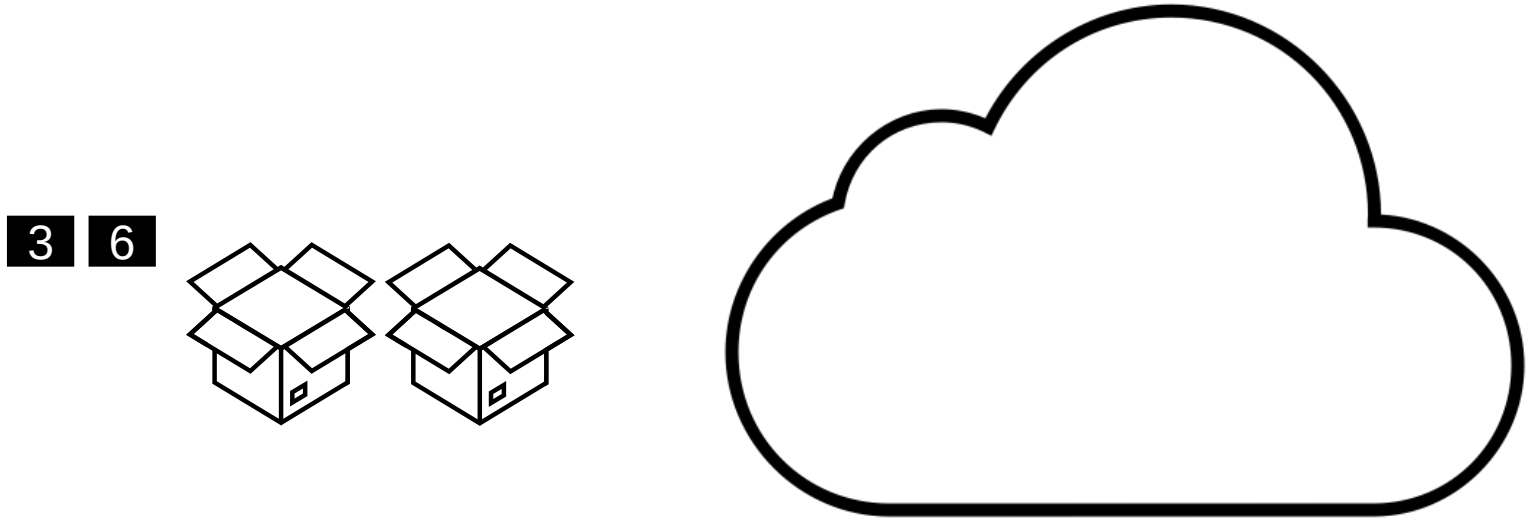
Alexander Viand*, Patrick Jattke, Anwar Hithnawi

ETH zürich

*alexander.viand@inf.ethz.ch

What is FHE?

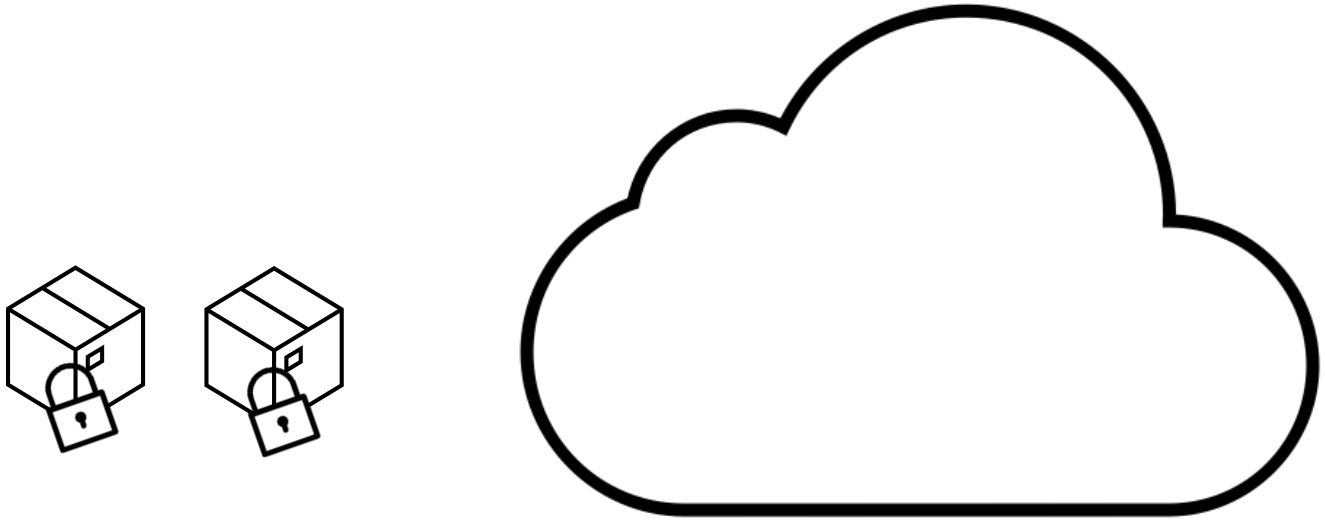
- Allows data to be processed while remaining encrypted



What is FHE?

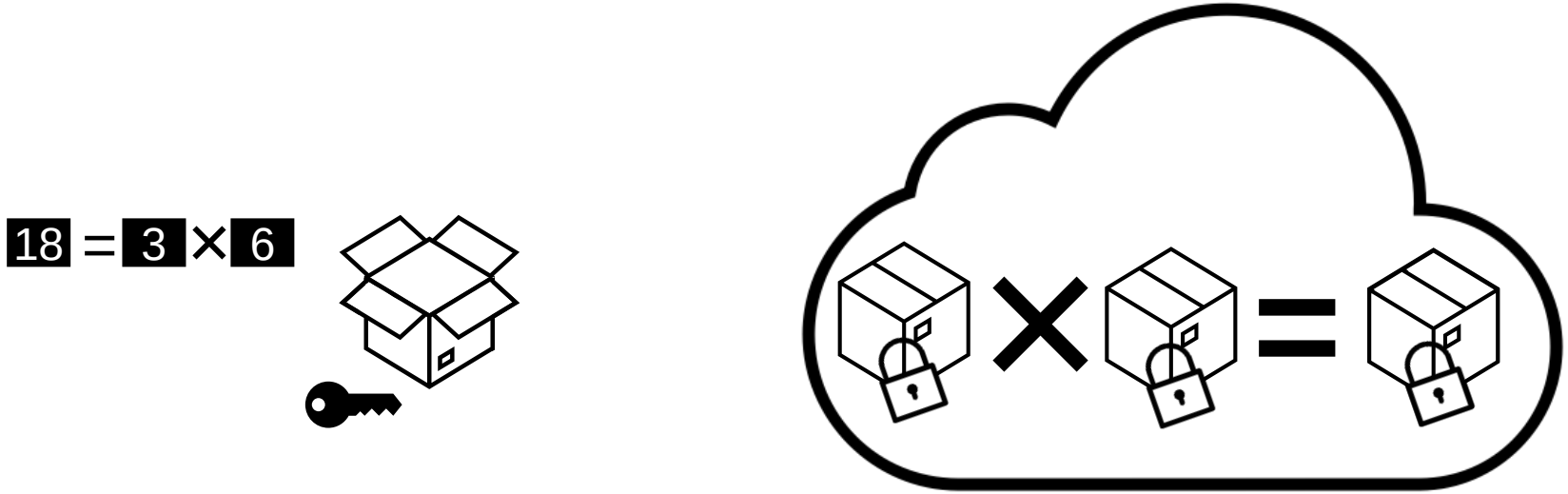
ETH zürich

- Allows data to be processed while remaining encrypted



What is FHE?

- Allows data to be processed while remaining encrypted



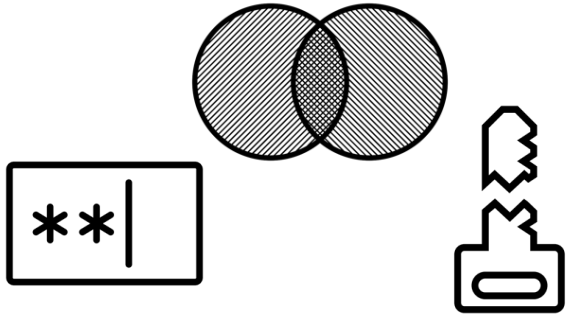
FHE Applications

ETH zürich

- Secure Outsourcing

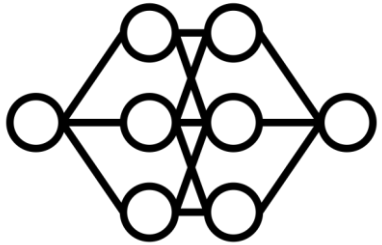
Is a gene mutation associated to a disease?

