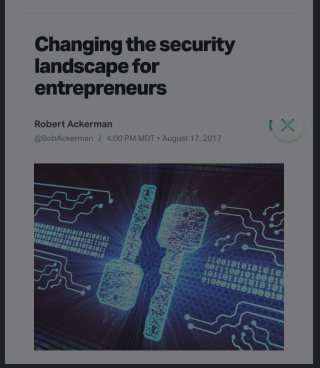
Google | Privacy, Safety & Security

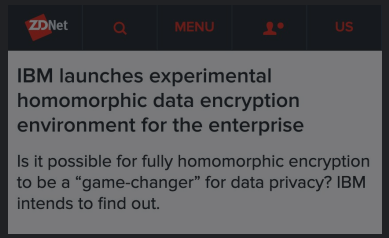
Unlocking the potential of Fully Homomorphic Encryption

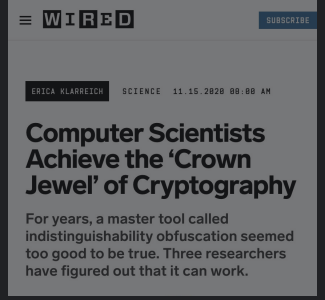
Shruthi Gorantala
gshruthi@google.com
October 30, 2023

Agenda

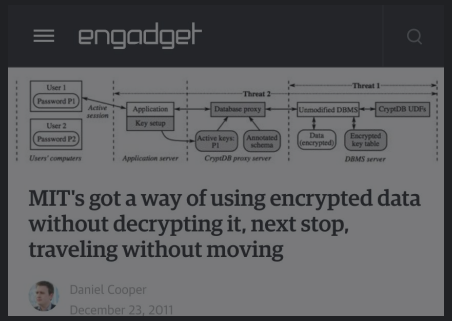
1. What is FHE
2. Brief history of FHE
3. FHE Transpiler
4. HEIR - Foundation for FHE Compilers
5. Questions



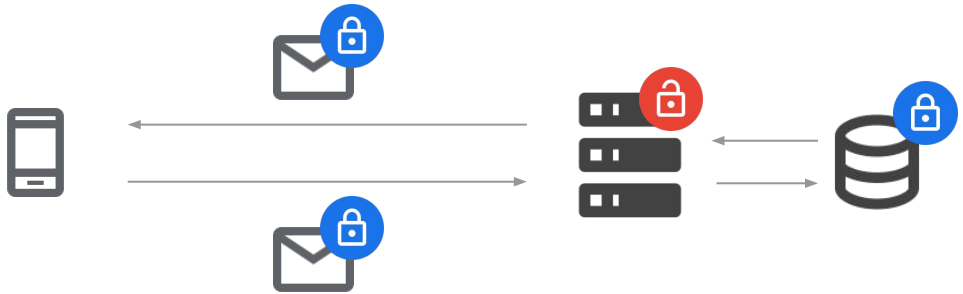




Fully Homomorphic Encryption (FHE) makes computations on encrypted data possible.

