Moving toward real-world deployment of FHE

Cathie Yun
Cryptography Engineer
cathieyun@ {gmail, github, medium, ... }

NIST Crypto Reading Club
December 14, 2022
Meet the Team

Shruthi Gorontala
Tech Lead
gshruthi@

Jeremy Kun
Cryptography Engineer
jkun@

Cathie Yun
Cryptography Engineer
cathieyun@
AGENDA

FHE Background

FHE Transpiler

FHE Hierarchy of Needs
1.0
Why should we care about Fully Homomorphic Encryption (FHE)?
Modern Cryptography

**Ubiquitous Adoption**
Conventional Crypto
- Encryption & Digital Signatures

**End-to-End Security**
- data in transit
  - secure communication
- data at rest
  - secure storage
- data in use
  - secure computation

**Just Starting**
Advanced Crypto
- Homomorphic Encryption
- Secure Multi-party Computation
- Zero Knowledge Proofs
Without FHE

With FHE
Fully Homomorphic Encryption

Enables **computation** on encrypted data

Delegate the **processing** of data without giving away **access** to it
1.1

FHE: A Brief History
40 Years of FHE History

Reference: Alexander Viand
Problem

Learning with Errors (LWE)
FHE: A Brief History

Learning With Errors

Gentry '09

Reference: Alexander Viand
Solving systems of linear equations

Linear system problem: given blue and green, find red

\[
\begin{bmatrix}
3 & 3 & 11 & 9 \\
6 & 5 & 7 & 5 \\
4 & 8 & 1 & 11 \\
2 & 6 & 2 & 2 \\
11 & 2 & 3 & 4 \\
5 & 8 & 12 & 2 \\
3 & 3 & 5 & 0 \\
\end{bmatrix}
\times
\begin{bmatrix}
4 \\
8 \\
1 \\
10 \\
4 \\
12 \\
9 \\
\end{bmatrix}
= 
\begin{bmatrix}
? \\
? \\
? \\
? \\
? \\
? \\
\end{bmatrix}
\]
Learning with Errors (LWE) problem

(Search) LWE problem: given blue and green, find red (or yellow)

$$\mathbb{Z}_{13}^{7\times 4} \times \mathbb{Z}_{13}^{4\times 1} + \mathbb{Z}_{13}^{7\times 1} = \mathbb{Z}_{13}^{7\times 1}$$
LWE Encryption
Post Quantum Cryptography
Encrypting with LWE

\[ \mathbb{Z}_{13}^{7\times 4} \]

\[ \begin{array}{cccc}
3 & 3 & 11 & 9 \\
6 & 5 & 7 & 5 \\
4 & 8 & 1 & 11 \\
2 & 6 & 2 & 2 \\
11 & 2 & 3 & 4 \\
5 & 8 & 12 & 2 \\
3 & 3 & 5 & 0 \\
\end{array} \]

\[ \times \]

\[ \mathbb{Z}_{13}^{4\times 1} \]

\[ \begin{array}{c}
1 \\
0 \\
1 \\
-1 \\
0 \\
1 \\
1 \\
\end{array} \]

\[ \mathbb{Z}_{13}^{7\times 1} \]

\[ \begin{array}{c}
0 \\
0 \\
1 \\
1 \\
0 \\
1 \\
0 \\
\end{array} \]

\[ \mathbb{Z}_{13}^{7\times 1} \]

\[ \begin{array}{c}
4 \\
8 \\
2 \\
11 \\
5 \\
12 \\
9 \\
\end{array} \]

\[ \text{Secret} \]

\[ \text{Small noise} \]

\[ \text{Message} \]

\[ \text{Encryption} \]

\[ = \]
Encrypting with LWE

<table>
<thead>
<tr>
<th>Toy world</th>
<th>Real world</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{Z}_{13}^{7 \times 4}$</td>
<td>$\mathbb{Z}_{2^{15}}^{2430 \times 750}$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3673</td>
<td>12393</td>
</tr>
<tr>
<td>5484</td>
<td>599</td>
</tr>
<tr>
<td>13455</td>
<td>9830</td>
</tr>
<tr>
<td>19321</td>
<td>3948</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Rule of thumb (bit encryption):** dimension more than 750, modulus 16+ bits.