- Private Set Intersection (PSI)



Microsoft Edge Password Monitor

- Private Machine Learning as a Service



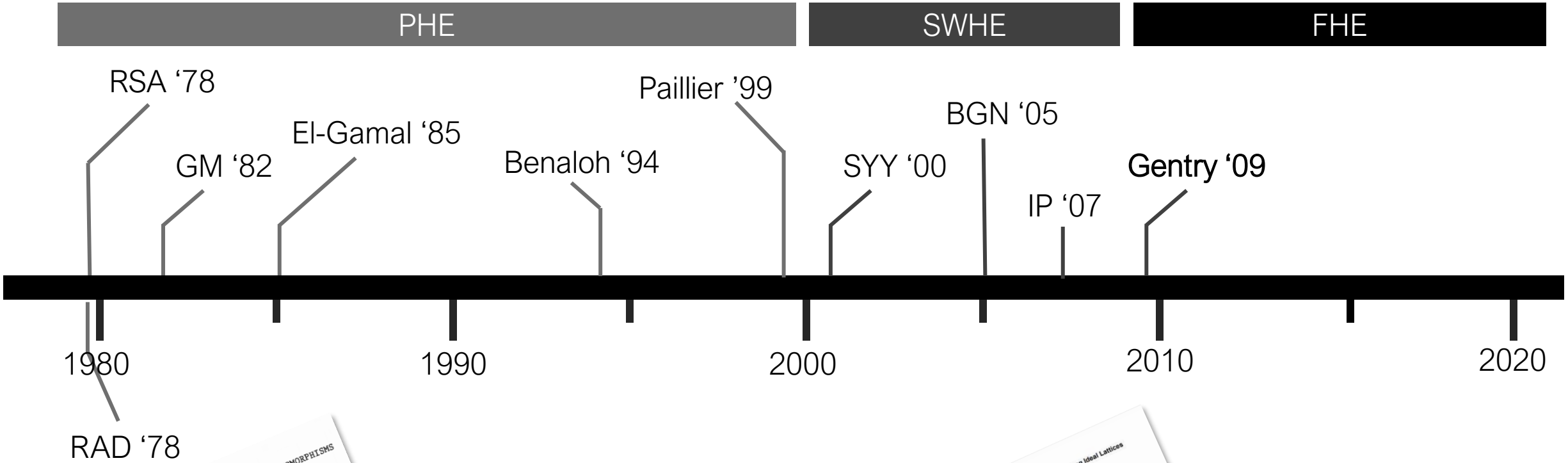
NuCypher

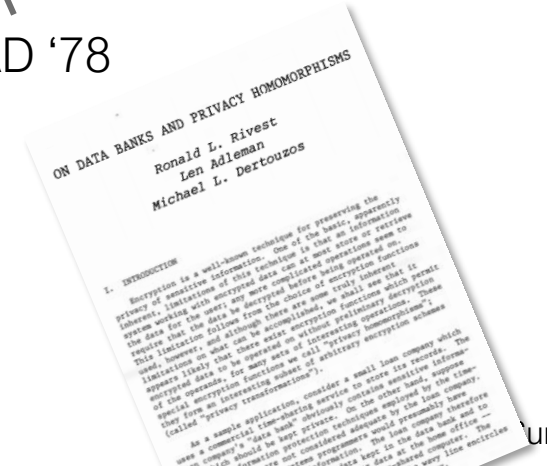
ZAMA

Duality

40 Years of FHE History

ETH zürich




ON DATA BANKS AND PRIVACY HOMOMORPHISMS

Ronald L. Rivest
Len Adleman
Michael L. Dertouzos

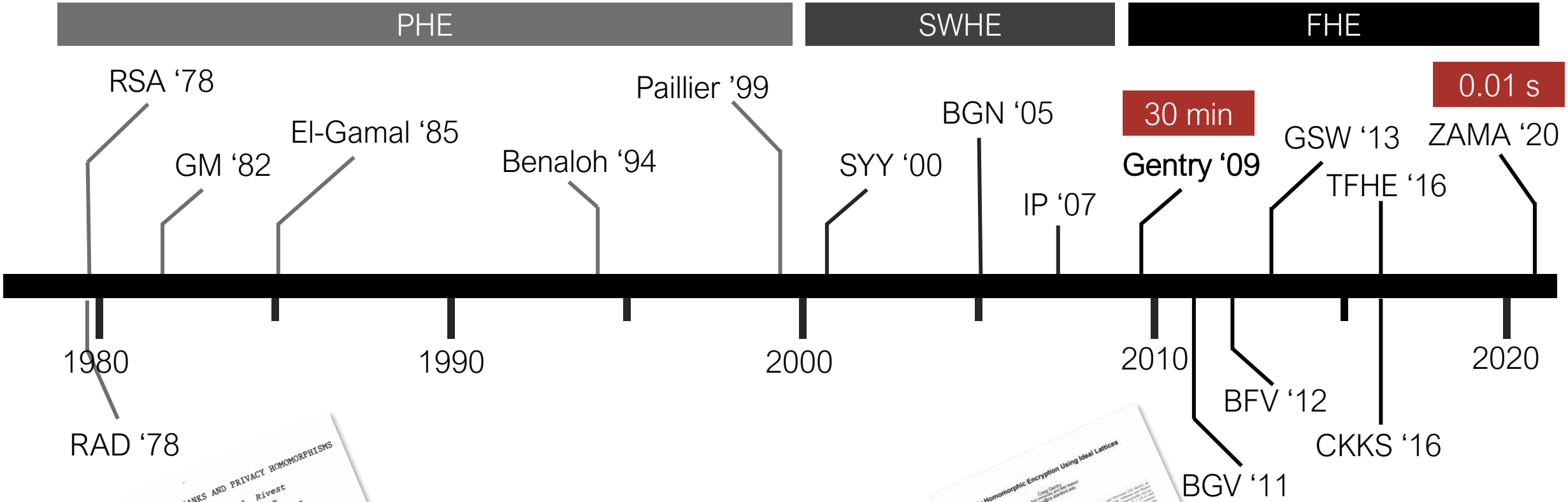

Fully Homomorphic Encryption Using Ideal Lattices

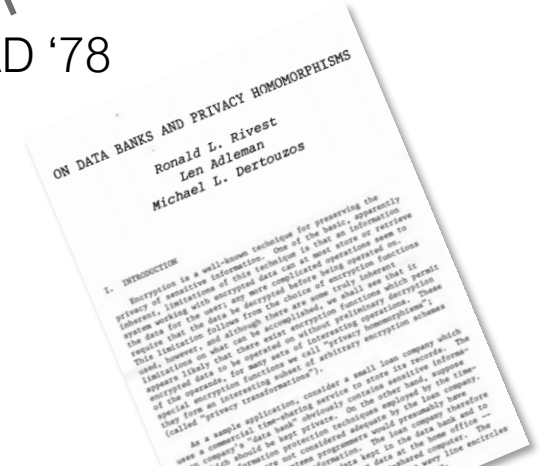
Craig Gentry

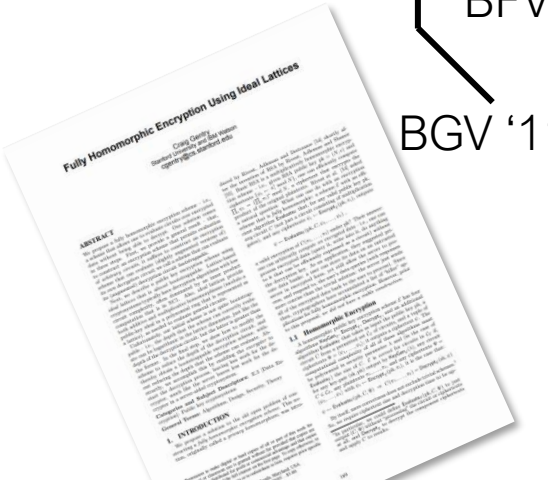
urvey on Homomorphic Encryption Schemes: Theory and Impleme . Surv. 51, 4 (Jul. 2018), 1–35

40 Years of FHE History

ETH zürich




ON DATA BANKS AND PRIVACY HOMOMORPHISMS
Ronald L. Rivest
Len Adleman
Michael L. Dertouzos