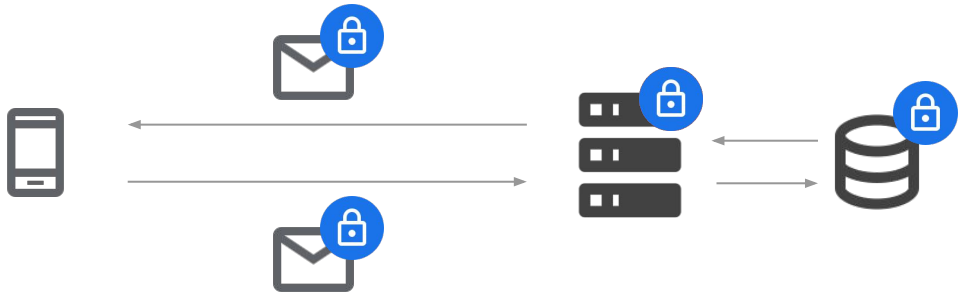


Privacy At Compute-time



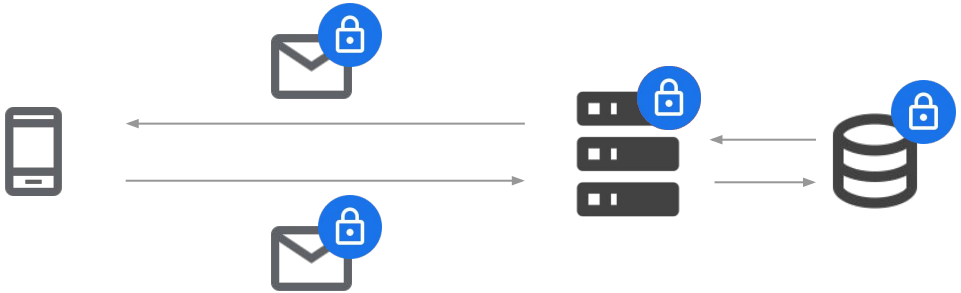
Without FHE

Privacy At Compute-time



With FHE

Privacy At Compute-time



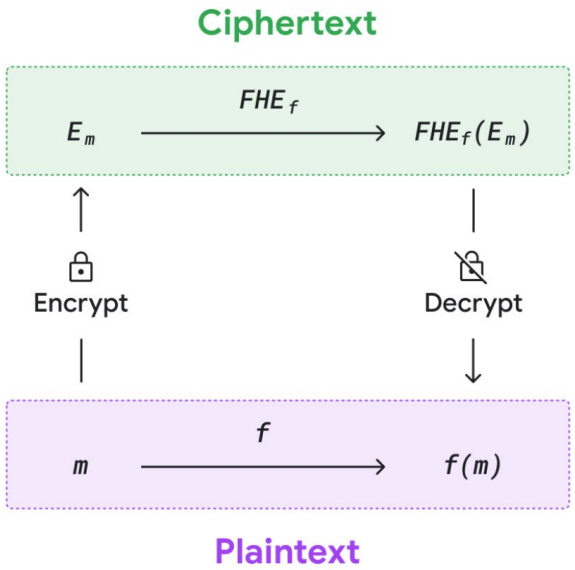
With FHE

Insider risk

Policy compliance

Data minimization

What is Fully Homomorphic Encryption

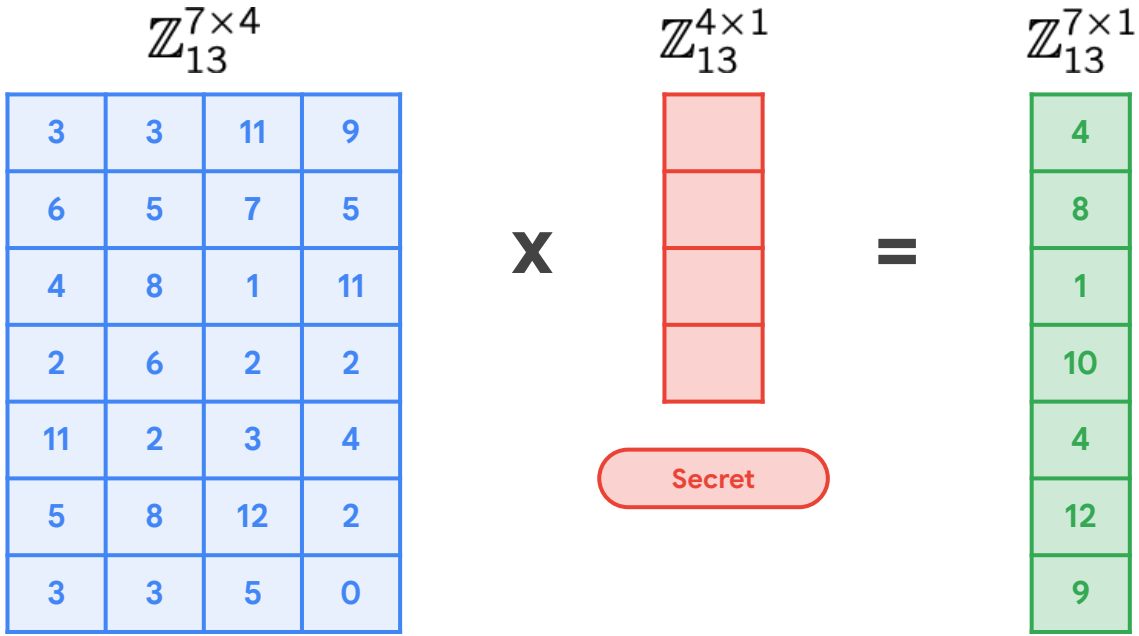


Problem

Learning with Errors (LWE)

