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

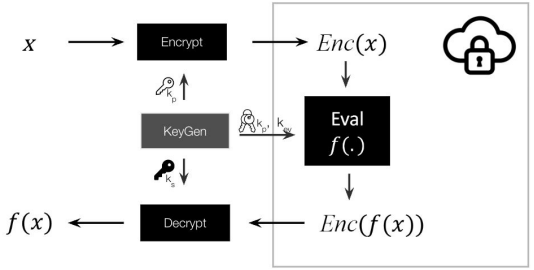
Shruthi Gorontala
Tech Lead
gshruthi@

Jeremy Kun
Cryptography Engineer
jkun@

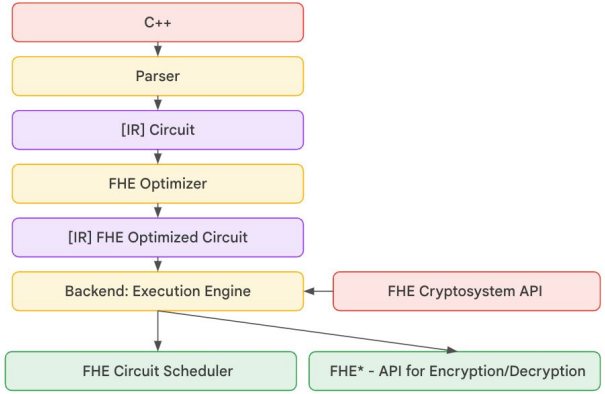
Cathie Yun
Cryptography Engineer
cathieyun@

AGENDA

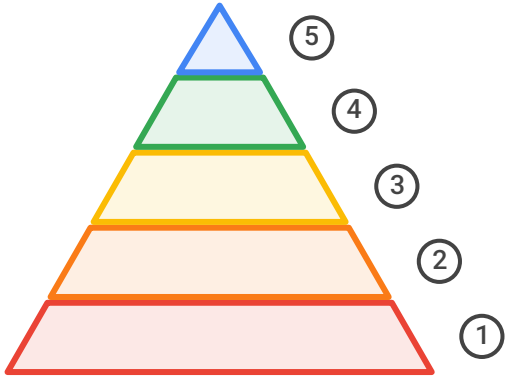
FHE Background



FHE Transpiler



FHE Hierarchy of Needs



1.0

Why should we care about Fully Homomorphic Encryption (FHE)?

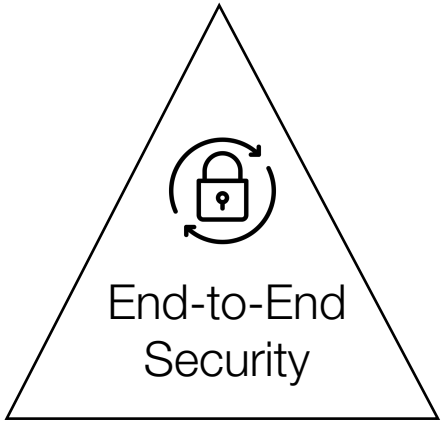
Modern Cryptography

data at rest
secure storage

Ubiquitous Adoption

Conventional Crypto

Encryption & Digital Signatures



Just Starting

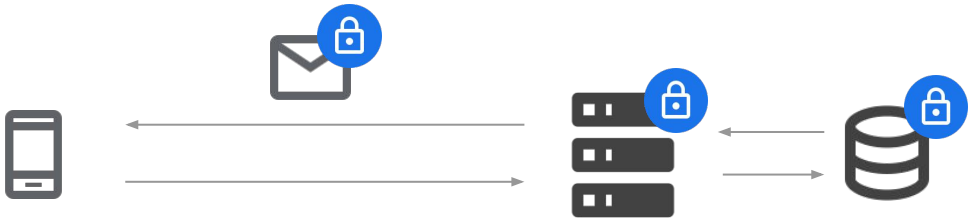
Advanced Crypto

- Homomorphic Encryption
- Secure Multi-party Computation
- Zero Knowledge Proofs

data in transit
secure communication

data in use
secure computation

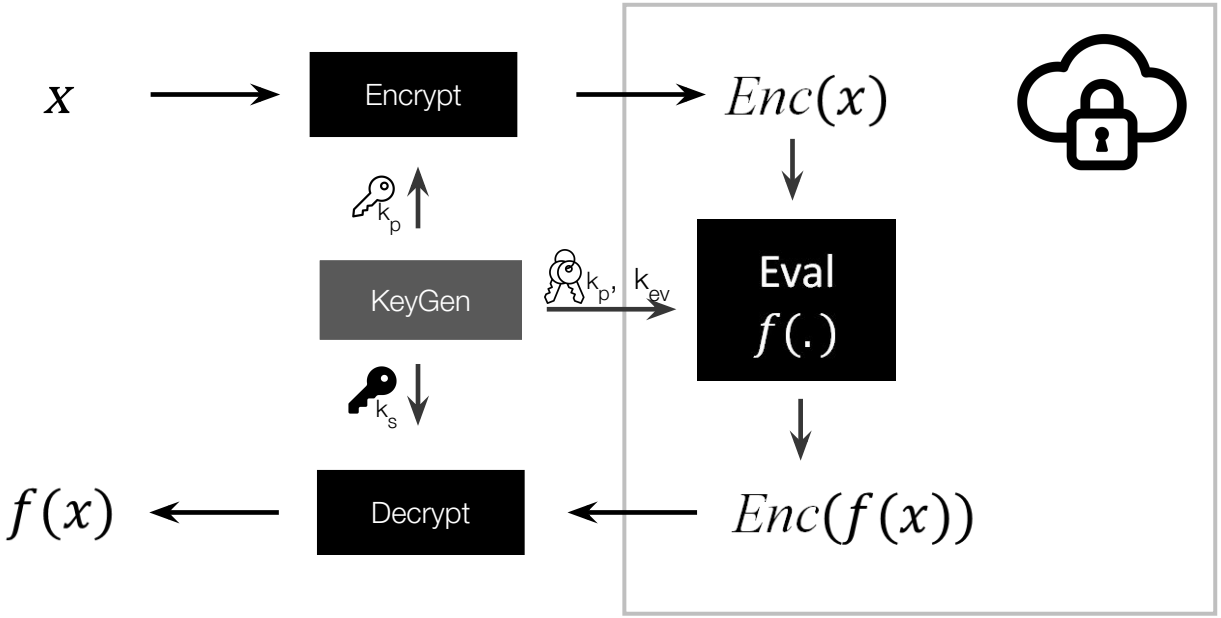
Without FHE



With FHE

Fully Homomorphic Encryption

Enables **computation** on encrypted data

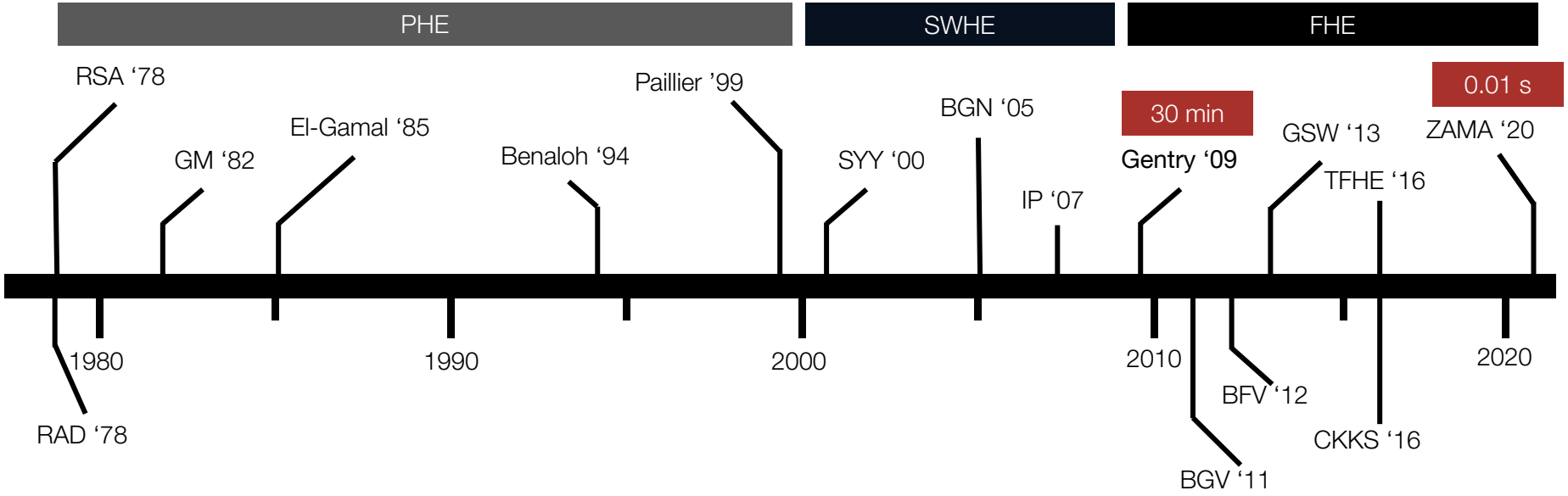


Delegate the **processing** of data without giving away **access** to it

1.1

FHE: A Brief History

40 Years of FHE History



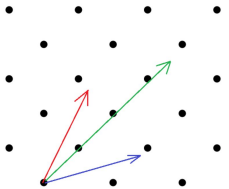
Reference: Alexander Viand

Problem

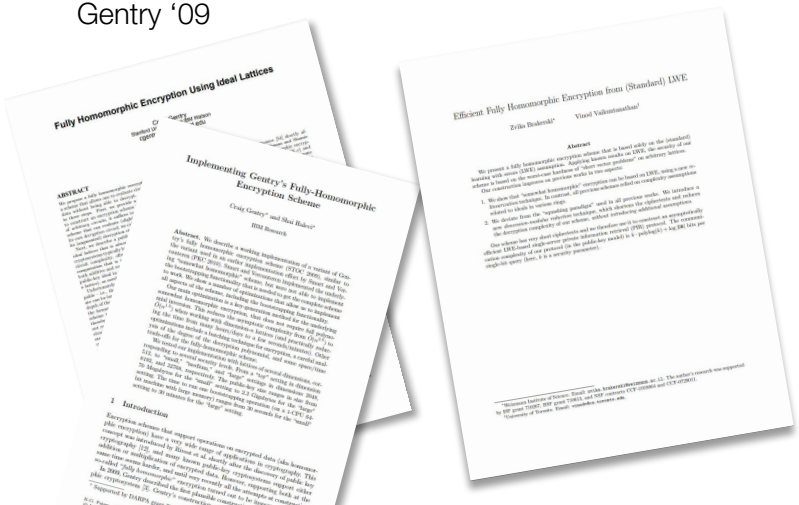
Learning with Errors (LWE)

FHE: A Brief History

Learning **W**ith **E**rrors



Gentry ‘09



Reference: Alexander Viand

Solving systems of linear equations

Linear system problem: given blue and green, find red

$$\mathbb{Z}_{13}^{7\times 4} \quad \mathbb{Z}_{13}^{4\times 1} \quad \mathbb{Z}_{13}^{7\times 1}$$