Fully Homomorphic Encryption Using Ideal Lattices

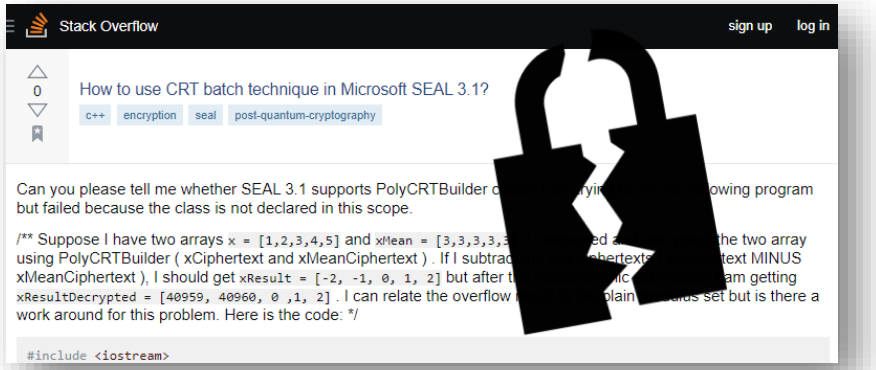
The path to adoption

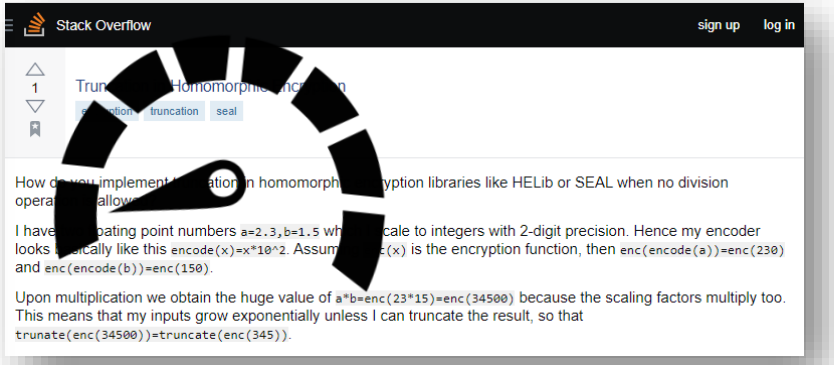
ETH zürich

Underlying Math → Development → Deployment

← Cryptographic Challenges →

← Engineering Challenges →





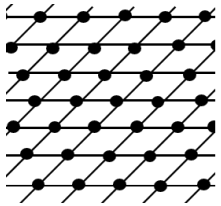
This Work

ETH zürich

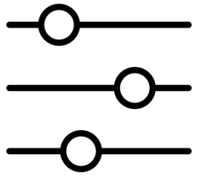
Answer	What makes developing FHE applications hard
Compare	Design decisions in existing FHE compilers and how they address some of FHE's complexities
Highlight	Where barriers to entry have been lowered and where they still remain
Outline	Future directions for FHE tool development

Challenges of Developing FHE Applications

ETH zürich



Underlying Math

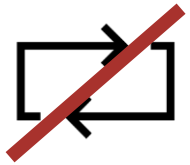


Parameter Selection

Security

Libraries

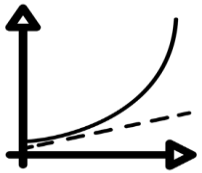
No If/Else



No Loops

No Jumps

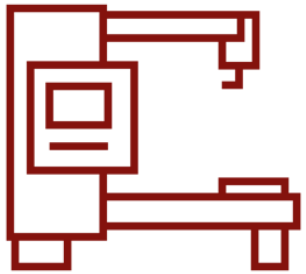
Optimizations



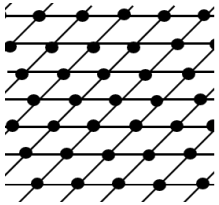
Approximations

SIMD Batching

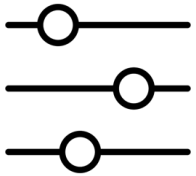
Compilers



Cryptographic Challenges



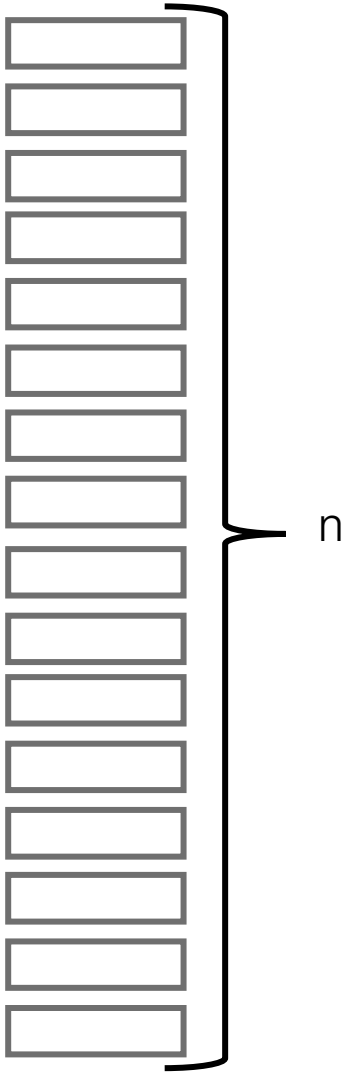
Underlying Math



Parameter Selection

Security

Cryptographic Challenges



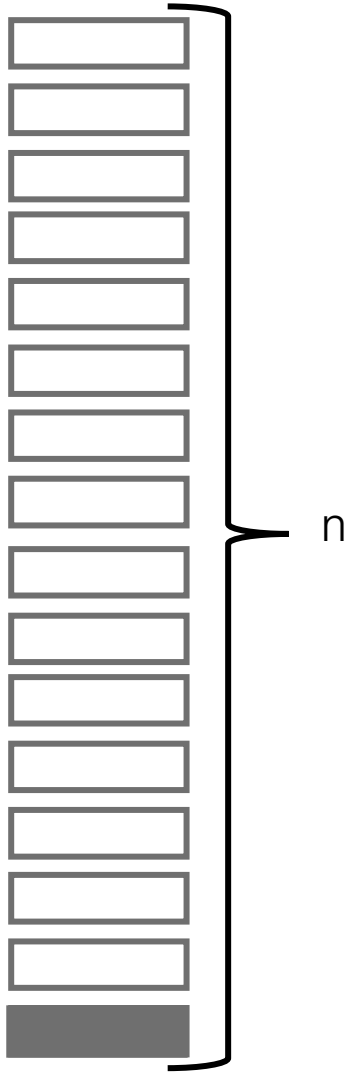
(**R**ing-)**L**earning **W**ith **E**rrors: It is hard to find s given c and a

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

$$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$$

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

Cryptographic Challenges



(**R**ing-)**L**earning **W**ith **E**rrors: It is hard to find s given c and a

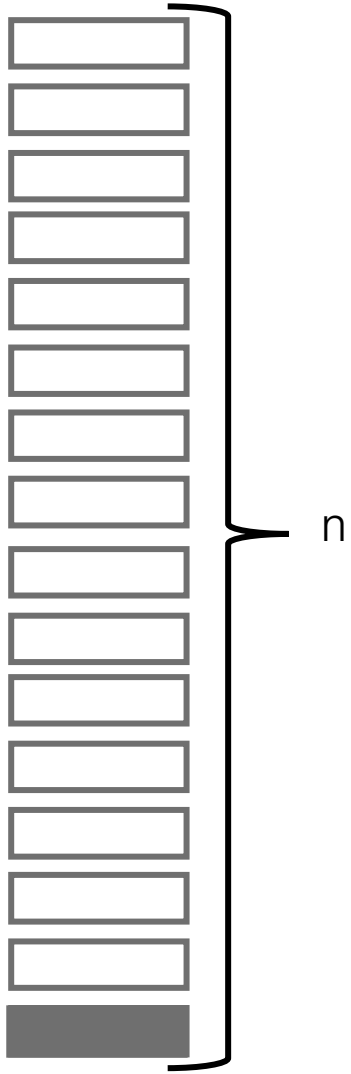
$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$