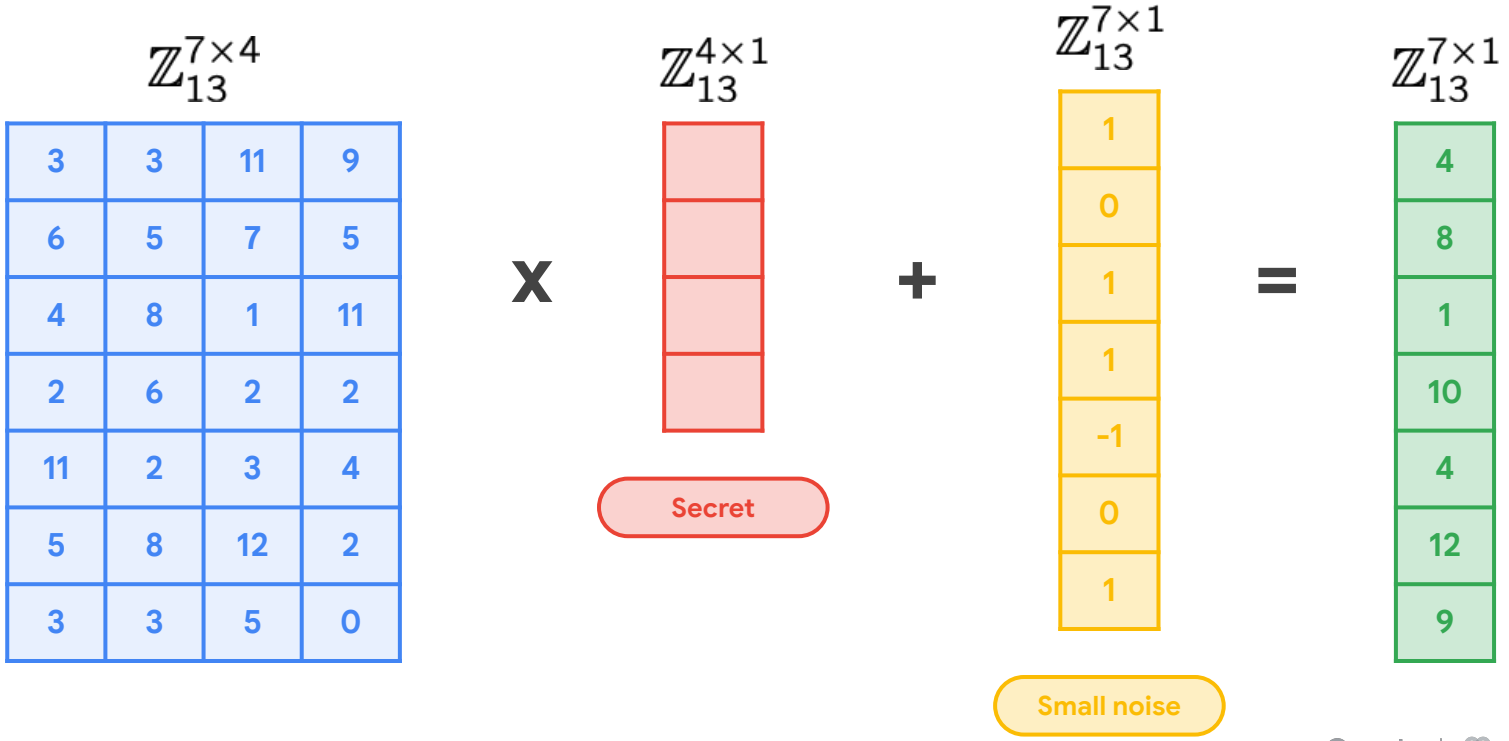
Solving systems of linear equations

Linear system problem: given blue and green, find red

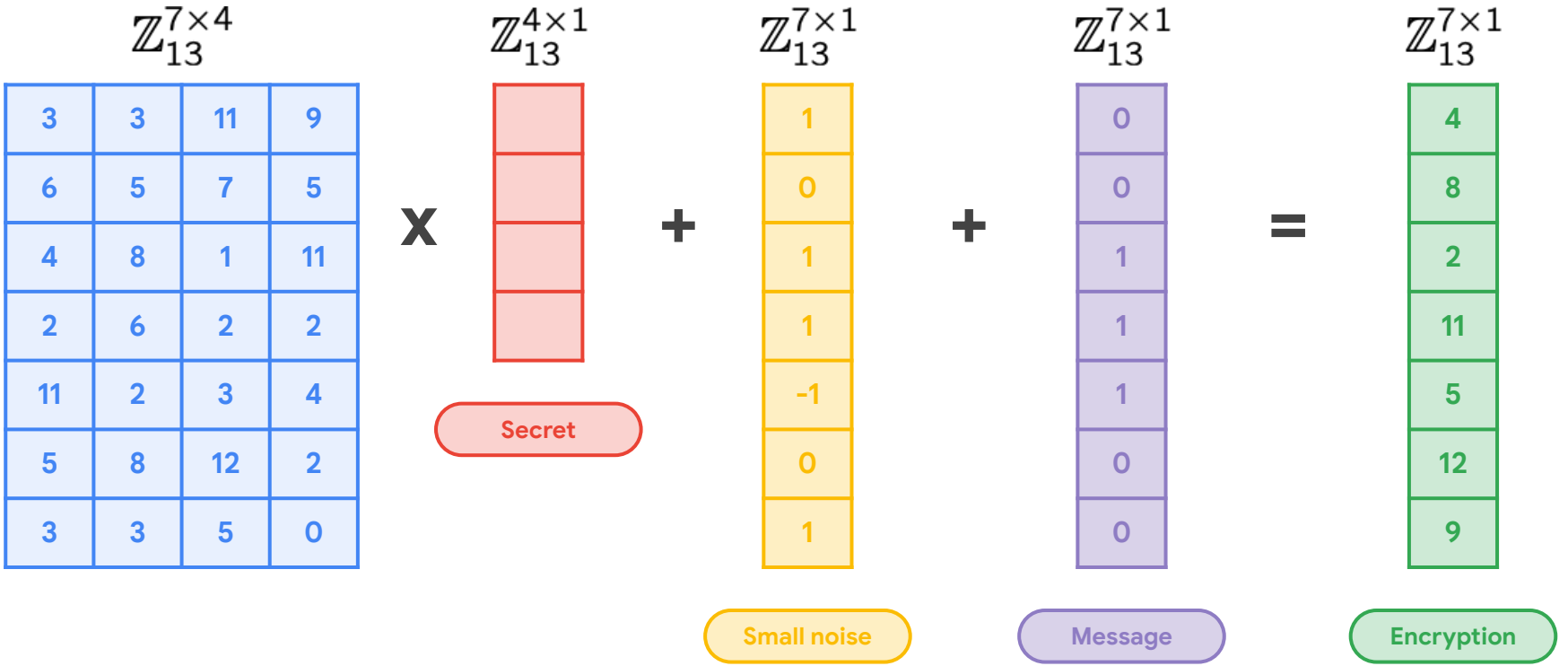


Learning with Errors (LWE) problem

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



Encrypting with LWE



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	...
...	...	...	...
...	...	...	...