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

x

Secret

=

4
8
1
10
4
12
9

Learning with Errors (LWE) problem

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

$\mathbb{Z}_{13}^{7\times 4}$

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

x

$\mathbb{Z}_{13}^{4\times 1}$

Secret

+

$\mathbb{Z}_{13}^{7\times 1}$

1
0
1
1
-1
0
1

Small noise

=

$\mathbb{Z}_{13}^{7\times 1}$

4
8
1
10
4
12
9

LWE Encryption

Post Quantum Cryptography

Encrypting with LWE

$$\mathbb{Z}_{13}^{7\times 4} \times \mathbb{Z}_{13}^{4\times 1} + \mathbb{Z}_{13}^{7\times 1} + \mathbb{Z}_{13}^{7\times 1} = \mathbb{Z}_{13}^{7\times 1}$$

$$\begin{bmatrix} 3 & 3 & 11 & 9 \\ 6 & 5 & 7 & 5 \\ 4 & 8 & 1 & 11 \\ 2 & 6 & 2 & 2 \\ 11 & 2 & 3 & 4 \\ 5 & 8 & 12 & 2 \\ 3 & 3 & 5 & 0 \end{bmatrix} \times \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ -1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 2 \\ 11 \\ 5 \\ 12 \\ 9 \end{bmatrix}$$

Secret

Small noise

Message

Encryption

Encrypting with LWE

Toy world

$\mathbb{Z}_{13}^{7\times 4}$

4	1	11	10
5	5	9	5
3	9	0	10
1	3	3	2
12	7	3	4
6	5	11	4
3	3	5	0

Real world

$\mathbb{Z}_{2^{15}}^{2430\times 750}$

3673	12393	11334	...
5484	599	9200	...
13455	9830	0	...
19321	3948	383	...
...	...	...	...
...	...	...	...

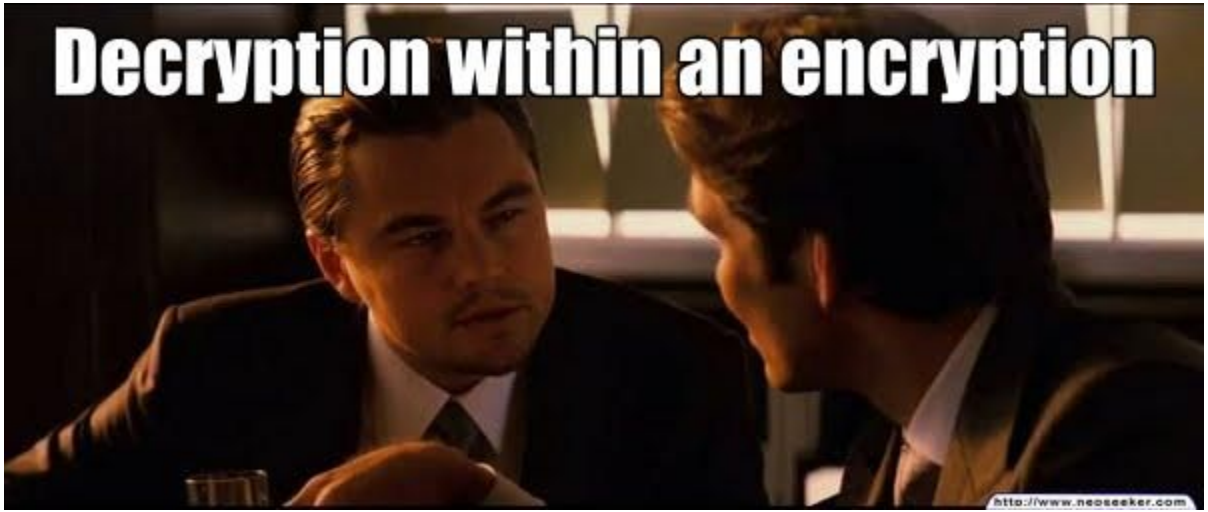
Rule of thumb (bit encryption): dimension more than 750, modulus 16+ bits.
Rule of thumb (homomorphic encryption): dimension >2000, modulus >60 bits.

FHE Computations and Noise

Decryption fails if error grows too large

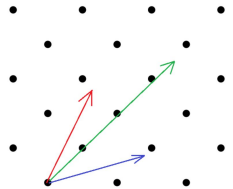
Bootstrapping

Homomorphically evaluates the decryption circuit to reduce noise



FHE: A Brief History

$Enc(0) \odot Enc(1)$



Ring-**L**earning **W**ith **E**rrors

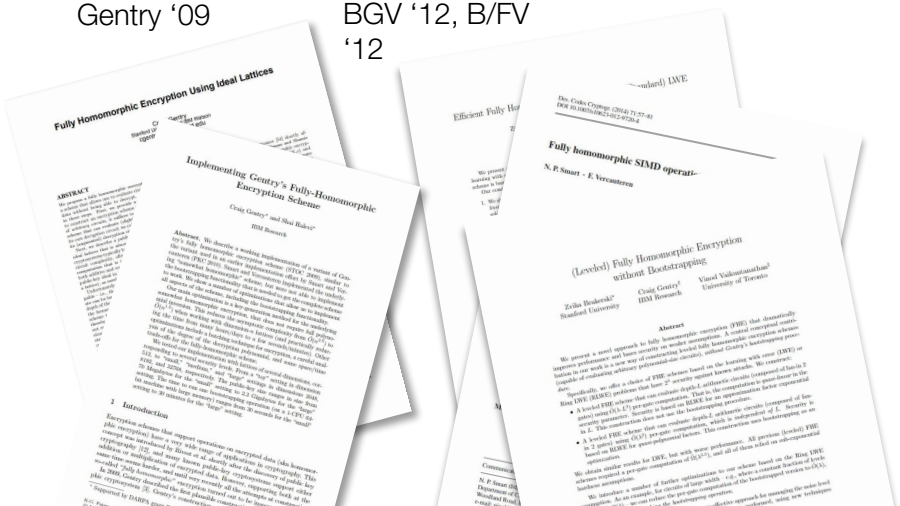
$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

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

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

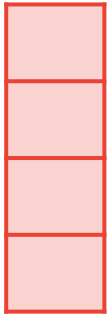
Gentry '09

BGV '12, B/FV '12



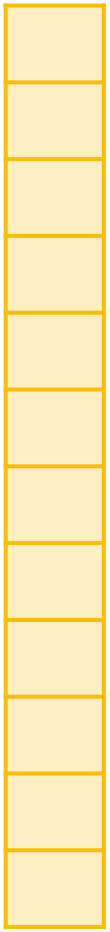
4	1	11	10
-10	4	1	11
-11	-10	4	1
-1	-11	-10	4
12	7	3	9
-9	12	7	3
-3	-9	12	7
-7	-3	-9	12
8	7	1	0
0	8	7	1
-1	0	8	7
-7	-1	0	8

x



Secret

+



Small noise

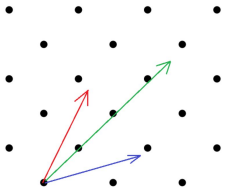
=

4
8
1
10
4
12
9
0
4
7
11
8

Encryption

FHE: A Brief History

$Enc(0) \odot Enc(1)$



Gentry '09

Ring-**L**earning **W**ith **E**rrors

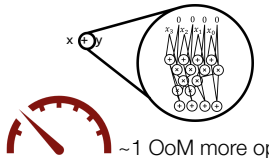
$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

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

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

BGV '12, B/FV '12

Fast Binary FHE



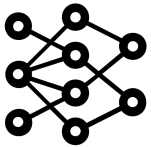
~1 OoM more operations but each operation is incredibly fast

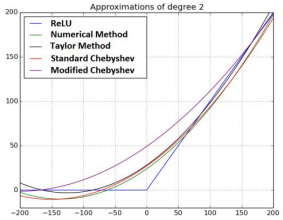
TFHE '16

CKKS '16

Approximate FHE

$Dec\big(Enc(m)\big) \approx m$





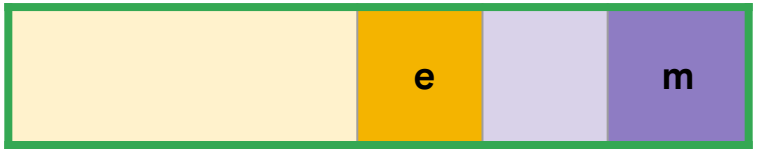
(a) Approximation of ReLU using different methods

Figure from CryptoDL: Deep Neural Networks over Encrypted Data [HTG17]

FHE Schemes

Vanilla LWE+

b = a.s + e + {q/2 if m = 1, 0, if m = 0}

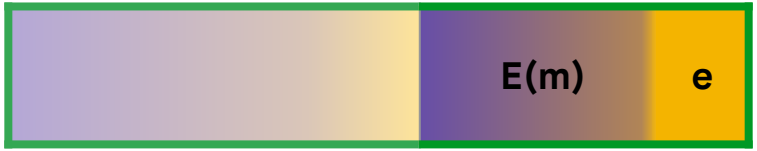


BGV

b = a.s + te + m , m is in Z_t

BFV

b = a.s + e + [q/t]m, m is scaled

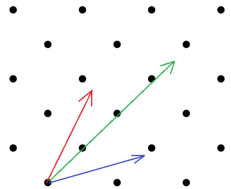


CKKS

B = a.s + e + **Δ*Inv**F(m) - approximations

FHE: A Brief History

$Enc(0) \odot Enc(1)$



Gentry '09

Ring-**L**earning **W**ith **E**rrors

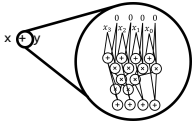
$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

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

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

BGV '12, B/FV '12

Fast Binary FHE



TFHE '16

Look-Up Tables

X	f(x)
-1	-0.84
-0.5	-0.47
0	0
0.5	0.47
1	0.84

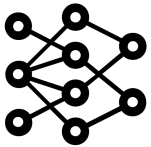
ZAMA '21

DPRIVE

CKKS '16

Approximate FHE

$Dec(Enc(m)) \approx m$



Hardware Acceleration

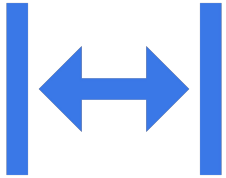
intel. Duality galois SRI International

2.0

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

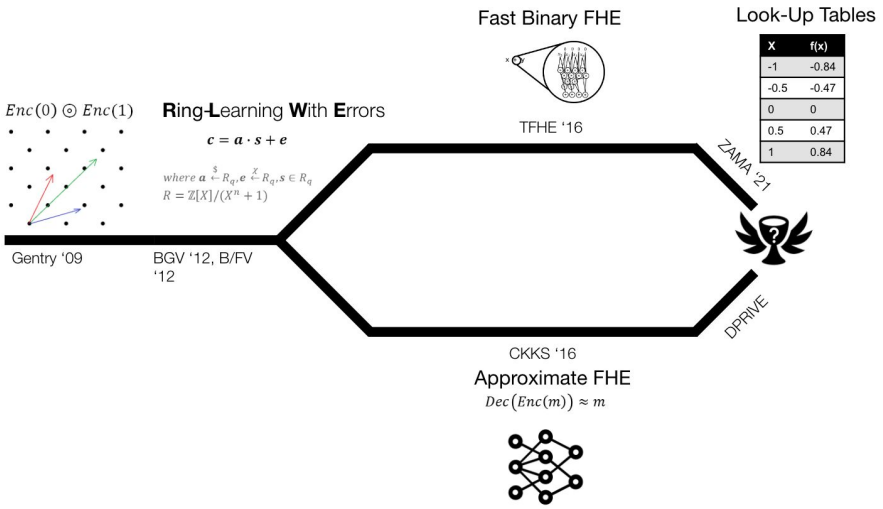
Need to abstract away cryptographic details from engineers