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

Cryptographic Challenges

ETH zürich



(**R**ing-)**L**earning **W**ith **E**rrors: It is hard to find s given c and a

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$

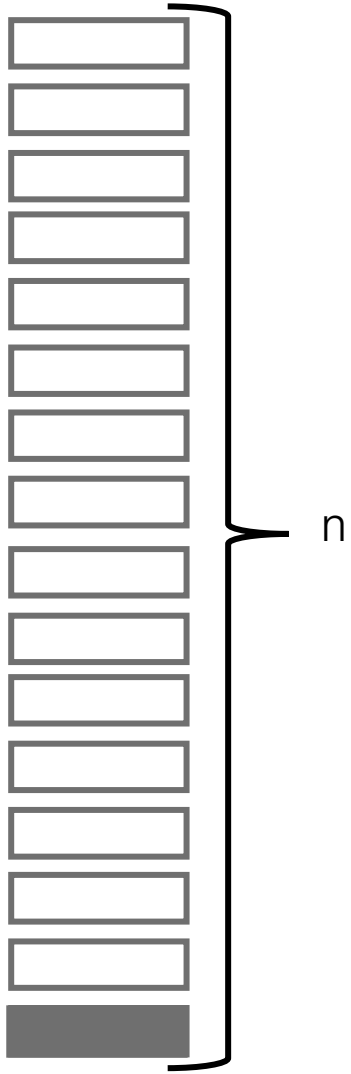
$R = \mathbb{Z}[X]/(X^n + 1)$

Encryption

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{\mu} + \boldsymbol{e}$$

$where\ \boldsymbol{\mu} = m * \left\lfloor \frac{q}{t} \right\rceil for\ \boldsymbol{m} \in R_t, t \ll q$

Cryptographic Challenges



(**R**ing-)**L**earning **W**ith **E**rrors: It is hard to find s given c and a

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$
$R = \mathbb{Z}[X]/(X^n + 1)$

Encryption

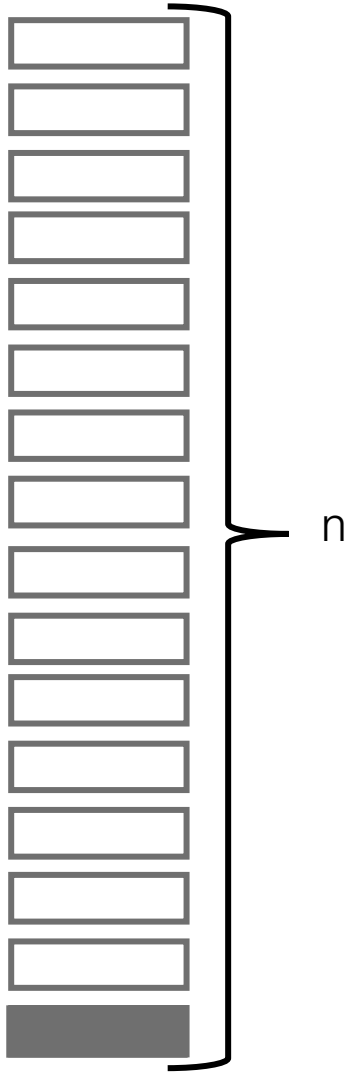
$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{\mu} + \boldsymbol{e}$$

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

Homomorphic Addition

- Noise increases linearly

Cryptographic Challenges



(**R**ing-)**L**earning **W**ith **E**rrors: It is hard to find s given c and a

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$

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

Encryption

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{\mu} + \boldsymbol{e}$$

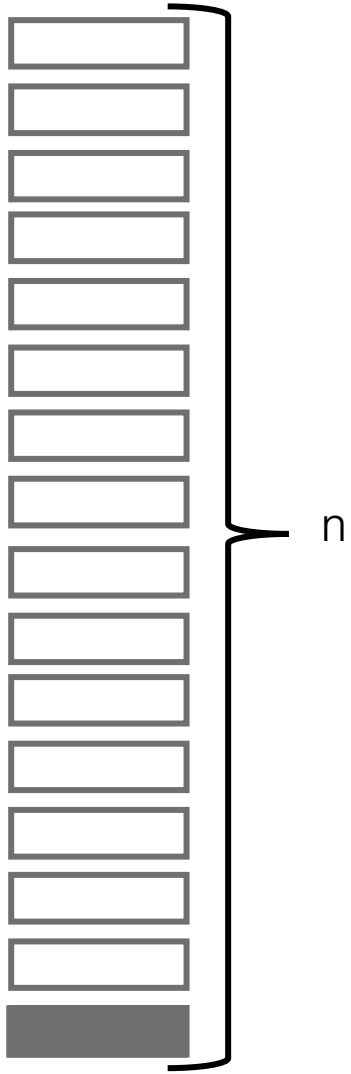
$where\ \boldsymbol{\mu} = m * \left\lfloor \frac{q}{t} \right\rfloor for\ \boldsymbol{m} \in R_t, t \ll q$

Homomorphic Addition / Multiplication

- Noise increases linearly / quadratically

Cryptographic Challenges

ETH zürich



(**R**ing-)**L**earning **W**ith **E**rrors: It is hard to find s given c and a

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$
$R = \mathbb{Z}[X]/(X^n + 1)$

Encryption

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{\mu} + \boldsymbol{e}$$

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

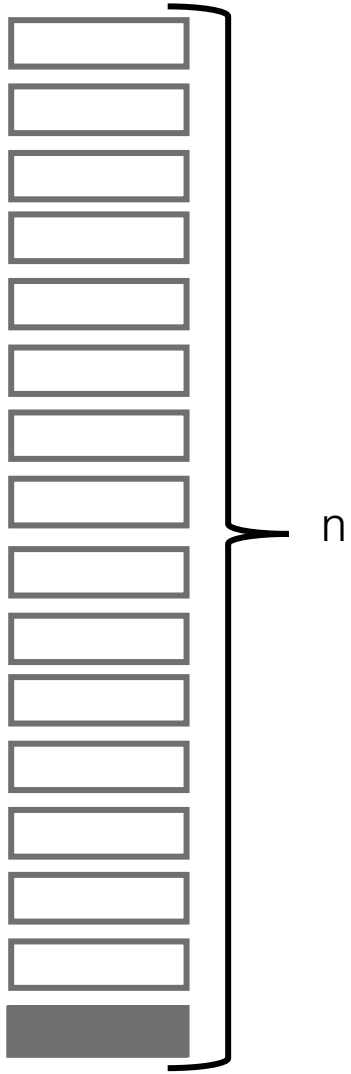
Homomorphic Addition / Multiplication

- Noise increases linearly / quadratically

If noise grows too large, decryption fails

- Bootstrapping reduces noise homomorphically

Cryptographic Challenges



(**R**ing-)**L**earning **W**ith **E**rrors: It is hard to find s given c and a

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$
$R = \mathbb{Z}[X]/(X^n + 1)$

Encryption

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{\mu} + \boldsymbol{e}$$

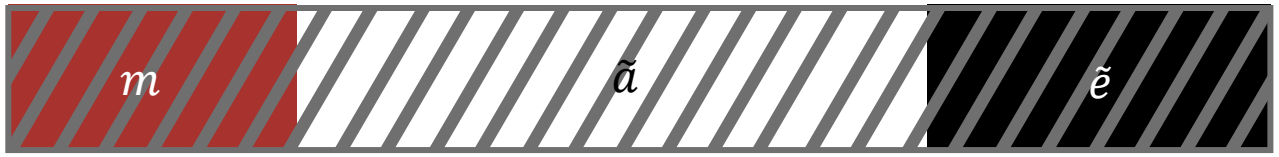
$where\ \boldsymbol{\mu} = m * \left\lfloor \frac{q}{t} \right\rfloor for\ \boldsymbol{m} \in R_t, t \ll q$