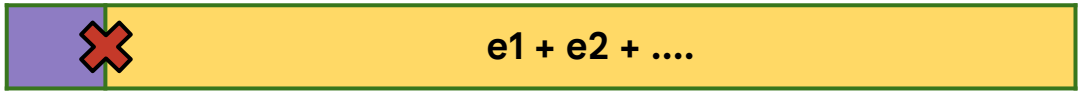
Rule of thumb (bit encryption): dimension more than 750, modulus 16+ bits.
Rule of thumb (homomorphic encryption): dimension >2000, modulus >60 bits.

FHE Computations and Noise

Decryption fails if error grows too large

Noise Management

Problem: Decryption fails if error grows too large



Solution: Choose **large parameters** to fit the entire computation

Solution : Track noise and perform **ciphertext-refresh operations**

Challenges

Usability

Speed

FHE Programming

Need to abstract away cryptographic details from engineers

Cryptography

Engineering

FHE Programming : Cryptographic challenges

01
Circuit depth analysis

02
Noise tracking

03
Bootstrapping

04
Parameter selection

FHE Programming challenges

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

```
if (condition) {
  return a;
} else {
  return b;
}
```

```
return
      condition.a +
      (1-condition).b;
```

FHE Programming : Engineering challenges

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

02 Loops need static upper bounds

03 No branch and bound optimizations

04 Using the right FHE scheme for the right type of computation.

2. FHE - A Brief History

FHE over the years

FHE is possible

30 mins for a single MUL

(Craig Gentry's Thesis)