Cryptography

Engineering

Reference: SoK: Fully Homomorphic Encryption Compilers, Viand et al

FHE challenges : Cryptographic



01 **Circuit depth analysis**

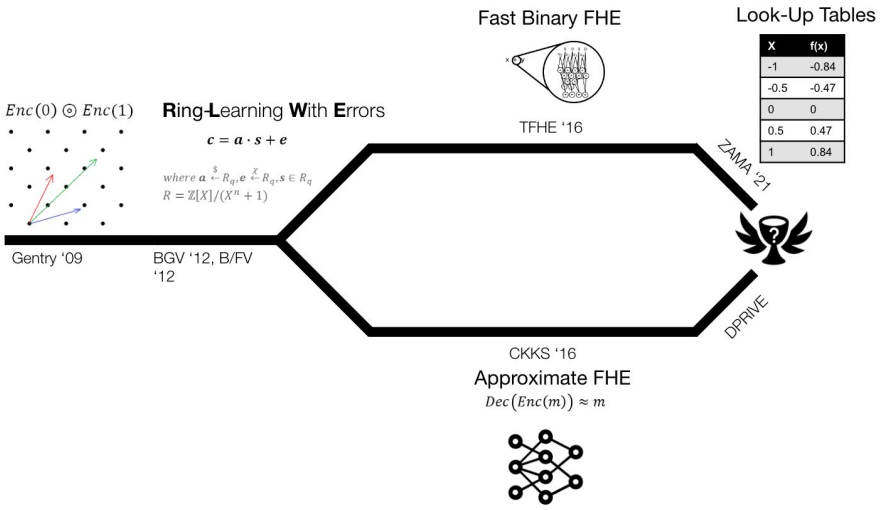
02 **Noise tracking**

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

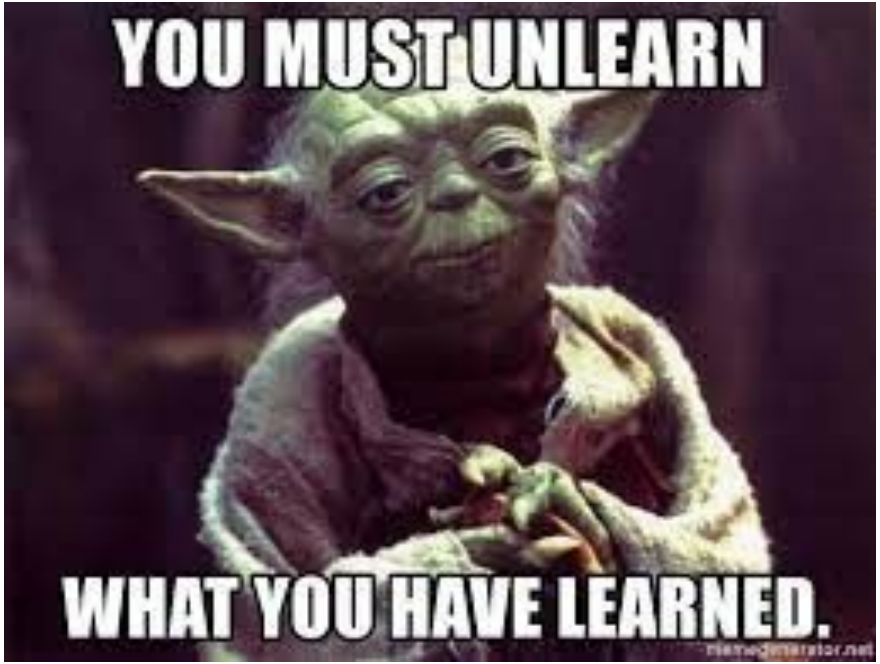
No branch and bound optimizations

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

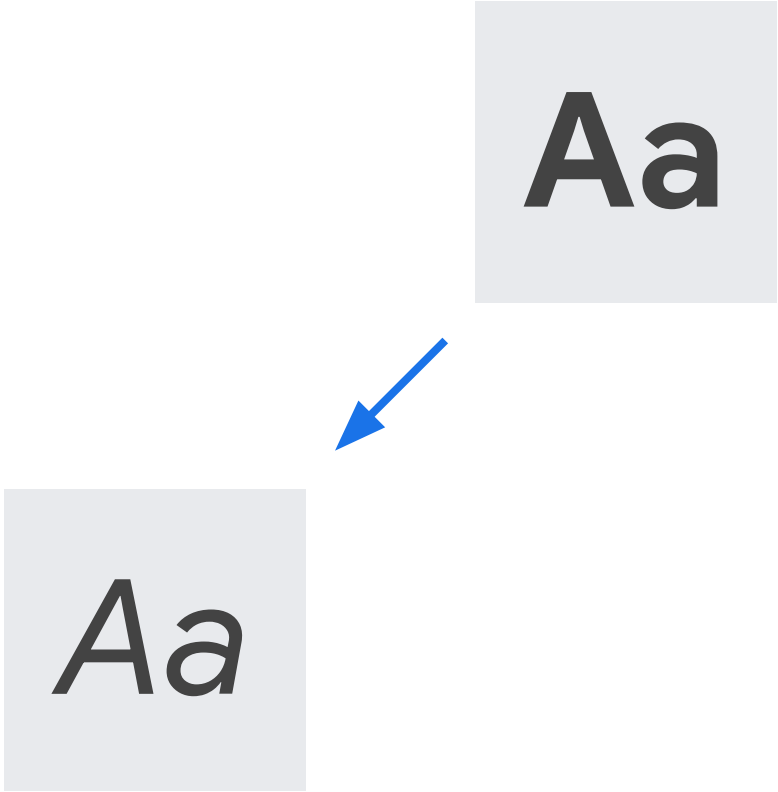


2.1

FHE Transpiler: Reusing hardware tooling

FHE transpiler

Allows us to easily convert existing code
that works on plaintext to work on ciphertext.



FHE transpiler

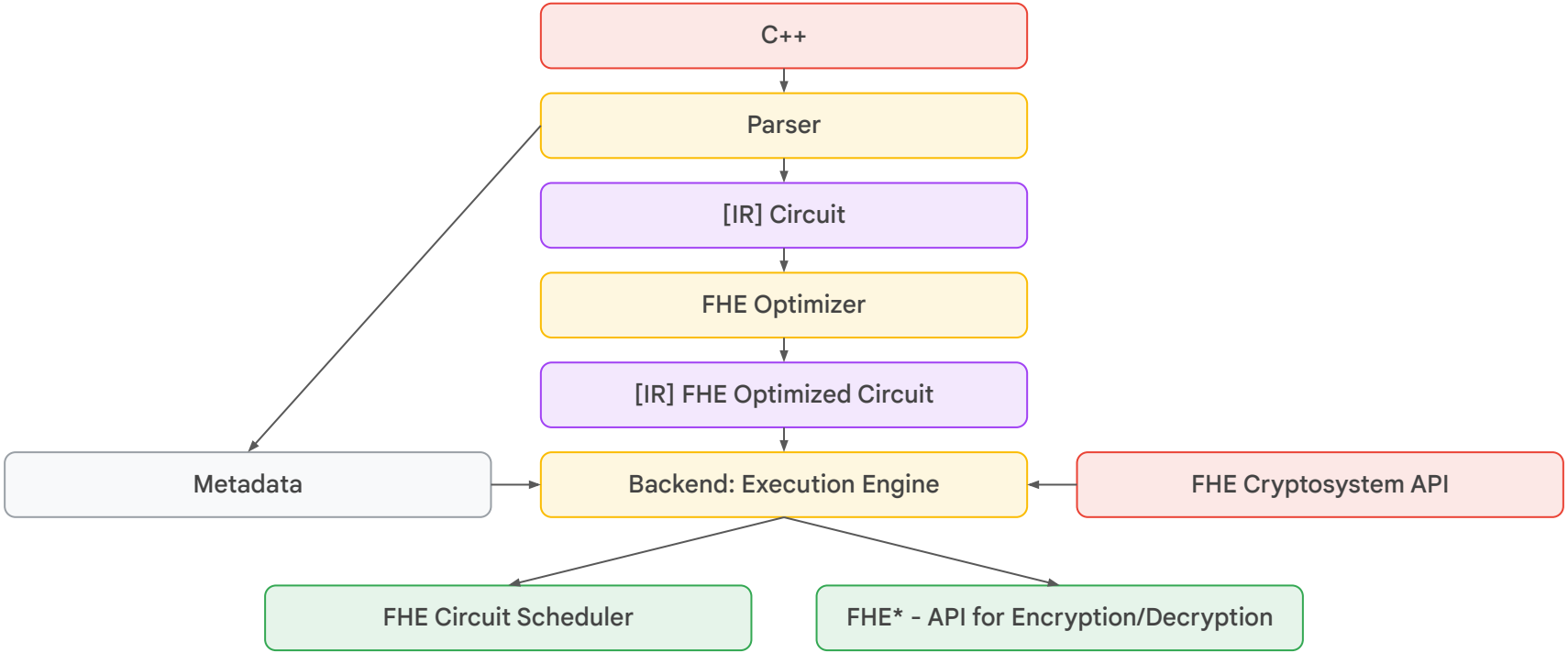
Convert C++ code that works on plaintext to work on ciphertext.

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

→

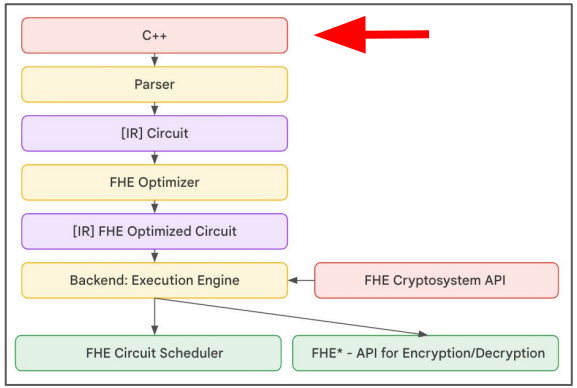
```c
#include <tfhe.h>

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);
}
```

Transpiler Architecture



Basic Example: String Capitalization

The following code capitalizes words in a string and is written using standard C++ syntax.