**Rule of thumb (homomorphic encryption):** dimension >2000, modulus >60 bits.
FHE Computations and Noise

Decryption fails if error grows too large
Bootstrapping

Homomorphically evaluates the decryption circuit to reduce noise
FHE: A Brief History

\[ \text{Enc}(0) \oplus \text{Enc}(1) \]

**Ring-Learning With Errors**

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

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

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

Gentry '09

BGV '12, B/FV '12
<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>1</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>4</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td>-10</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-11</td>
<td>-10</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>-9</td>
<td>12</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>-9</td>
<td>12</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td>-3</td>
<td>-9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td>-1</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Secret} \times \begin{bmatrix}
4 & 8 & 1 & 0 \\
10 & 1 & 0 & 8 \\
-9 & 4 & 12 & 0 \\
8 & 1 & 11 & 8
\end{bmatrix}
\]

\[
= \begin{bmatrix}
4 & 8 & 1 & 0 \\
1 & 10 & 0 & 8 \\
9 & 12 & 0 & 8 \\
11 & 11 & 0 & 8
\end{bmatrix}
\]

Small noise

Encryption
FHE: A Brief History

**Enc(0) ⊗ Enc(1)**

**Ring-Learning With Errors**

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

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

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

Gentry '09  
BGV '12, B/FV '12

Fast Binary FHE

~1 OoM more operations but each operation is incredibly fast

TFHE '16

CKKS '16

Approximate FHE

\[ \text{Dec}(\text{Enc}(m)) \approx m \]
FHE Schemes

Vanilla LWE+
\[ b = a.s + e + \left\{ \frac{q}{2} \text{ if } m = 1, 0, \text{ if } m = 0 \right\} \]

BGV
\[ b = a.s + te + m, \text{ } m \text{ is in } \mathbb{Z}_t \]

BFV
\[ b = a.s + e + \left[ \frac{q}{t} \right]m, \text{ } m \text{ is scaled} \]

CKKS
\[ B = a.s + e + \Delta \cdot \text{Inv}F(m) \text{ - approximations} \]
FHE: A Brief History

Gentry '09
BGV '12, B/FV '12
Ring-Learning With Errors
$c = a \cdot s + e$
where $a \sim_R q, e \sim_R q, s \in_R q$
$R = \mathbb{Z}[X]/(X^n + 1)$

Fast Binary FHE
TFHE '16

CKKS '16
Approximate FHE
$\text{Dec} (\text{Enc}(m)) \approx m$

Look-Up Tables

<table>
<thead>
<tr>
<th>X</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-0.84</td>
</tr>
<tr>
<td>-0.5</td>
<td>-0.47</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>0.47</td>
</tr>
<tr>
<td>1</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Hardware Acceleration

Fact that we still see state of the art solutions with 10 year old schemes hints at the fact that crypto innovation in terms of performance is plateauing.

Instead, we'll either see HW acceleration or tailoring schemes more closely to application needs bringing us closer to 1-10x regime of overhead compared to "non-FHE" computation.
Why aren't we already using FHE for everything?
Challenges

- Speed
- Ciphertext Expansion
- Ease of Use
Challenges

Speed
Ciphertext Expansion
Ease of Use
FHE Programming

Need to abstract away cryptographic details from engineers

Reference: SoK: Fully Homomorphic Encryption Compilers, Viand et al
FHE challenges: Cryptographic

01 Circuit depth analysis
02 Noise tracking
03 Bootstrapping
04 Parameter selection

Reference: SoK: Fully Homomorphic Encryption Compilers, Viand et al
FHE challenges: Engineering

01
No if/else (well... sort of)

02
Loops need static upper bounds

03
No branch and bound optimizations

04
Using the right scheme for the right type of computation

Reference: SoK: Fully Homomorphic Encryption Compilers, Viand et al
Adding FHE to an existing application