1978 — **2009**

FHE over the years : Ring LWE

FHE is possible

30 mins for a single MUL

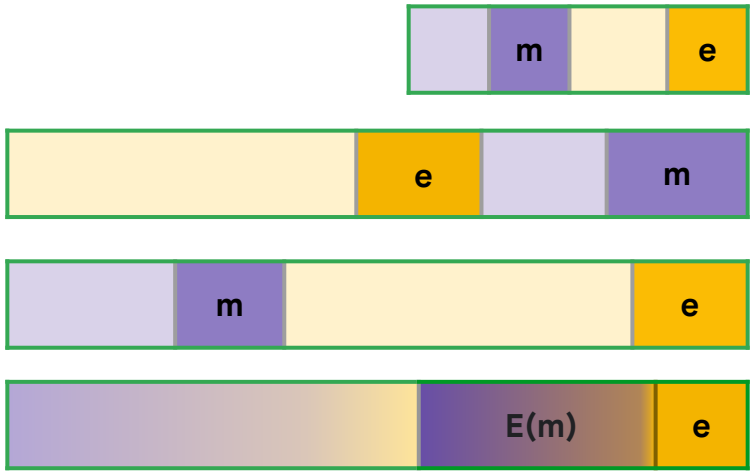
1978 — **2009** — 2016

FHE over the years : FHE Schemes

1978 — **2009** — 2016

FHE is possible

30 mins for a single MUL



TFHE / DM / CGGI / FHEW
Gates and LUT

BGV
Integer Arithmetic

BFV
Integer Arithmetic

CKKS
Approx fixed point arithmetic

FHE over the years

1978

2009

FHE is possible

30 mins for a single MUL

2016

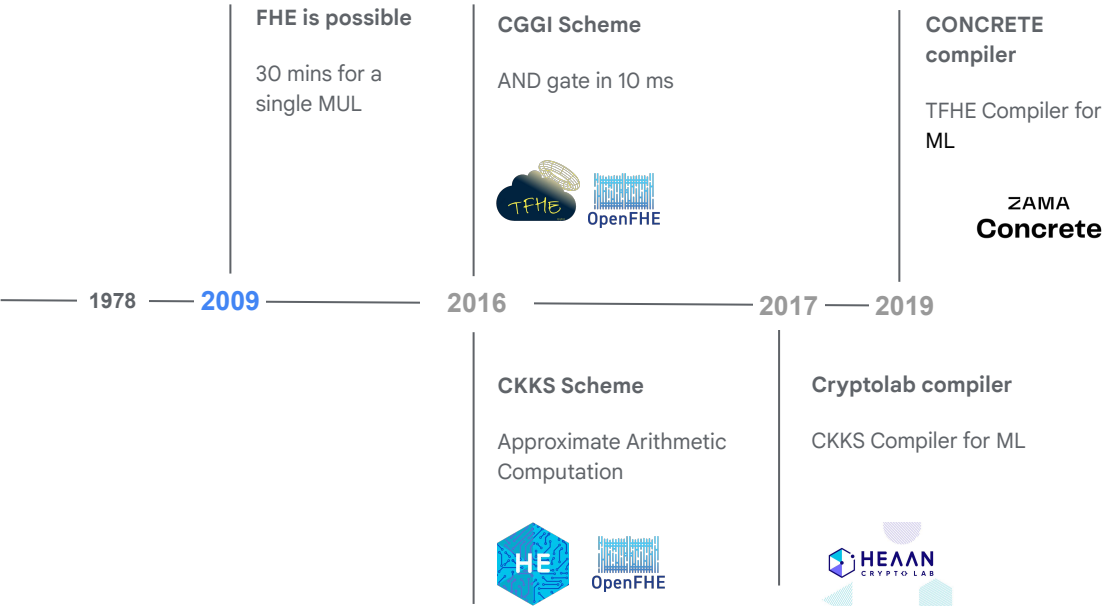
CGGI Scheme

AND gate in 10 ms

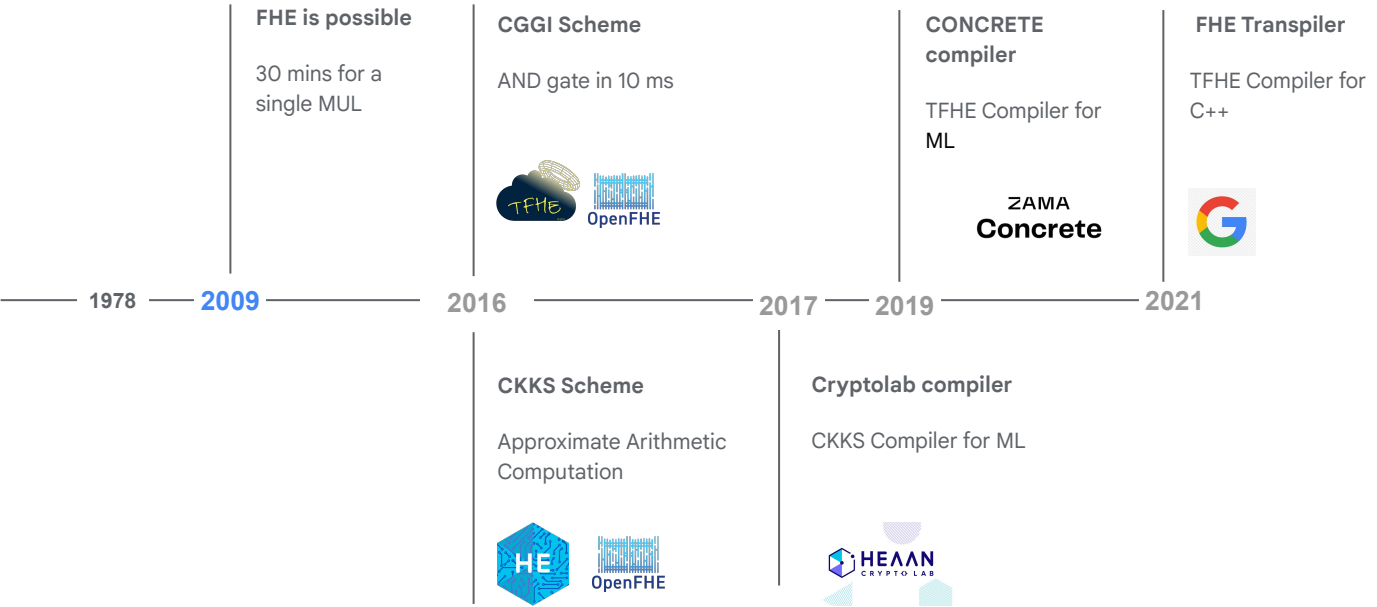
CKKS Scheme

Approximate Arithmetic Computation

FHE over the years



FHE over the years



FHE over the years

1978

2009 — **FHE is possible**

30 mins for a single MUL

2016 — **CGGI Scheme**

AND gate in 10 ms

2017 — **CKKS Scheme**

Approximate Arithmetic Computation

2019 — **CONCRETE compiler**

TFHE Compiler for ML

ZAMA
Concrete

Cryptolab compiler

CKKS Compiler for ML

2021 — **FHE Transpiler**

TFHE Compiler for C++

Video processing

2023 — **Hardware Accelerators**

FPT: Gates in 0.03 ms (FPGA)

Jaxite: FHE on CPU/GPU/TPU

HEIR

Foundation for FHE Compilers

FHE over the years

1978

2009

FHE is possible

30 mins for a single MUL

2016

CGGI Scheme

AND gate in 10 ms

CKKS Scheme

Approximate Arithmetic Computation

2017

Cryptolab compiler

CKKS Compiler for ML

2019

CONCRETE compiler