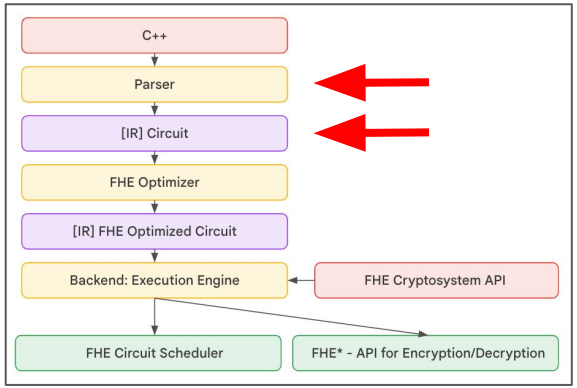


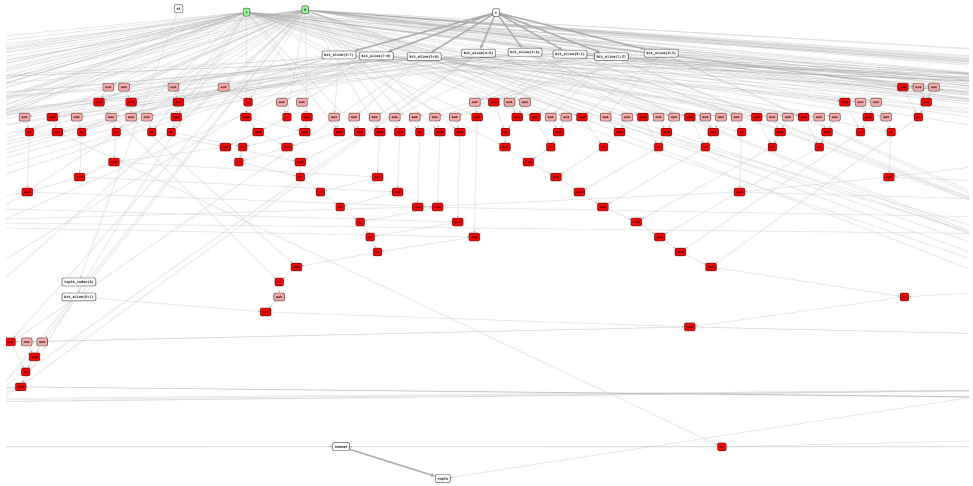

```
#include "string_cap.h"
State::State()
 : last_was_space_(true) {
}

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);
}
```

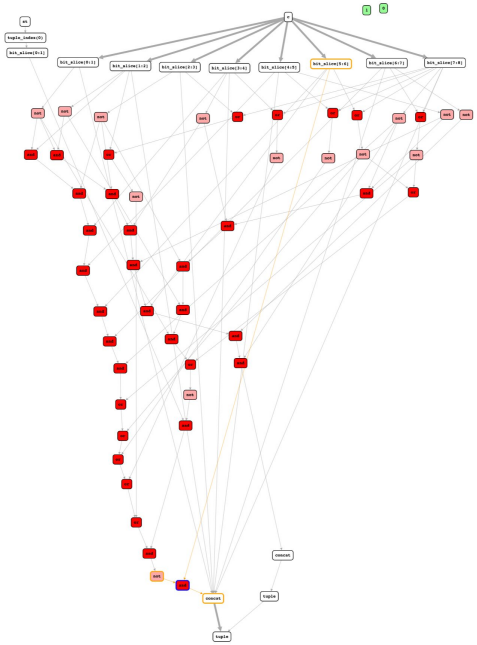
XLS IR

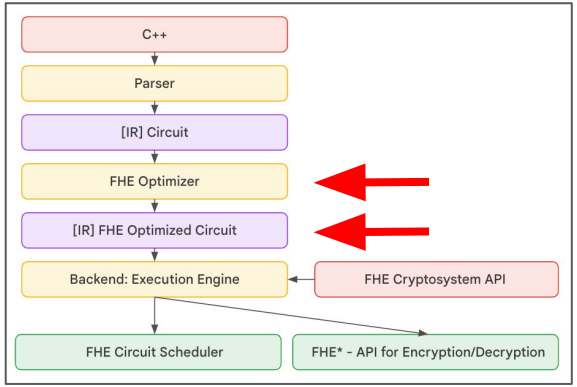
Intermediate representation

```
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)
}
```





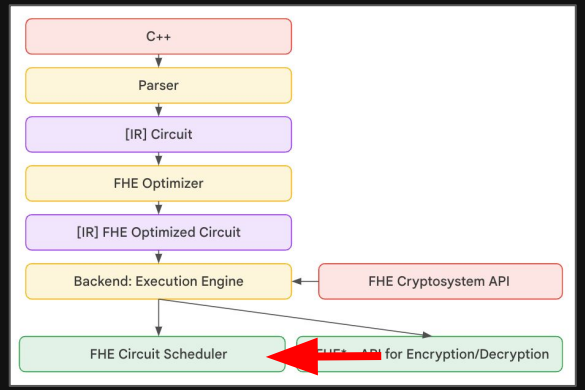



C++
Parser
[IR] Circuit
FHE Optimizer
[IR] FHE Optimized Circuit
Backend: Execution Engine
FHE Cryptosystem API
FHE Circuit Scheduler
FHE* - API for Encryption/Decryption

```
plaintext(28):do or do not there is no try
Encryption done
Sanity check by decryption:
do or do not there is no try
```

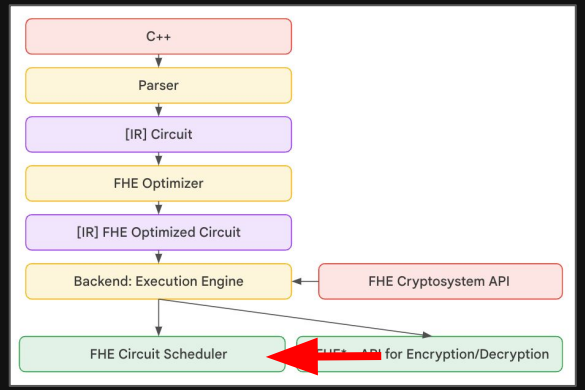
```
Server side computation:
char 0 done.
char 1 done.
char 2 done.
char 3 done.
char 4 done.
char 5 done.
char 6 done.
char 7 done.
char 8 done.
char 9 done.
char 10 done.
char 11 done.
char 12 done.
char 13 done.
char 14 done.
char 15 done.
char 16 done.
char 17 done.
char 18 done.
char 19 done.
char 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
```

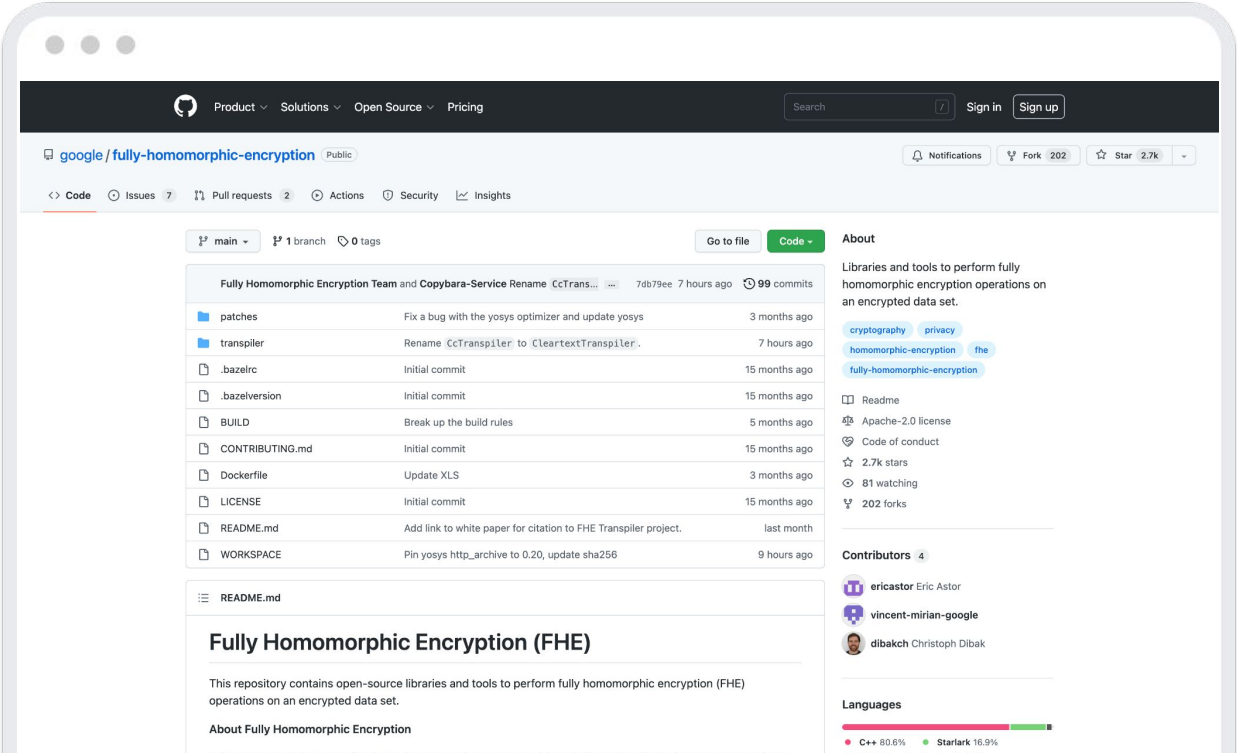


Execution engine: Multi-threaded Interpreter
(96 cores)

```
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

                                        Server side computation:
Starting!
                                        Total time: 0.756137 secs
                                          CPU time: 46.4172 secs
                                        Computation done
Decrypted result: Do Or Do Not There Is No Try
Decryption done
```