Homomorphic Addition / Multiplication

- Noise increases linearly / quadratically

If noise grows too large, decryption fails

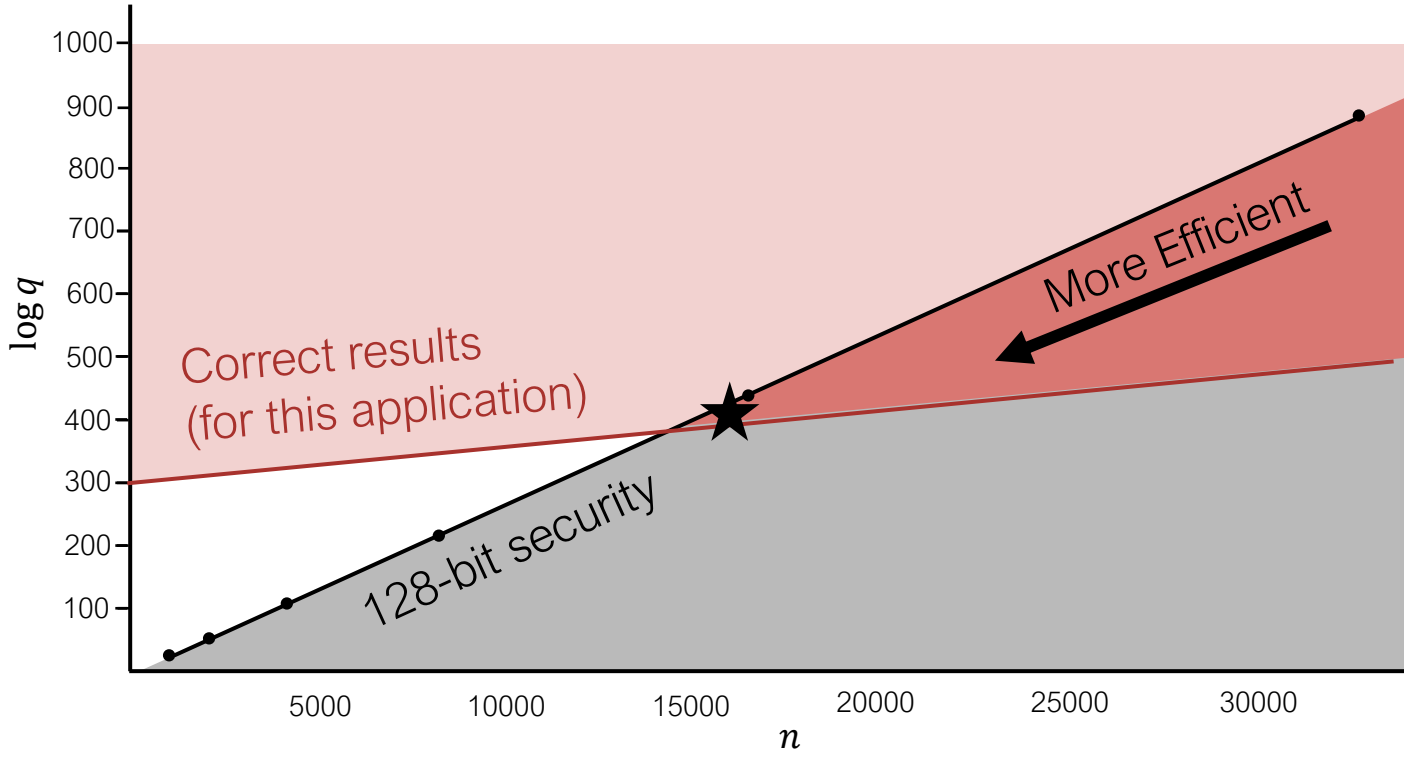
- Bootstrapping reduces noise homomorphically



Cryptographic Challenges

ETH zürich

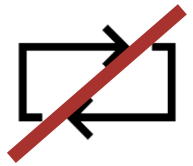
- Parameter Selection
  - Based on current knowledge about Learning With Error (LWE) hardness
  - Careful trade-off between efficiency, security and correctness



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

Engineering Challenges

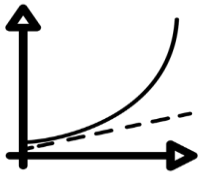
No If/Else



No Loops

No Jumps

Optimizations

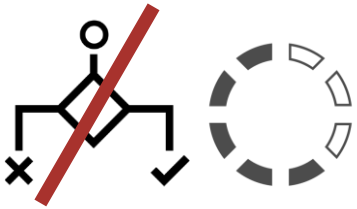


Approximations

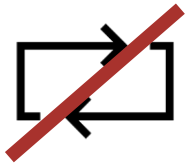
SIMD Batching

FHE Programming Paradigm

ETH zürich

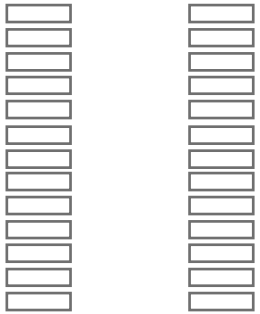


No If/Else



No Loops

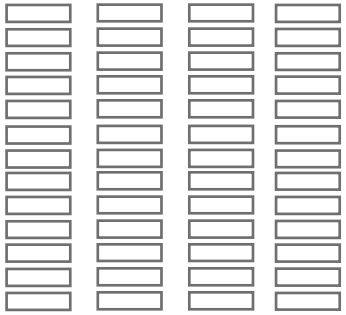
SIMD Batching



Enc(a) Enc(b)

Standard C++

```
int foo(int a, int b) {

  if(a < b) {
    return a * b;
  } else {
    return a + b;
  }

}
```



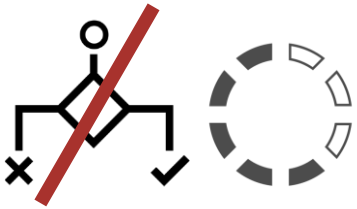
Enc(a_3) Enc(a_1)
Enc(a_2) Enc(a_0)

FHE

```
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

ETH zürich



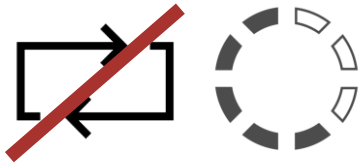
No If/Else

Standard C++

```
int foo(int a, int b) {

   if(a < b) {
      return a * b;
   } else {
      return a + b;
   }

}
```

FHE

```
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++

```
int foo(int a, int b){

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

}
```

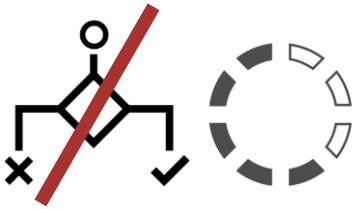
FHE

```
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;
   s = less(a,b) *
        (s + 2*b) +
        (1-c) * s;
```

SIMD Batching

FHE Programming Paradigm

ETH zürich



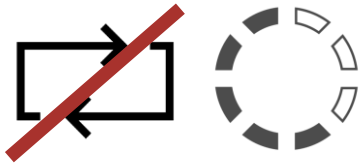
No If/Else

Standard C++

```
int foo(int a, int b) {

   if(a < b) {
      return a * b;
   } else {
      return a + b;
   }

}
```

FHE

```
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++

```
int foo(int a, int b){

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

}
```

FHE

```
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;
   s = less(a,b) *
        (s + 2*b) +
        (1-c) * s;
```

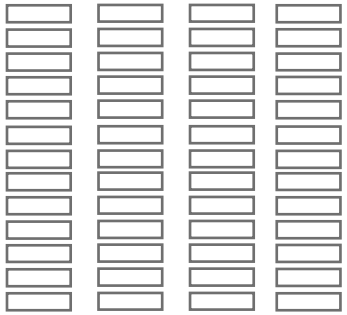
SIMD Batching

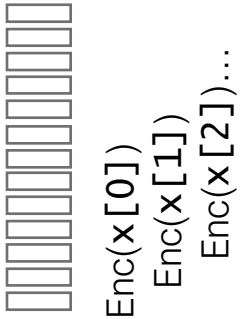
Standard C++