TFHE Compiler for ML

ZAMA
Concrete

2021

FHE Transpiler

TFHE Compiler for C++

HEIR

Foundation for FHE Compilers

2023

Hardware Accelerators

FPT: Gates in 0.03 ms (FPGA)

COSIC

Jaxite: FHE on CPU/GPU/TPU

Claim: 10000x speedup (Photonics)

Optalysys

2025 and beyond

Hardware Accelerators

Claim: 10000x speedup (ASIC)

galois | Duality intel

MIT Massachusetts Institute of Technology SRI International

FHE over the years

1978

2009
FHE is possible

30 mins for a single MUL

2016
CGGI Scheme

AND gate in 10 ms

2017
CKKS Scheme

Approximate Arithmetic Computation

2019
Cryptolab compiler

CKKS Compiler for ML

2021
CONCRETE compiler

TFHE Compiler for ML

ZAMA
Concrete

FHE Transpiler

TFHE Compiler for C++

HEIR

Foundation for FHE Compilers

2023
Hardware Accelerators

FPT: Gates in 0.03 ms (FPGA)

COSIC

Jaxite: FHE on CPU/GPU/TPU

Claim: 10000x speedup (Photonics)

Optalysys

2025 and beyond
Hardware Accelerators

Claim: 10000x speedup (ASIC)

galois | Duality intel

MIT Massachusetts Institute of Technology

SRI International

3. FHE Transpiler

github.com/google/fully-homomorphic-encryption

FHE transpiler

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

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

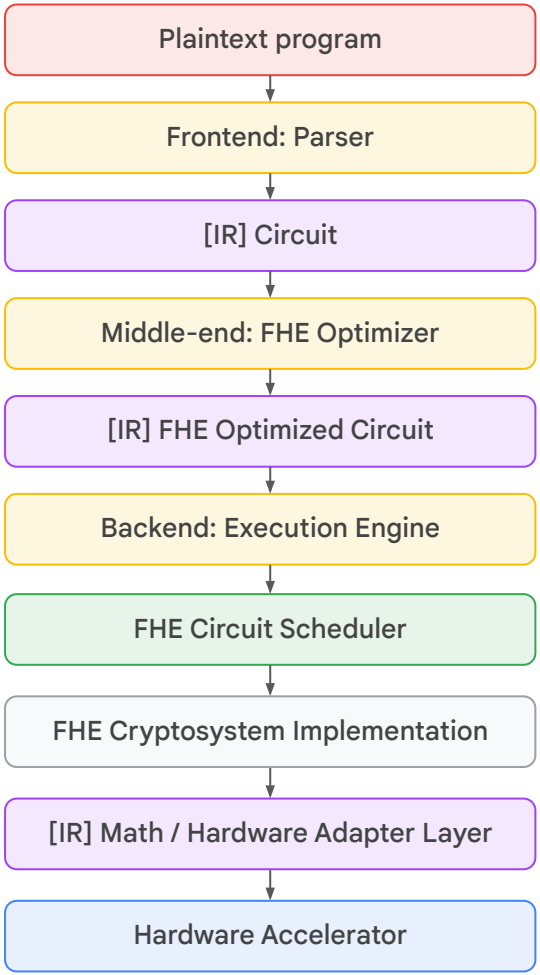
```
#include <tfhe.h>

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);
}
```