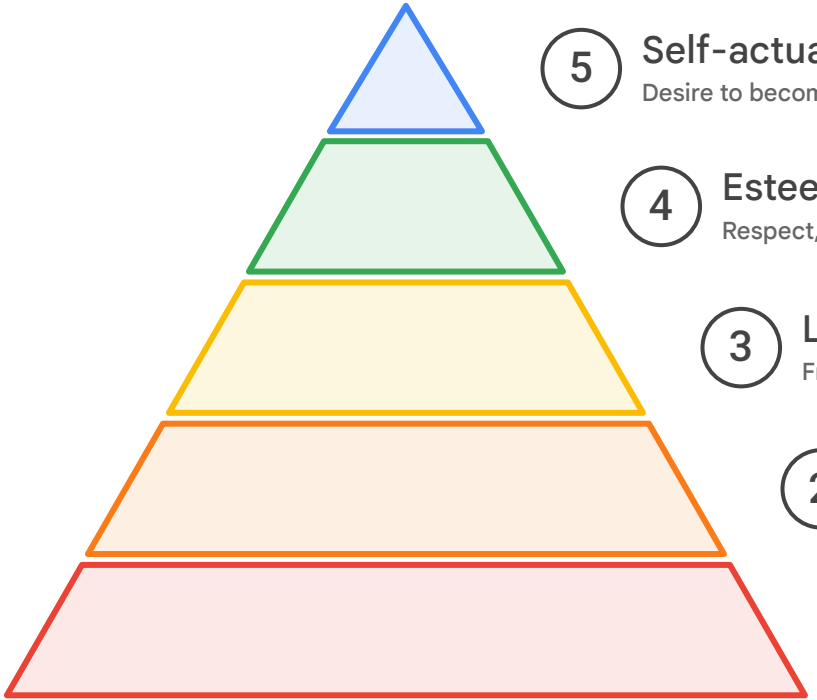
Open Source


Product
Solutions
Open Source
Pricing
Search
Sign in
Sign up
google / fully-homomorphic-encryption
Public
Notifications
Fork 202
Star 2.7k
Code
Issues 7
Pull requests 2
Actions
Security
Insights
main
1 branch
0 tags
Go to file
Code
Fully Homomorphic Encryption Team and Copybara-Service Rename CcTrans...
7db79ee 7 hours ago
99 commits
patches
Fix a bug with the yosys optimizer and update yosys
3 months ago
transpiler
Rename CcTranspiler to CleartextTranspiler.
7 hours ago
.bazelrc
Initial commit
15 months ago
.bazelversion
Initial commit
15 months ago
BUILD
Break up the build rules
5 months ago
CONTRIBUTING.md
Initial commit
15 months ago
Dockerfile
Update XLS
3 months ago
LICENSE
Initial commit
15 months ago
README.md
Add link to white paper for citation to FHE Transpiler project.
last month
WORKSPACE
Pin yosys http_archive to 0.20, update sha256
9 hours ago
README.md
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
About
Libraries and tools to perform fully homomorphic encryption operations on an encrypted data set.
cryptography
privacy
homomorphic-encryption
fhe
fully-homomorphic-encryption
Readme
Apache-2.0 license
Code of conduct
2.7k stars
81 watching
202 forks
Contributors 4
ericastor Eric Astor
vincent-mirian-google
dibakch Christoph Dibak
Languages
C++ 80.6%
Starlark 16.9%

3.0

Making FHE usable

Maslow's Hierarchy of Needs



5 **Self-actualization**
Desire to become the most that one can be

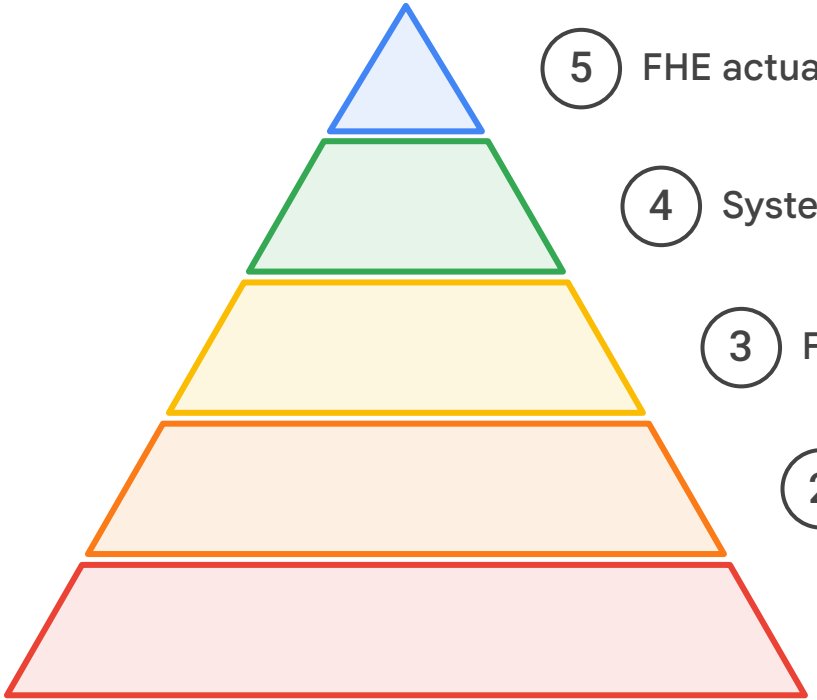
4 **Esteem**
Respect, self-esteem, status, recognition, strength, freedom

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

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

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

FHE Hierarchy of Needs



5 FHE actualization

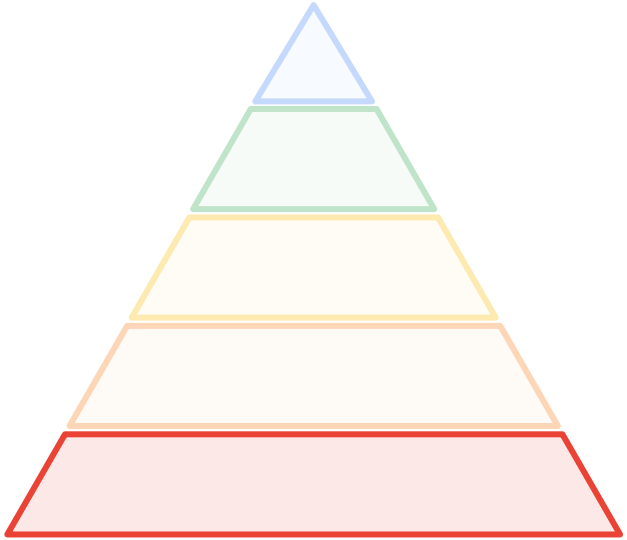
4 Systems integration / Privacy engineering

3 FHE application development

2 Compilers / Transpilers

1 FHE Instruction Set Architecture

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
- ...

2. Compilers / Transpilers

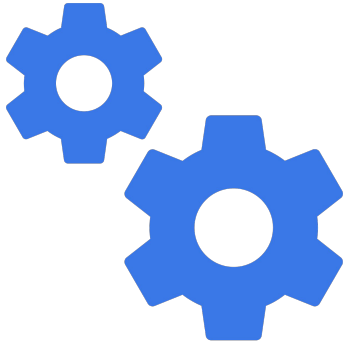
General Purpose Compilers
Boolean circuit optimizations

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

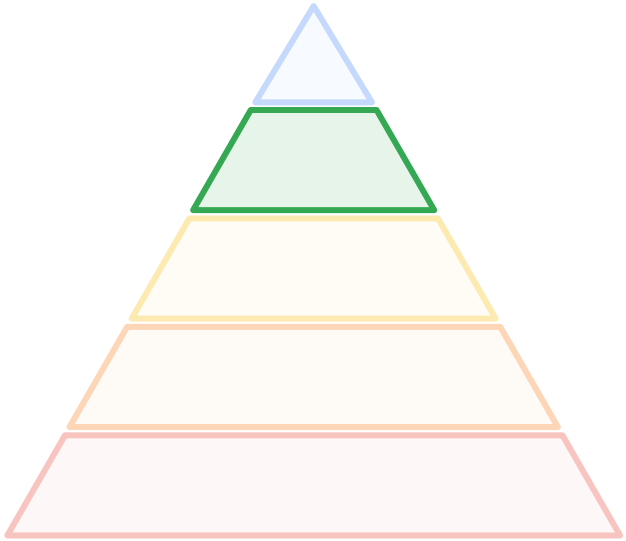
Trade Offs
Performance tradeoff across hardware

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

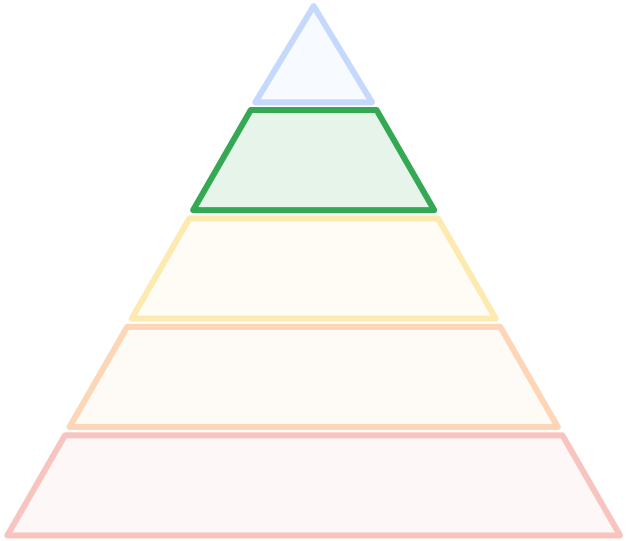
Trust Model Development
May need other Privacy Enhancing Technologies

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

Open Problem: Ciphertext Expansion
Packing and hybrid homomorphic encryption

4. Privacy Engineering

Trust Model



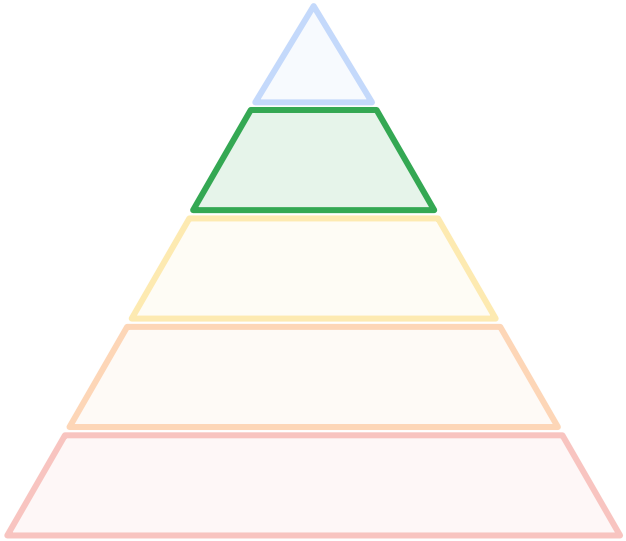
Adversarial Server
Zero Knowledge Proofs

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

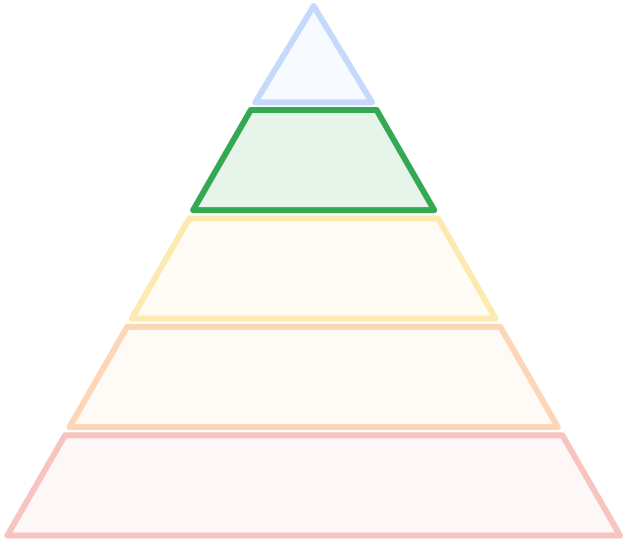
Quantifying privacy

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

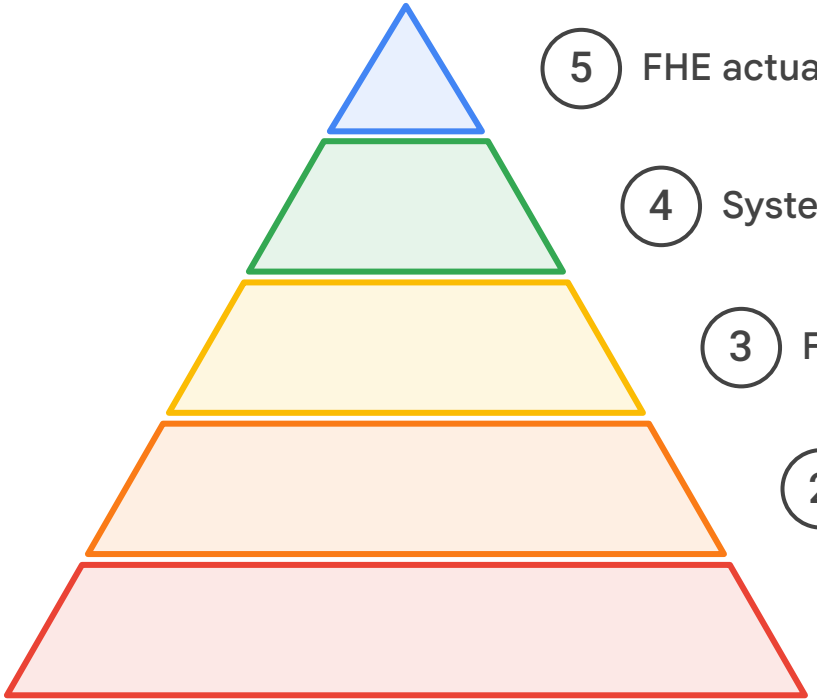
Latency / Network Bandwidth / Privacy
Metrics