YOU MUST UNLEARN

WHAT YOU HAVE LEARNED.
2.1

FHE Transpiler: Reusing hardware tooling
FHE transpiler

Allows us to easily convert existing code that works on plaintext to work on ciphertext.
FHE transpiler
Convert C++ code that works on plaintext to work on ciphertext.

```cpp
#include <tfhe.h>

void sum(LweSample* result, const LweSample* a, const LweSample* b,
const int nb_bits, const TFheKeySet* bk) {
    LweSample* carry = new_ciphertext(bk->params);
    LweSample* temp = new_ciphertext(bk->params);
    // Initialize the carry to 0
    bootsCONSTANT(&carry, 0, bk);
    // Compute bitwise addition
    for (int i = 0; i < nb_bits; i++) {
        // Compute Sum
        bootsXOR(&temp, &a[i], &b[i], bk);
        bootsXOR(&result[i], &temp, &carry, bk);
        // Compute carry
        bootsAND(&carry, &carry, &temp, bk);
        bootsAND(&temp, &a[i], &b[i], bk);
        bootsOR(&carry, &temp, &carry, bk);
    }
    delete_ciphertext(carry);
    delete_ciphertext(temp);
}

int sum(int a, int b) {
    return (a + b);
}
```
Transpiler Architecture

C++

Parser

[IR] Circuit

FHE Optimizer

[IR] FHE Optimized Circuit

Backend: Execution Engine

FHE Cryptosystem API

Metadata

FHE Circuit Scheduler

FHE* - API for Encryption/Decryption
The following code capitalizes words in a string and is written using standard C++ syntax.

```cpp
#include "string_cap.h"

State::State() :
    last_was_space_(true) {
}

unsigned char State::process(unsigned char c) {
    unsigned char ret = c;

    if(last_was_space_ && (c >= 'a') && (c <= 'z'))
        ret -= ('a' - 'A');

    last_was_space_ = (c == ' ');
    return ret;
}

// Note: Way to mark State::process() as main function
char my_package(State &st, char c) {
    return st.process(c);
}
```
package my_package
fn _ZN5State7processEh(this: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  literal.20: bits[8] = literal(value=97, pos=1,10,2)
  literal.31: bits[8] = literal(value=97, pos=1,11,3)
  ...
  ret tuple.44: (bits[8], (bits[1])) = tuple(sel.36, tuple.43, pos=1,7,1)
}
fn my_package(st: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  invoke.45: (bits[8], (bits[1])) = invoke(st, c, to_apply=_ZN5State7processEh, pos=1,19,2)
  tuple_index.46: bits[8] = tuple_index(invoke.45, index=0, pos=1,19,2)
  tuple_index.47: (bits[1]) = tuple_index(invoke.45, index=1, pos=1,19,2)
  ret tuple.48: (bits[8], (bits[1])) = tuple(tuple_index.46, tuple_index.47, pos=1,18,1)
}
plaintext(28): do or do not there is no try
Encryption done
Sanity check by decryption: do or do not there is no try

Server side computation:
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
done.
Total time: 28 secs
Computation done

Decrypted result: Do Or Do Not There Is No Try
Decryption done
Execution engine: Multi-threaded Interpreter (96 cores)

plaintext size: 32
plaintext: 'do or do not there is no try'
Encryption done
Initial state check by decryption:
do or do not there is no try

Starting!

Server side computation:
Total time: \(0.756137\) secs
CPU time: \(46.4172\) secs
Computation done

Decrypted result: Do Or Do Not There Is No Try
Decryption done
Open Source

Fully Homomorphic Encryption (FHE)

This repository contains open-source libraries and tools to perform fully homomorphic encryption (FHE) operations on an encrypted data set.

About Fully Homomorphic Encryption
3.0

Making FHE usable
Maslow’s Hierarchy of Needs

1. **Physiological needs**
   - Air, water, food, shelter, sleep, clothing, reproduction

2. **Safety needs**
   - Personal security, employment, resources, health, property

3. **Love and belonging**
   - Friendship, intimacy, family, sense of connection

4. **Esteem**
   - Respect, self-esteem, status, recognition, strength, freedom

5. **Self-actualization**
   - Desire to become the most that one can be
1. **FHE Instruction SET Architecture (ISA)**

- **Data Manipulation Ops**
  - ADD, MUL, Gates, LUT

- **Ciphertext Maintenance Ops**
  - Bootstrapping, Key-switching, Relinearization, Rescaling,
    Scheme Switching, Rotation

- **Modeling Cost of Micro Benchmarks**
  - Scheme, Noise budget, latency, dependencies?
FHE Libraries

- TFHE
- CONCRETE
- OpenFHE
- HELib
- FHEW
- SEAL
- HEEAN
- Lattigo
- CuFHE
- ...