```
int foo(int[] x,int[] y){

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

}
```

FHE

```
int foo(int[] x,int[] y){

   int[] r;
   r[0] = x[0] * y[0];
   r[1] = x[1] * y[1];
   r[2] = x[2] * y[2];
   r[3] = x[3] * y[3];
   r[4] = x[4] * y[4];
   r[5] = x[5] * y[5];
   return r;
```



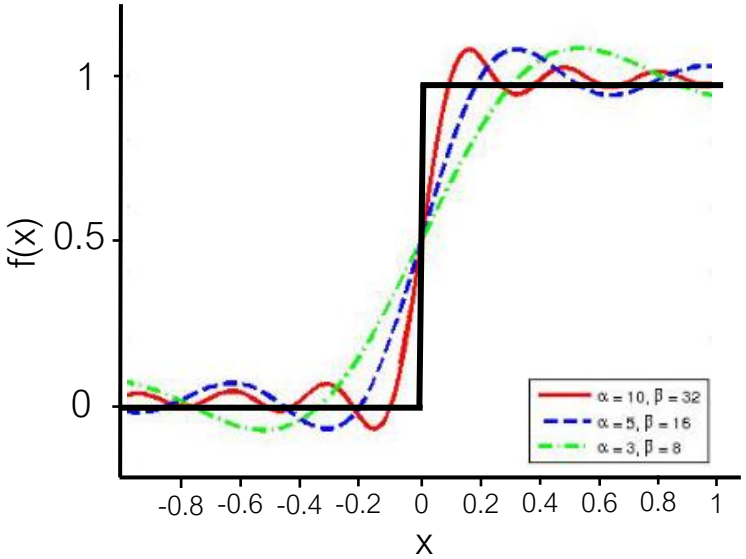


Approximation

ETH zürich

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

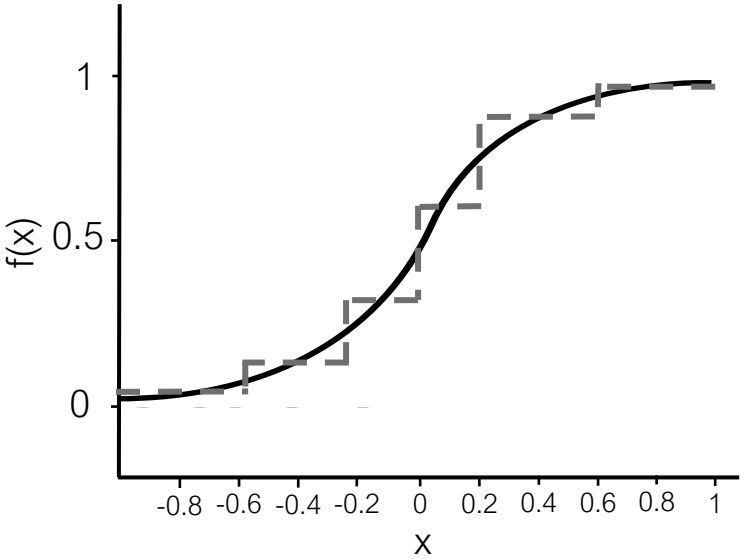
- Polynomial Approximation



x	$f(x)$
-1	0
-0.6	0
-0.2	0
0	1
0.2	1
0.6	1
1	1

- Frequently requires too many chained multiplications

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



x	$\tilde{f}(x)$
-1	0.05
-0.6	0.15
-0.2	0.33
0	0.66
0.2	0.85
0.6	0.95
1	1

- Limited to univariate functions (single ctxt as input)

Challenges of Developing FHE Applications

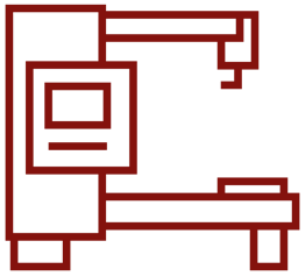
ETH zürich

Libraries

- Underlying Math ✓
- Parameter Selection ✗
- Security ✓

Compilers

- No If/Else ✗
- No Loops ✗
- No Jumps ✗
- Optimizations ✓
- Approximations ✗
- SIMD Batching ✓



FHE Compilers

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

Input Language

ETH zürich

SEAL (library)

```
// First Dense Layer
DenseLayer d1(32, input_size);
seal::Ciphertext result;
mvp(*galoisKeys,*evaluator, *encoder, d1.units(), d1.input_size(),
        d1.weights_as_diags(), image_ctxt, result);
seal::Plaintext b1;
encoder->encode(d1.bias(), result.parms_id(), result.scale(), b1);
evaluator->add_plain_inplace(result, b1);
evaluator->rescale_to_next_inplace(result);

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

// Duplication of certain values necessary for the SIMD layout
seal::Plaintext mask; encoder->encode(vec(1, d1.units()),
                                      result.parms_id(),
                                      result.scale(), mask);
evaluator->multiply_plain_inplace(result, mask);
seal::Ciphertext tmp;
evaluator->rotate_vector(result, d1.units(), *galoisKeys, tmp);
evaluator->add_inplace(tmp, result);
evaluator->rescale_to_next_inplace(tmp);

//  Second Dense Layer
DenseLayer d2(16, d1.units());
mvp(*galoisKeys, *evaluator, *encoder, d2.units(), d2.input_size(),
        d2.weights_as_diags(), tmp, result);
seal::Plaintext b2; encoder->encode(d2.bias(), result.parms_id(),
                                    result.scale(), b2);
evaluator->add_plain_inplace(result, b2);
evaluator->rescale_to_next_inplace(result);

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

EVA

```
def diag(matrix, d):
    m, n = matrix.shape
    r = [0] * n
    for k in range(n):
        r[k] = matrix[k % m][(k + d) % n]
    return r

def mvp(ptxt_matrix, enc_vector):
    m, n = ptxt_matrix.shape
    log2_n_div_m = math.ceil(math.log(n // m, 2)) t = 0
    for i in range(m):
        t += (enc_vector << i) * diag(ptxt_matrix, i)

    for i in range(log2_n_div_m):
        offset = n // (2 << i) t += t << offset
    return t

def compile():
    mlp = EvaProgram('NN (MLP)', vec_size=32 * 32)
    with mlp:
        image = Input('input_0')
        d1 = mvp(weights_1, image)
        d1 = d1 + bias_1.tolist()

        act1 = d1 * d1
        d2 = mvp(weights_2, act1)
        d2 = d2 + bias_2.tolist()
        act2 = d2 * d2

        Output('output', act2)
        Output('output', d1)
```

nGraph-HE

```
import numpy as np
import tensorflow as tf
from tf.keras.layers import Input, 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")(y)
    known_shape = y.get_shape()[1:] size =
np.prod(known_shape)
    return y
```

SEALion

```
from sealion.heras.models import Sequential
from sealion.heras.layers import Dense, Activation

model = Sequential()
model.add(Dense(units=30, input_dim=784,
                input_saturation=4, saturation=2 ** 4))
model.add(Activation())
model.add(Dense(units=10, saturation=2 ** 4))

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

Input Language

ETH zürich

SEAL (library)

```
// First Dense Layer
DenseLayer d1(32, input_size);
seal::Ciphertext result;
mvp(*galoisKeys,*evaluator, *encoder, d1.units(), d1.input_size(),
        d1.weights_as_diags(), image_ctxt, result);
seal::Plaintext b1;
encoder->encode(d1.bias(), result.parms_id(), result.scale(), b1);
evaluator->add_plain_inplace(result, b1);
evaluator->rescale_to_next_inplace(result);

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

// Duplication of certain values necessary for the SIMD layout
seal::Plaintext mask; encoder->encode(vec(1, d1.units()),
                                      result.parms_id(),
                                      result.scale(), mask);
evaluator->multiply_plain_inplace(result, mask);
seal::Ciphertext tmp;
evaluator->rotate_vector(result, d1.units(), *galoisKeys, tmp);
evaluator->add_inplace(tmp, result);
evaluator->rescale_to_next_inplace(tmp);

//  Second Dense Layer
DenseLayer d2(16, d1.units());
mvp(*galoisKeys, *evaluator, *encoder, d2.units(), d2.input_size(),
        d2.weights_as_diags(), tmp, result);
seal::Plaintext b2; encoder->encode(d2.bias(), result.parms_id(),
                                    result.scale(), b2);
evaluator->add_plain_inplace(result, b2);
evaluator->rescale_to_next_inplace(result);

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

EVA

```
def diag(matrix, d):
    m, n = matrix.shape
    r = [0] * n
    for k in range(n):
        r[k] = matrix[k % m][(k + d) % n]
    return r

def mvp(ptxt_matrix, enc_vector):
    m, n = ptxt_matrix.shape
    log2_n_div_m = math.ceil(math.log(n // m, 2)) t = 0
    for i in range(m):
        t += (enc_vector << i) * diag(ptxt_matrix, i)

    for i in range(log2_n_div_m):
        offset = n // (2 << i) t += t << offset
    return t

def compile():
    mlp = EvaProgram('NN (MLP)', vec_size=32 * 32)
    with mlp:
        image = Input('input_0')
        d1 = mvp(weights_1, image)
        d1 = d1 + bias_1.tolist()

        act1 = d1 * d1
        d2 = mvp(weights_2, act1)
        d2 = d2 + bias_2.tolist()
        act2 = d2 * d2

        Output('output', act2)
        Output('output', d1)
```

nGraph-HE

```
import numpy as np
import tensorflow as tf
from tf.keras.layers import Input, 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")(y)
    known_shape = y.get_shape()[1:] size =
np.prod(known_shape)
    return y
```

SEALion

```
from sealion.heras.models import Sequential
from sealion.heras.layers import Dense, Activation

model = Sequential()
model.add(Dense(units=30, input_dim=784,
                input_saturation=4, saturation=2 ** 4))
model.add(Activation())
model.add(Dense(units=10, saturation=2 ** 4))

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

Input Language

SEAL (library)

```cpp
// First Dense Layer
DenseLayer d1(32, input_size);
seal::Ciphertext result;
mvp(*galoisKeys,*evaluator, *encoder, d1.units(), d1.input_size(),
        d1.weights_as_diags(), image_ctxt, result);
seal::Plaintext b1;
encoder->encode(d1.bias(), result.parms_id(), result.scale(), b1);
evaluator->add_plain_inplace(result, b1);
evaluator->rescale_to_next_inplace(result);