$$$ for Hardware
Budget

Product Timelines
Launch dates

Software Developer Hours
Engineering time

FHE Hierarchy of Needs



5 FHE actualization

4 Systems integration / Privacy engineering

3 FHE application development

2 Compilers / Transpilers

1 FHE Instruction Set Architecture

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

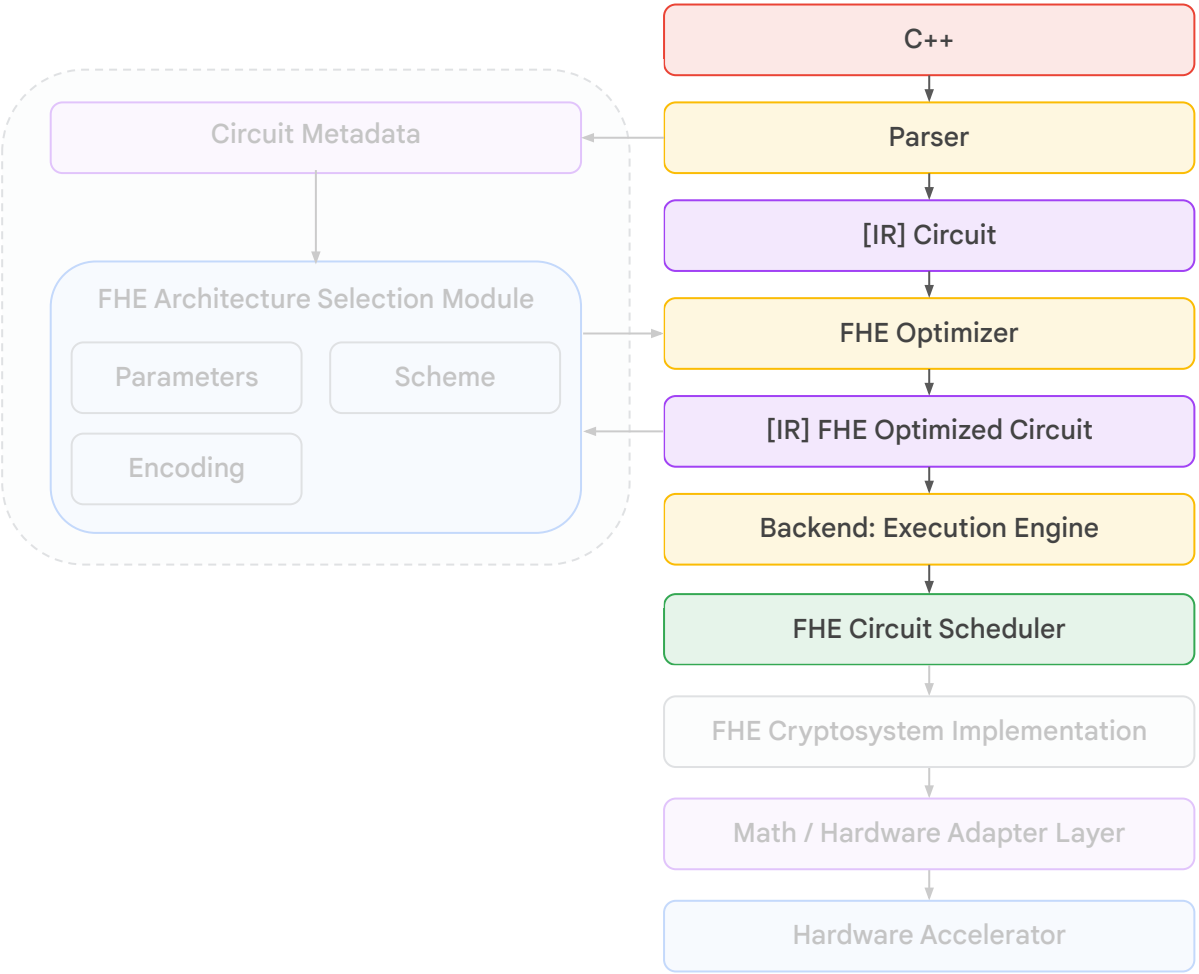
Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds

- Chilloti et al

TFHE API

```
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


C++
Parser
[IR] Circuit
FHE Optimizer
[IR] FHE Optimized Circuit
Backend: Execution Engine
FHE Circuit Scheduler
FHE Cryptosystem Implementation
Math / Hardware Adapter Layer
Hardware Accelerator
Circuit Metadata
FHE Architecture Selection Module
Parameters
Scheme
Encoding

Transpiler Design

Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

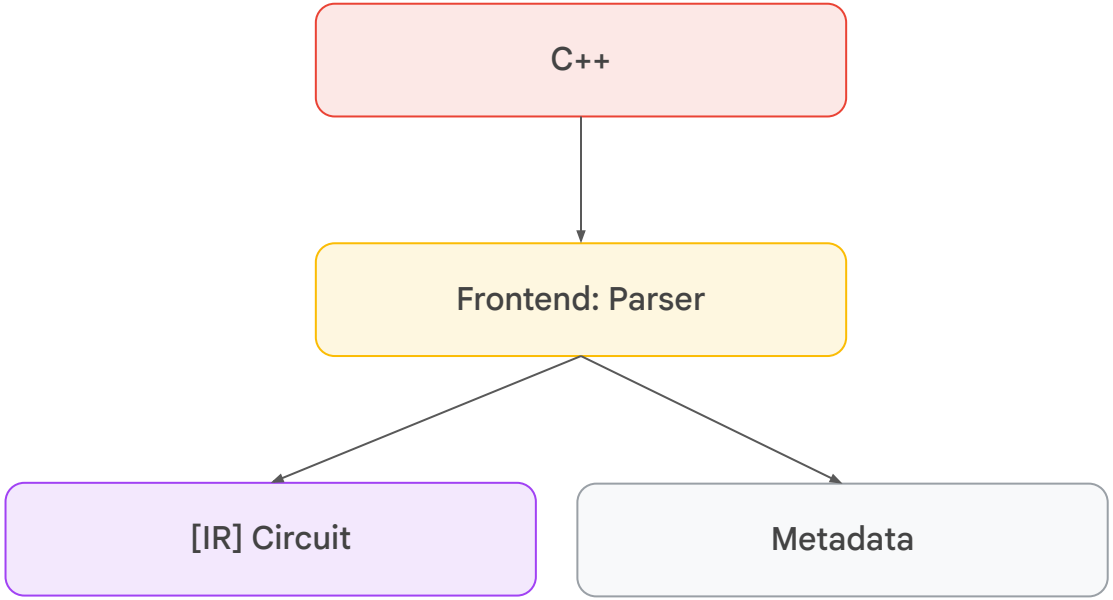
Transpiler Design

Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

Transpiler Frontend: Parser



Basic Example: String Capitalization

The following code capitalizes words in a string and is written using standard C++ syntax.

```
#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);
}
```

XLS IR

Intermediate representation