45
2. Compilers / Transpilers

- **General Purpose Compilers**
  - Boolean circuit optimizations

- **Domain Specific Compilers**
  - Optimizers for arithmetic

- **Stitching Together**
  - Scheme switching

- **Cost for Macro Benchmarks**
  - Optimizers and hardware
FHE Compilers

- Cingulata
- E3
- FHE Transpiler
- CHET
- ALCHEMY
- RAMPARTS

- CONCRETE-ML
- EVA
- nGraph-HE
- SEALion
- Marble
- ....
3. FHE Application Development

- **Trade Offs**
  Performance tradeoff across hardware

- **Debuggability**
  Program verifiability with plaintext execution

- **Compiler Usability**
  Application composing multiple FHE circuits

- **Application Level Benchmarks**
  HEBench
4. Systems Integration / Privacy Engineering

Key Management
Private key, Public key, Multi-key

Trust Model Development
May need other Privacy Enhancing Technologies

Open Problem: PrivacyLeaks in Data Size
Privacy / Efficiency trade off

Open Problem: Ciphertext Expansion
Packing and hybrid homomorphic encryption
4. Privacy Engineering

Trust Model

- Adversarial Server
  Zero Knowledge Proofs
- Adversarial Client
  Differential privacy
- Multi-Party Aggregation with Honest-but-Curious
  Secure Multi Party Computation
4. Privacy Engineering

Privacy Leaks in Data Size

**Leak Size in Plaintext**
Minimum privacy

**Bucket Size**
S, M, L

**Open Problem: Probabilistic Sizing**
Quantifying privacy

**Max Size**
Maximum privacy
4. Systems Integration

Ciphertext Expansion

Does Hybrid Homomorphic Encryption solve everything?

What about ciphertext expansion of the result from server to client?
5. FHE Actualization

Latency / Network Bandwidth / Privacy
Metrics

$$\text{$$$ for Hardware}$$
Budget

Product Timelines
Launch dates

Software Developer Hours
Engineering time
FHE Hierarchy of Needs

1. FHE Instruction Set Architecture
2. Compilers / Transpilers
3. FHE application development
4. Systems integration / Privacy engineering
5. FHE actualization
Thanks for listening!

google/fully-homomorphic-encryption

Find us IRL at Cryptography / FHE conferences

fhe-open-source@google.com
cathieyun@{google, gmail}.com
Extra slides
FHE Transpiler Building Blocks
FHE Compilers

Parameter selection

Domain Specific Compilers
BGV, BFV, CKKS

General Purpose Compilers
FHEW, TFHE
General purpose FHE compilers

Cingulata
Armadillo: a compilation chain for privacy preserving applications
- Carpov et al

E3
A Framework for Compiling C++ Programs with Encrypted Operands
- Chielle et al
General purpose FHE compilers

Cingulata

Armadillo: a compilation chain for privacy preserving applications

- Carpov et al

Uses boolean circuit

E3

A Framework for Compiling C++ Programs with Encrypted Operands

- Chielle et al
FHE Programming Hardware Design
High-Level Synthesis (HLS) Tools

XLS: Accelerated HW Synthesis
Composable FHE

FHEW
FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second
- Ducas, Micciancio