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

// Duplication of certain values necessary for the SIMD layout
seal::Plaintext mask; encoder->encode(vec(1, d1.units()),
                                      result.parms_id(),
                                      result.scale(), mask);
evaluator->multiply_plain_inplace(result, mask);
seal::Ciphertext tmp;
evaluator->rotate_vector(result, d1.units(), *galoisKeys, tmp);
evaluator->add_inplace(tmp, result);
evaluator->rescale_to_next_inplace(tmp);

//  Second Dense Layer
DenseLayer d2(16, d1.units());
mvp(*galoisKeys, *evaluator, *encoder, d2.units(), d2.input_size(),
        d2.weights_as_diags(), tmp, result);
seal::Plaintext b2; encoder->encode(d2.bias(), result.parms_id(),
                                    result.scale(), b2);
evaluator->add_plain_inplace(result, b2);
evaluator->rescale_to_next_inplace(result);

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

```python
def diag(matri
    m, n = mat
    r = [0] * n
    for k in ran
        r[k] = m
    return r

def mvp(ptxt_
    m, n = ptxt
    log2_n_div
    for i in ran
        t += (en

    for i in ran
        offset =
    return t

def compile():
    mlp = EvaF
    with mlp:
        image =
        d1 = mv
        d1 = d1

        act1 = c
        d2 = mv
        d2 = d2
        act2 = c

        Output(
        Output(
```

```cpp
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 &ctv, 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!");
    if (n!=diagonals[0].size() || n==0)
        throw invalid_argument( "Diagonals must have non-zero dimension that matches n");

    size_t n_div_m = n/m;
    size_t log2_n_div_m = ceil(log2(n_div_m));
    if (m*n_div_m!=n || (2ULL << (log2_n_div_m - 1)!=n_div_m && n_div_m!=1))
        throw invalid_argument( "Matrix dimension m must divide n and the result must be power of two");

    Ciphertext ctxt_t;
    for (size_t i = 0; i < m; ++i) {
        // rotated_v = rot(v,i)
        Ciphertext ctxt_rotated_v = ctv;
        if ( i != 0)
            evaluator.rotate_vector_inplace(ctxt_rotated_v, i, galois_keys);

        Plaintext ptxt_current_diagonal;
        encoder.encode(diagonals[i], ctxt_rotated_v.parms_id(), ctxt_rotated_v.scale(), ptxt_current_diagonal);
        Ciphertext ctxt_tmp;
        evaluator.multiply_plain(ctxt_rotated_v, ptxt_current_diagonal, ctxt_tmp);

        if (i==0)
            ctxt_t = ctxt_tmp;
        else
            evaluator.add_inplace(ctxt_t, ctxt_tmp);

        Ciphertext ctxt_r = std::move(ctxt_t);

        for (int i = 0; i < log2_n_div_m; ++i) {
            Ciphertext ctxt_rotated_r = ctxt_r
            size_t offset = n/(2ULL << i);
            evaluator.rotate_vector_inplace(ctxt_rotated_r, offset, galois_keys);
            evaluator.add_inplace(ctxt_r, ctxt_rotated_r);
        }
    enc_result = std::move(ctxt_r);
}
```

Input Language

ETH zürich

SEAL (library)

```
// First Dense Layer
DenseLayer d1(32, input_size);
seal::Ciphertext result;
mvp(*galoisKeys,*evaluator, *enc
        d1.weights_as_diags(), ima
seal::Plaintext b1;
encoder->encode(d1.bias(), result
evaluator->add_plain_inplace(result, b1);
evaluator->rescale_to_next_inplace(result);

// Activation, x -> x^2
evaluator->square_inplace(result)
evaluator->relinearize_inplace(res
evaluator->rescale_to_next_inplac

// Duplication of certain values nec
seal::Plaintext mask; encoder->en
                                result.parms_id(),
                                result.scale(), mask);
evaluator->multiply_plain_inplace(result, mask);
seal::Ciphertext tmp;
evaluator->rotate_vector(result, d1
evaluator->add_inplace(tmp, resu
evaluator->rescale_to_next_inplac

//  Second Dense Layer
DenseLayer d2(16, d1.units());
mvp(*galoisKeys, *evaluator, *enc
        d2.weights_as_diags(), tmp, result);
seal::Plaintext b2; encoder->encode(d2.bias(), result.parms_id(),
                                    result.scale(), b2);
evaluator->add_plain_inplace(result, b2);
evaluator->rescale_to_next_inplace(result);

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

EVA

```
def diag(matrix, d):

def mvp(ptxt_matrix, enc_vector):
    m, n = ptxt_matrix.shape
    log2_n_div_m = math.ceil(math.log(n // m, 2)) t = 0
    for i in range(m):

def compile():
    mlp = EvaProgram('NN (MLP)', vec_size=32 * 32)
    with mlp:
        image = Input('input_0')

        act2 = d2 * d2

        Output('output', act2)
        Output('output', d1)
```

nGraph-HE

```
import numpy as np
                                    vation

        known_shape = input.get_shape()[1:]

    size = np.prod(known_shape) print('size', size)
    y = tf.reshape(input, [-1, size])
                                    use_bias=True)(y)

                                    ="output")(y)
```

SEALion

```
                                    vation

                                    on=2 ** 4))
model.add(Activation())
model.add(Dense(units=10, saturation=2 ** 4))

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

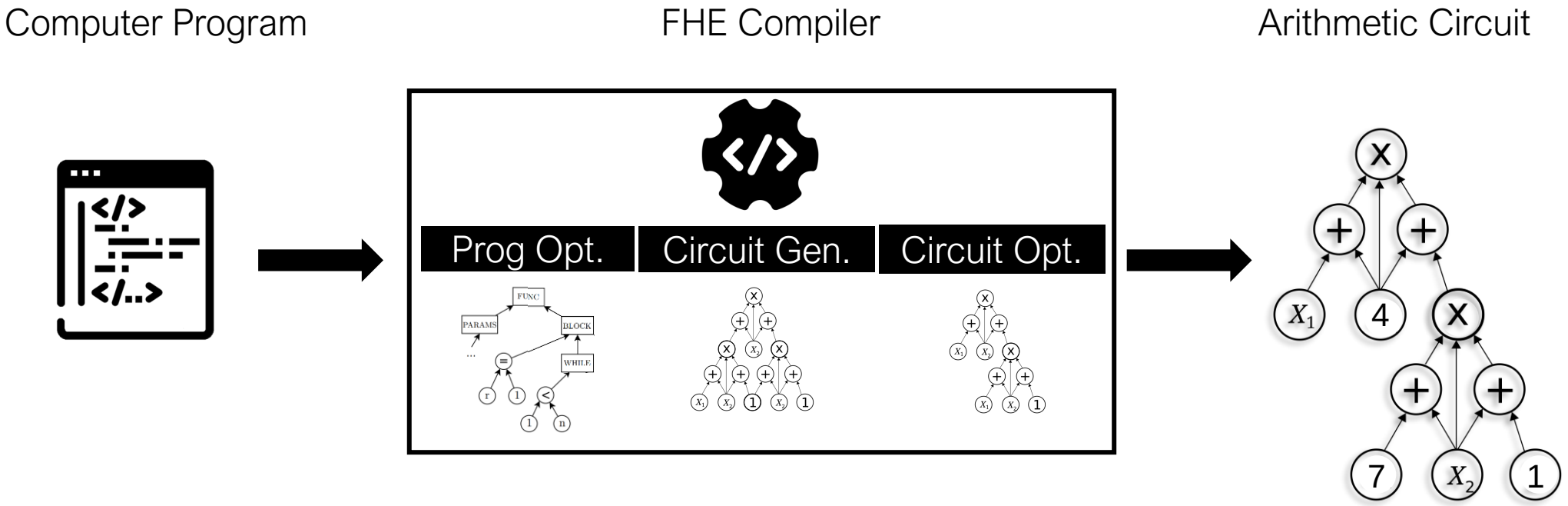
Disconnect between low- and high-level languages

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

ETH zürich

- Where do compilers apply optimizations?