```
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

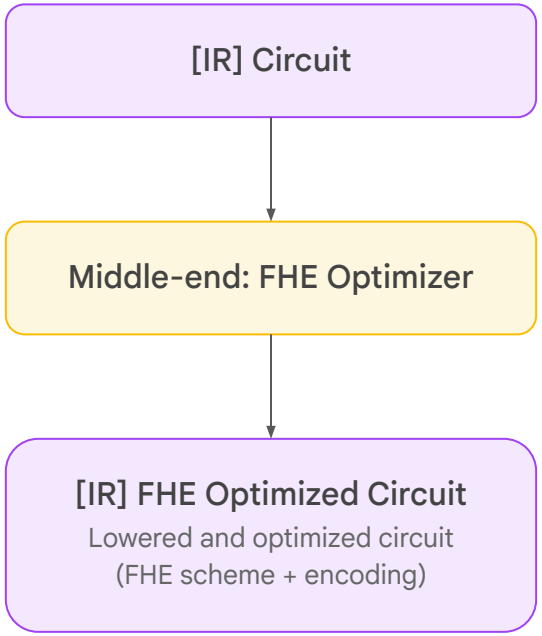
Transpiler Design

Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

Transpiler Middle-end: FHE Optimizer

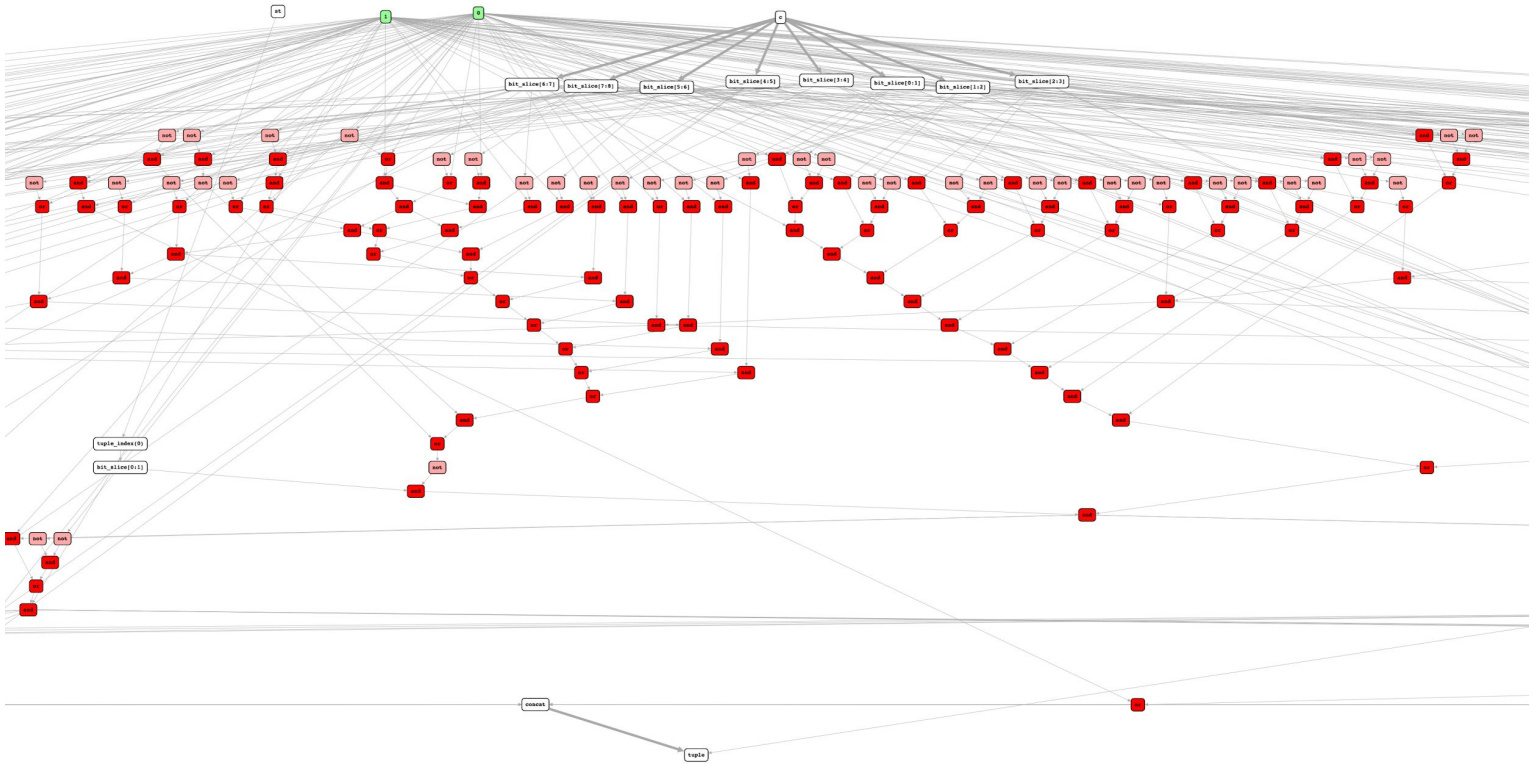


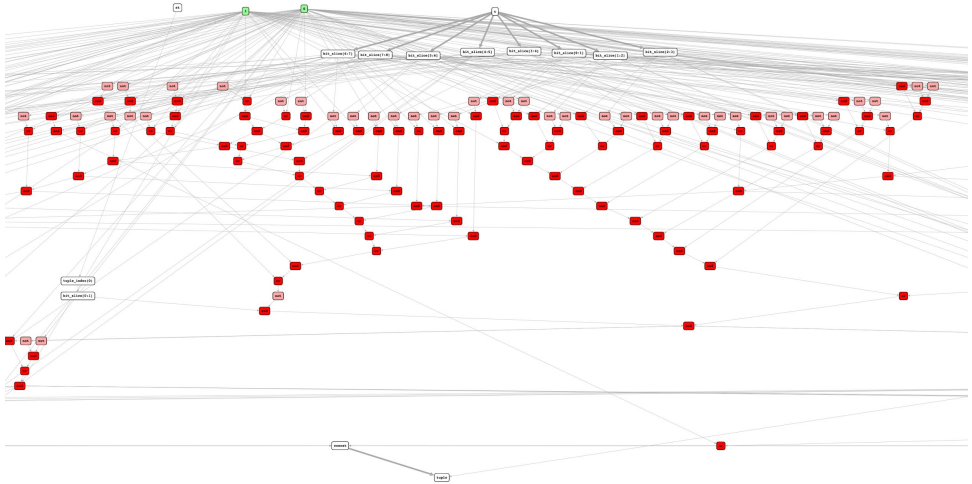
Booleanized XLS IR

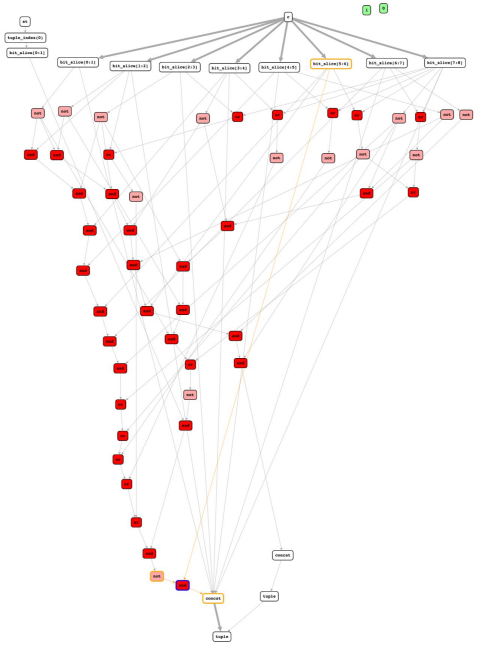
IR after running through the booleanifier.

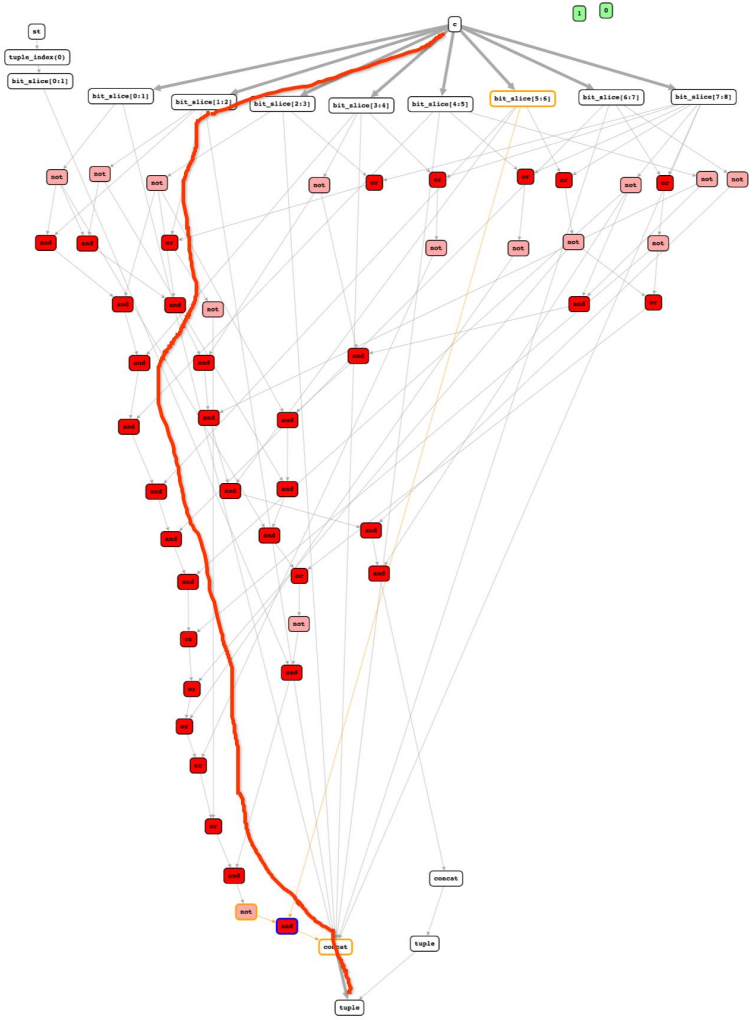
Now it only contains AND, OR, and NOT gates.

```
Boolean-ized IR:
package p

fn my_package_boolean(st: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  literal.115: bits[8] = literal(value=1)
  shrl.117: bits[8] = shrl(c, literal.115)
  shrl.119: bits[8] = shrl(shrl.117, literal.115)
  shrl.121: bits[8] = shrl(shrl.119, literal.115)
  shrl.123: bits[8] = shrl(shrl.121, literal.115)
  shrl.125: bits[8] = shrl(shrl.123, literal.115)
  shrl.127: bits[8] = shrl(shrl.125, literal.115)
  ...
  ...
  ...

  shrl.131: bits[8] = shrl(shrl.129, literal.115)
  and.196: bits[1] = and(and.189, or.195)
  and.263: bits[1] = and(and.256, or.262)
  or.346: bits[1] = or(or.345, and.342)
  ret tuple.468: (bits[8], (bits[1])) = tuple(concat.465, tuple.467)
}
```


st
1
0
c
bit_slice[6:7]
bit_slice[7:8]
bit_slice[5:6]
bit_slice[4:5]
bit_slice[3:4]
bit_slice[0:1]
bit_slice[1:2]
bit_slice[2:3]
tuple_index(0)
bit_slice[0:1]
concat
tuple






st
tuple_index(0)
bit_slice[0:1]
c
1
0
bit_slice[0:1]
bit_slice[1:2]
bit_slice[2:3]
bit_slice[3:4]
bit_slice[4:5]
bit_slice[5:6]
bit_slice[6:7]
bit_slice[7:8]
concat
tuple
concat
tuple