CGGI / TFHE
Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds
- Chillot et al
void bootsNOT(LweSample* result, const LweSample* ca, const TFheGateBootstrappingCloudKeySet* bk);
void bootsNAND(LweSample* result, const LweSample* ca, const LweSample* cb, const TFheGateBootstrappingCloudKeySet* bk);
void bootsOR(LweSample* result, const LweSample* ca, const LweSample* cb, const TFheGateBootstrappingCloudKeySet* bk);
void bootsAND(LweSample* result, const LweSample* ca, const LweSample* cb, const TFheGateBootstrappingCloudKeySet* bk);
void bootsXOR(LweSample* result, const LweSample* ca, const LweSample* cb, const TFheGateBootstrappingCloudKeySet* bk);
void bootsNOR(LweSample* result, const LweSample* ca, const LweSample* cb, const TFheGateBootstrappingCloudKeySet* bk);
void bootsMUX(LweSample* result, const LweSample* a, const LweSample*b, const LweSample* c, const TFheGateBootstrappingCloudKeySet* bk);
FHE Transpiler Architecture
Transpiler Design

Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine
Transpiler Design

Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine
Transpiler Frontend: Parser

- C++
- Frontend: Parser
  - [IR] Circuit
  - Metadata
The following code capitalizes words in a string and is written using standard C++ syntax.

```cpp
#include "string_cap.h"
State::State()
  : last_was_space_(true) {}

unsigned char State::process(unsigned char c) {
  unsigned char ret = c;

  if(last_was_space_ && (c >= 'a') && (c <= 'z'))
    ret -= ('a' - 'A');

  last_was_space_ = (c == ' ');
  return ret;
}

// Note: Way to mark State::process() as main function
char my_package(State &st, char c) {
  return st.process(c);
}
```
package my_package

fn _ZN5State7processEh(this: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  literal.20: bits[8] = literal(value=97, pos=1,10,2)
  literal.31: bits[8] = literal(value=97, pos=1,11,3)
  ...
  ...
  ...
  ret tuple.44: (bits[8], (bits[1])) = tuple(sel.36, tuple.43, pos=1,7,1)
}

fn my_package(st: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  invoke.45: (bits[8], (bits[1])) = invoke(st, c, to_apply=_ZN5State7processEh, pos=1,19,2)
  tuple_index.46: bits[8] = tuple_index(invoke.45, index=0, pos=1,19,2)
  tuple_index.47: (bits[1]) = tuple_index(invoke.45, index=1, pos=1,19,2)
  ret tuple.48: (bits[8], (bits[1])) = tuple(tuple_index.46, tuple_index.47, pos=1,18,1)
}
Intermediate Representation
XLS-IR

Data types
Bits, Arrays, Tuples, Tokens

Data Manipulation operations
Gates, arithmetic.. Many more
Transpiler Design

Frontend: Parser
Middle-end: FHE Optimizer
Backend: Execution Engine
Transpiler Middle-end: FHE Optimizer

[IR] Circuit

Middle-end: FHE Optimizer

[IR] FHE Optimized Circuit
Lowered and optimized circuit (FHE scheme + encoding)
IR after running through the booleanifier.

Now it only contains AND, OR, and NOT gates.

**Booleanized XLS IR**

IR after running through the booleanifier.

Now it only contains AND, OR, and NOT gates.
Transpiler Design

Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine
Transpiler Backend: Execution Engine

- Metadata
- [IR] FHE Optimized Circuit
  Lowered and optimized circuit
  (FHE scheme + encoding)
- FHE Cryptosystem API
- FHE Circuit Scheduler
  Transpiler / Interpreter
- FHE* - Interface API for
  Encryption/Decryption
FHE C++

C++ that now works on ciphertext.

```cpp
#include <tfhe/tfhe.h>
#include <tfhe/tfhe_io.h>

#include <unordered_map>

void my_package_boolean(LweSample* result, LweSample* st, LweSample* c,
    const TFheGateBootstrappingCloudKeySet* bk) {
    std::unordered_map<int, LweSample*> temp_nodes;

    temp_nodes[115] = new_gate_bootstrapping_ciphertext(bk->params);
    bootsCONSTANT(temp_nodes[115], 1, bk);

    ...  
    ...  
    ...  

    bootsCOPY(&result[2], temp_nodes[459], bk);
    bootsCOPY(&result[3], temp_nodes[460], bk);
    bootsCOPY(&result[4], temp_nodes[461], bk);
    bootsCOPY(&result[5], temp_nodes[462], bk);
    for (auto it = temp_nodes.begin(); it != temp_nodes.end(); ++it) {
        delete_gate_bootstrapping_ciphertext(it->second);
    }
}
```
plaintext(28): do or do not there is no try
Encryption done
Sanity check by decryption:
do or do not there is no try