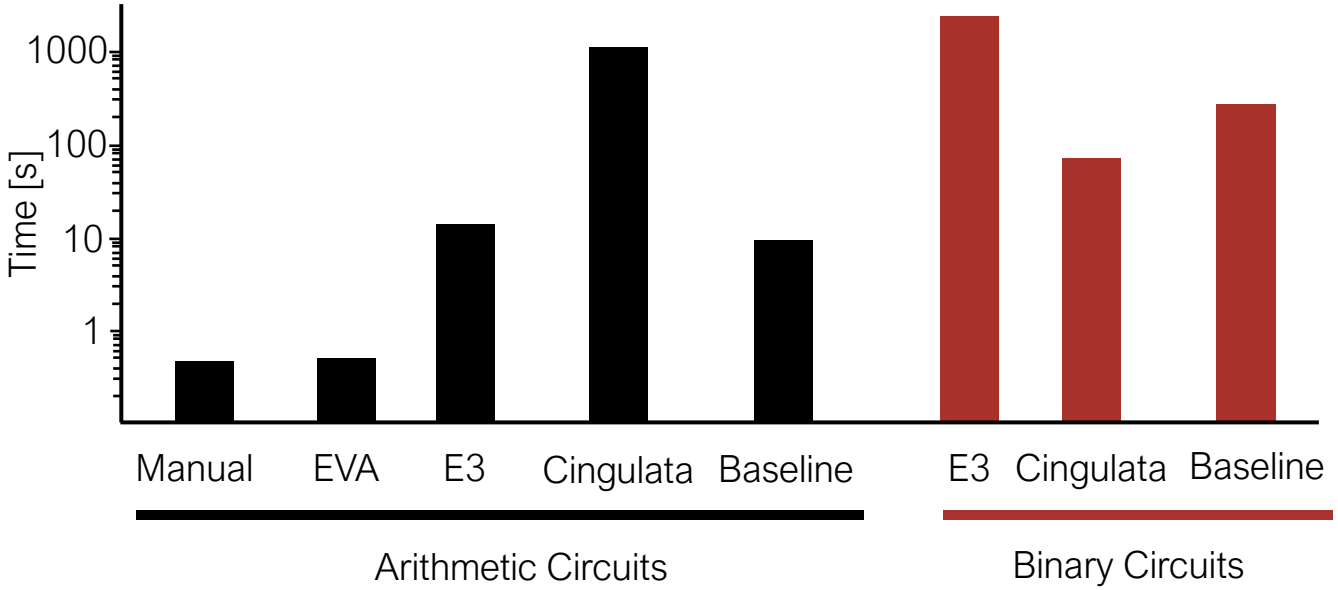
	Input	Prog Opt	Circuit Gen	Circuit Opt
ALCHEMY		✓		
Cingulata				✓
E^3			◌	
EVA				✓
Marble			◌	
Ramparts		✓		
CHET			✓	
nGraph-HE			✓	
SEALion	K		✓	

Evaluation: χ^2 Test

ETH zürich

- 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

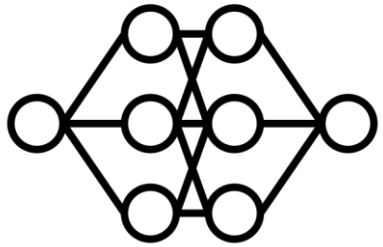
$$\alpha = (4N_0 N_2 - N_1^2)^2$$
$$\beta_1 = 2(2N_0+N_1)^2$$
$$\beta_2 = (2N_0+N_1)(2N_2+N_1)$$
$$\beta_3 = 2(2N_2+N_1)^2$$



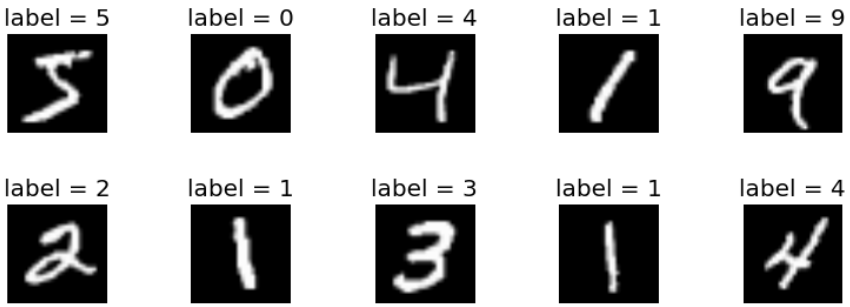
[LLN14] Lauter, K. et al. 2014. Private Computation on Encrypted Genomic Data. Progress in Cryptology - LATINCRYPT 2014, 3–27.

Evaluation: Machine Learning Inference

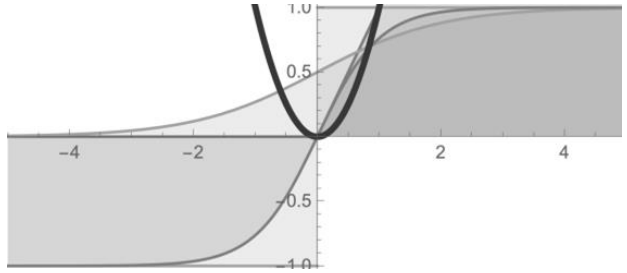
ETH zürich

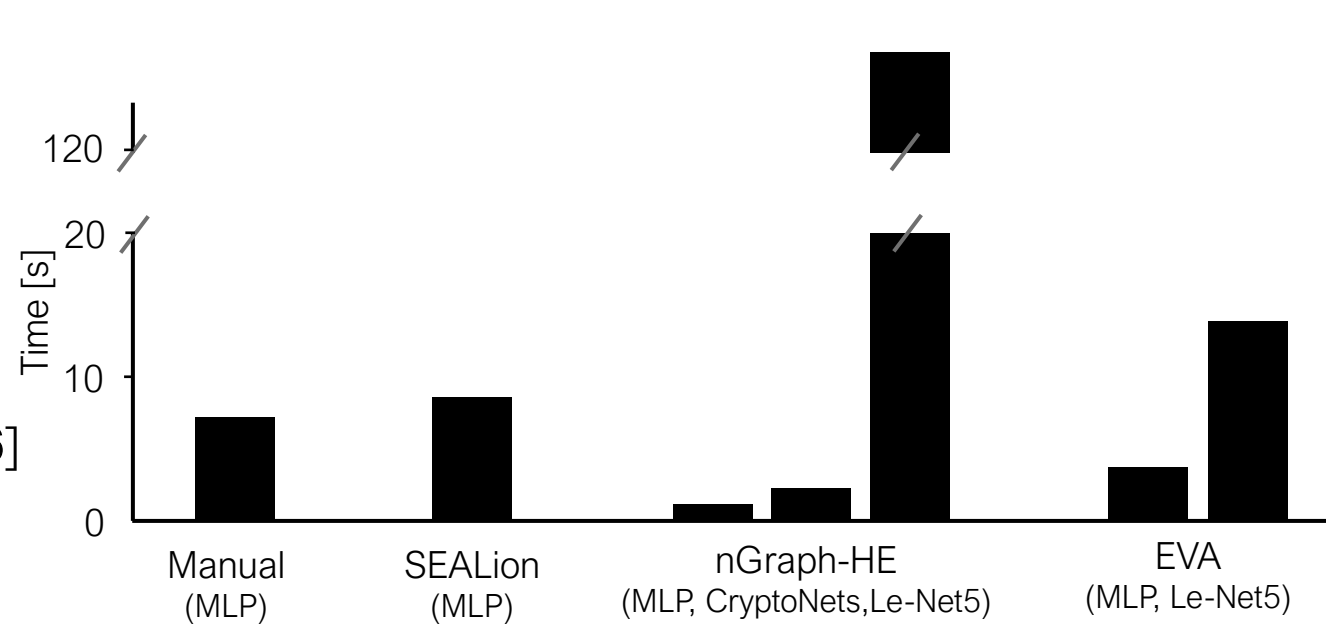


- MNIST – handwritten digit recognition
  - Simple task used as a reference benchmark in ML
  - Given an image of a digit, recognize a number in 0-9



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





[GDK+16] Gilad-Bacharach, R. et al. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. ICML 2016, 201–210.

Current State of the Art

ETH zürich

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

ETH zürich

E2E	Toolchains must address all aspects of FHE development
Automate	Translation between imperative paradigm and FHE
Optimize	Identify & exploit opportunities for SIMD parallelism


SoK: Fully Homomorphic Encryption Compilers

Alexander Viand, ETH Zürich, alexander.viand@inf.ethz.ch
Patrick Jattke, ETH Zürich, pjattke@ethz.ch
Anwar Hithnawi, ETH Zürich, anwar.hithnawi@inf.ethz.ch

GitHub

github.com/MarbleHE/SoK

docker

To appear at IEEE S&P 2021 preprint: arxiv.org/abs/2101.07078