JSON Cytoscape graph

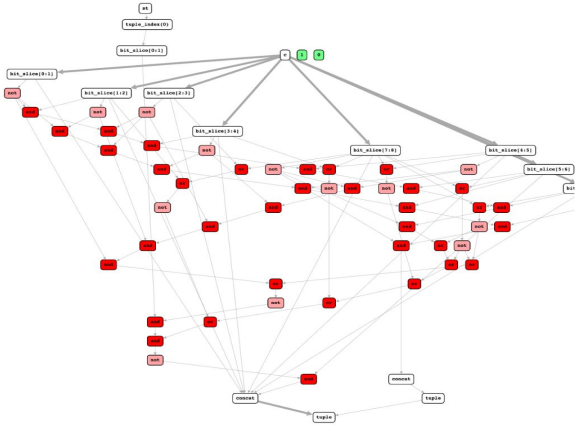
oload Choose file Browse

enchmarks... Opt lev Load

```
ckage my_package

my_package_boolean(st: (bits[1]), c: bits[8]) -> (bits[8], (bits[1]))

bit_slice.1075: bits[1] = bit_slice(c, start=0, width=1, id=1075)
not.1088: bits[1] = not(bit_slice.1075, id=1088)
bit_slice.1076: bits[1] = bit_slice(c, start=1, width=1, id=1076)
bit_slice.1077: bits[1] = bit_slice(c, start=2, width=1, id=1077)
and.1102: bits[1] = and(not.1088, bit_slice.1076, id=1102)
not.1086: bits[1] = not(bit_slice.1077, id=1086)
and.1103: bits[1] = and(and.1102, not.1086, id=1103)
bit_slice.1078: bits[1] = bit_slice(c, start=3, width=1, id=1078)
bit_slice.1082: bits[1] = bit_slice(c, start=7, width=1, id=1082)
bit_slice.1079: bits[1] = bit_slice(c, start=4, width=1, id=1079)
and.1104: bits[1] = and(and.1103, bit_slice.1078, id=1104)
not.1083: bits[1] = not(bit_slice.1082, id=1083)
not.1084: bits[1] = not(bit_slice.1079, id=1084)
and.1105: bits[1] = and(and.1104, bit_slice.1079, id=1105)
bit_slice.1080: bits[1] = bit_slice(c, start=5, width=1, id=1080)
and.1093: bits[1] = and(not.1083, not.1084, id=1093)
not.1085: bits[1] = not(bit_slice.1078, id=1085)
and.1106: bits[1] = and(and.1105, bit_slice.1080, id=1106)
bit_slice.1081: bits[1] = bit_slice(c, start=6, width=1, id=1081)
and.1094: bits[1] = and(and.1093, not.1085, id=1094)
and.1107: bits[1] = and(and.1106, bit_slice.1081, id=1107)
or.1089: bits[1] = or(bit_slice.1082, bit_slice.1081, id=1089)
or.1091: bits[1] = or(bit_slice.1082, bit_slice.1080, id=1091)
and.1095: bits[1] = and(and.1094, not.1086, id=1095)
not.1087: bits[1] = not(bit_slice.1076, id=1087)
and.1108: bits[1] = and(and.1107, not.1083, id=1108)
not.1090: bits[1] = not(or.1089, id=1090)
not.1092: bits[1] = not(or.1091, id=1092)
and.1096: bits[1] = and(and.1095, not.1087, id=1096)
or.1117: bits[1] = or(and.1108, not.1090, id=1117)
or.1109: bits[1] = or(bit_slice.1082, bit_slice.1079, id=1109)
and.1127: bits[1] = and(not.1088, not.1087, id=1127)
or.1100: bits[1] = or(not.1090, not.1092, id=1100)
and.1097: bits[1] = and(and.1096, not.1088, id=1097)
or.1118: bits[1] = or(or.1117, not.1092, id=1118)
not.1110: bits[1] = not(or.1109, id=1110)
or.1111: bits[1] = or(bit_slice.1082, bit_slice.1078, id=1111)
or.1113: bits[1] = or(bit_slice.1082, bit_slice.1077, id=1113)
and.1128: bits[1] = and(and.1127, not.1086, id=1128)
```

OK

iew Graph Critical Path Show only selected nodes



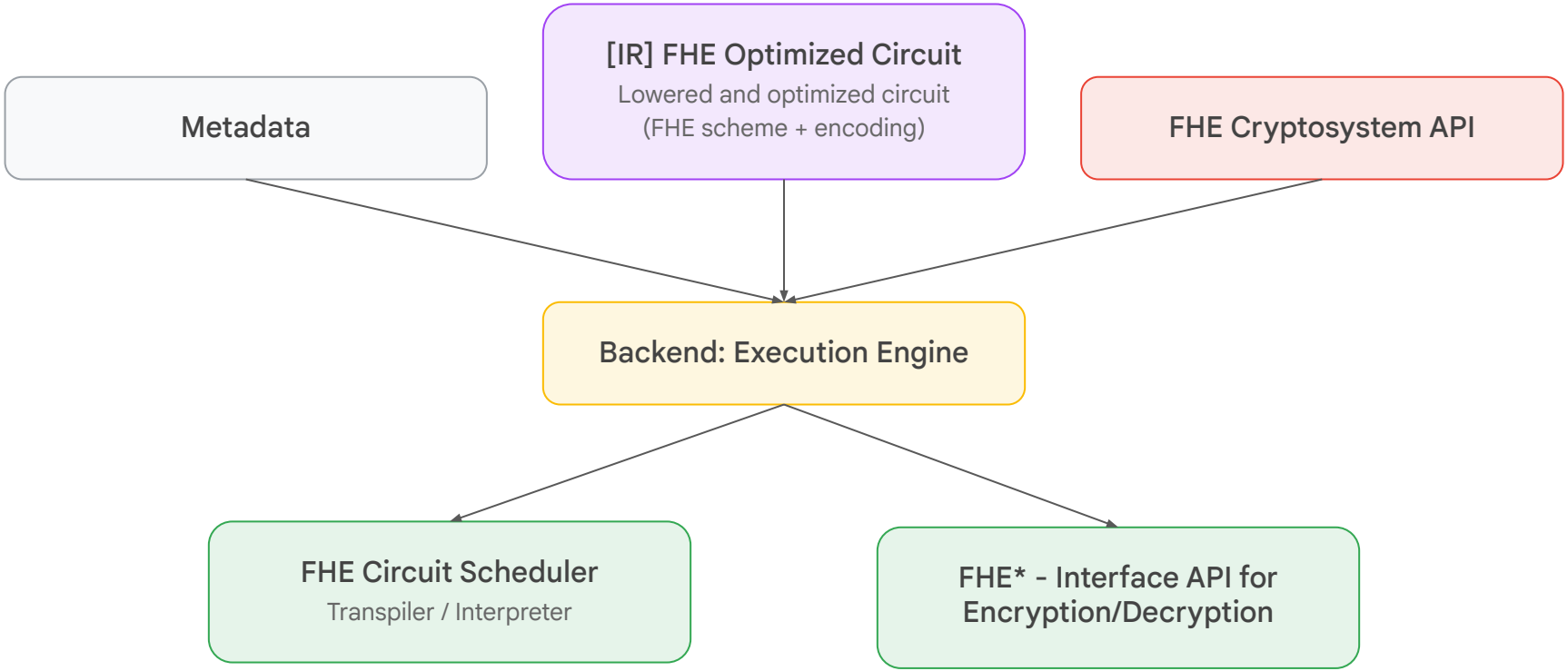
Transpiler Design

Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

Transpiler Backend: Execution Engine



FHE C++

C++ that now works on ciphertext.

```
#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(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
```

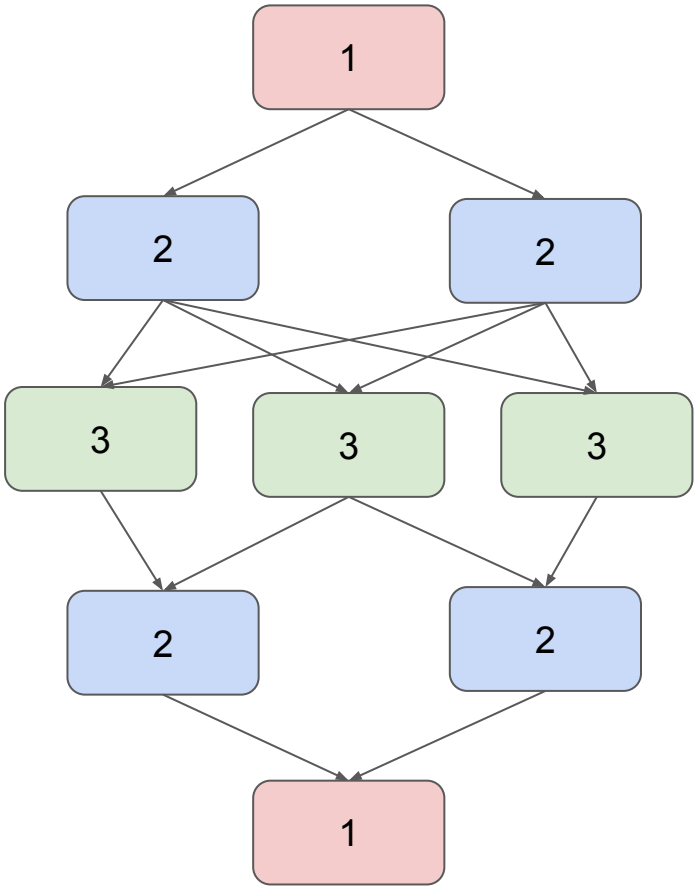
Execution engine: Multi-threaded Interpreter

```
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

                                        Server side computation:
Starting!
                                        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

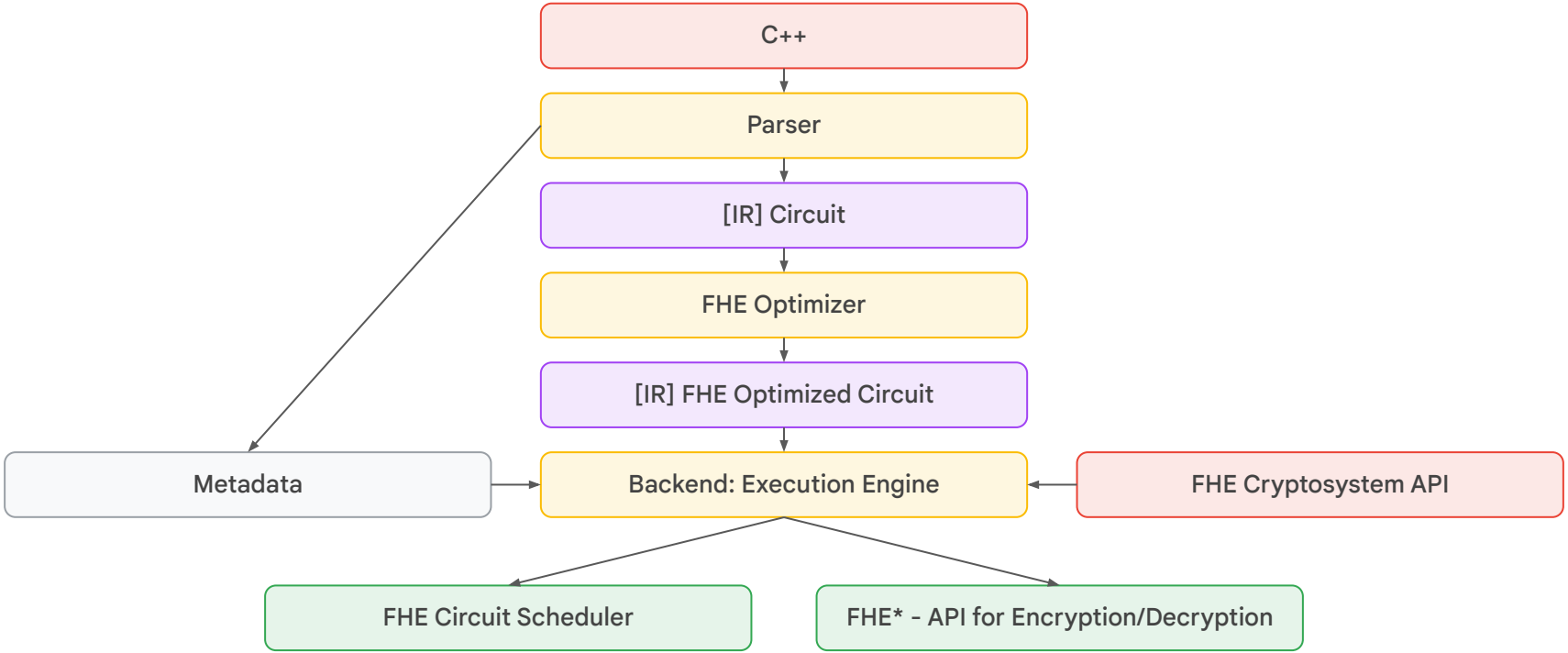
```
char 22 done.
char 23 done.
char 24 done.
char 25 done.
char 26 done.
char 27 done.
Total time : 28 secs
Computation done
t There Is No Try
```

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



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)
ryption:
ry --Y

                Server side computation:

                Total time: 0.456169 secs
                  CPU time: 27.4126 secs
                Computation done
 Not There Is No Try --Y
5 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
```