Server side computation:
char 0 done.
char 1 done.
char 2 done.
char 3 done.
char 4 done.
char 5 done.
char 6 done.
char 7 done.
char 8 done.
char 9 done.
char 10 done.
char 11 done.
char 12 done.
char 13 done.
char 14 done.
char 15 done.
char 16 done.
char 17 done.
char 18 done.
char 19 done.
char 20 done.
char 21 done.
char 22 done.
char 23 done.
char 24 done.
char 25 done.
char 26 done.
char 27 done.
Total time: 28 secs
Computation done

Decrypted result: Do or Do Not There Is No Try
Decryption done
plaintext(2R): do or do not there is no try
Encryption done
Sanity check by decryption:
do or do not there is no try

Server side computation:
char 0 done.
char 1 done.
char 2 done.
char 3 done.
char 4 done.
char 5 done.
char 6 done.
char 7 done.
char 8 done.
char 9 done.
char 10 done.
char 11 done.
char 12 done.
char 13 done.
char 14 done.
char 15 done.
char 16 done.
char 17 done.
char 18 done.
char 19 done.
char 20 done.
char 21 done.
char 22 done.
char 23 done.
char 24 done.
char 25 done.
char 26 done.
char 27 done.
Total time: 28 secs
Computation done

Decrypted result: Do Or Do Not There Is No Try
Decryption done
Execution engine: Multi-threaded Interpreter

plaintext size: 32
plaintext: 'do or do not there is no try'
Encryption done
Initial state check by decryption: do or do not there is no try

Starting!

Server side computation:
Total time: 0.756137 secs
CPU time: 46.4172 secs
Computation done

Decrypted result: Do Or Do Not There Is No Try
Decryption done
Interpreter

Straight line: fully serial graph
Not straight line: parallelism to exploit!

1. Toposort graph
2. Assign a thread as a node’s inputs are ready
3. See if any connected outputs are ready
4. Repeat until done
FHE Op Scheduler

Transpiler
- Single Threaded
- 8 cores

Interpreter
- Multi-threaded
- 96 cores

Server side computation:
Total time: 0.756137 secs
CPU time: 46.4172 secs
Computation done
Transpiler: Putting it all together

- Metadata
- C++
- Parser
- [IR] Circuit
- FHE Optimizer
- [IR] FHE Optimized Circuit
- Backend: Execution Engine
- FHE Cryptosystem API
- FHE Circuit Scheduler
- FHE* - API for Encryption/Decryption
Slight Detour
What FHE programs can you transpile?

**Static upper bounds**

**Bounded size**
Array and loop size needs to be known at compile time

**Bounded type**
Data types need to be known at compile time
Loops & circuit size

Loop unrolling
- Large circuit
- More optimizations across loop
- Slow compile time
- Fast execution time

Functional unit with state
- Small circuit
- Less optimizations only within loop
- Fast compile time
- Slow execution time

*Will revisit loops size again*
Loops & circuit size

Loop unrolling

0.728375 secs
Here is no try --Y'

0.00252208 secs)

Encryption: try --Y

Server side computation:

Total time: 0.456169 secs
CPU time: 27.4126 secs
Computation done

Not There Is No Try --Y

9 secs)

Functional unit with state

char 22: 0.158706 secs (0.411649 CPU secs)
char 23: 0.156575 secs (0.408601 CPU secs)
char 24: 0.159224 secs (0.414565 CPU secs)
char 25: 0.160251 secs (0.414091 CPU secs)
char 26: 0.16043 secs (0.412085 CPU secs)
char 27: 0.15928 secs (0.408458 CPU secs)
char 28: 0.160493 secs (0.412124 CPU secs)
char 29: 0.155668 secs (0.404195 CPU secs)
char 30: 0.159458 secs (0.411151 CPU secs)
char 31: 0.160049 secs (0.414351 CPU secs)

Total time: 5.10018 secs
Total CPU time: 13.2504 secs
Computation done
No Try --Y