FHE Transpiler

FHE C++ Compilation

Modular
Interoperable
Reusable



Plaintext Program

String Capitalization

```
#include "string_cap.h"

#pragma hls_top
void CapitalizeString(char my_string[MAX_LENGTH]) {
 bool last_was_space = true;
#pragma hls_unroll yes
 for (int i = 0; i < MAX_LENGTH; i++) {
   char c = my_string[i];
   if (last_was_space && c >= 'a' && c <= 'z') {
     my_string[i] = c - ('a' - 'A');
   }
   last_was_space = (c == ' ');
 }
}
```

Parser : Data Independent Programming

XLScc : High Level Synthesis applied to FHE

01

No if/else

(well... sort of)

02

Loops need static upper bounds

03

No branch and bound optimizations

```
if (condition) {
  return a;
} else {
  return b;
}
```

```
return
      condition.a +
      (1-condition).b;
```

Circuit : XLS IR

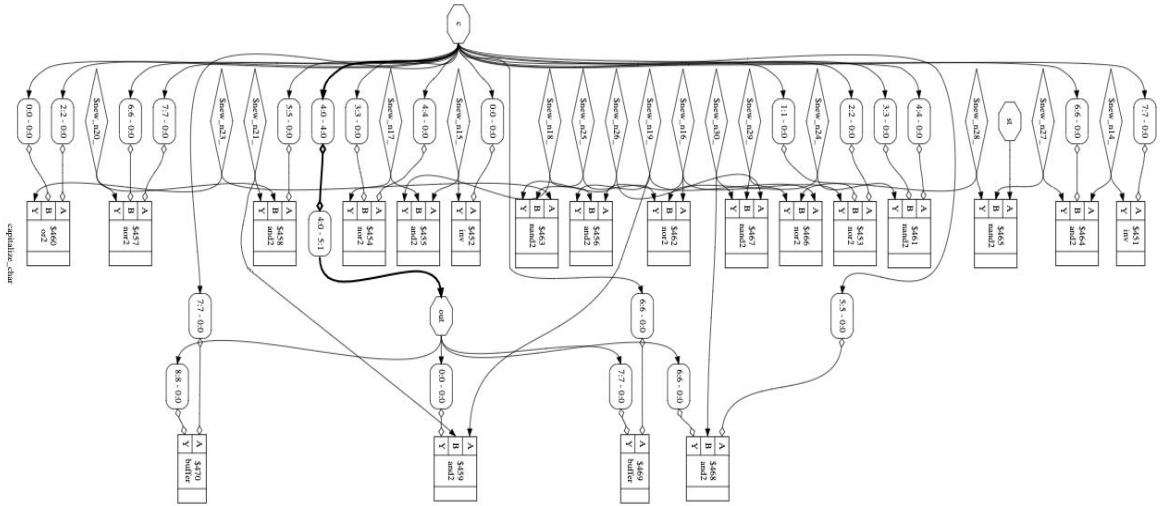
Intermediate Representations - Modularity & Interoperability

```
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)
}
```

FHE Optimizer : Netlist IR

Yosys ABC optimizations: reduces circuit size by 40 - 80%



Execution Engine : Transpiler Codegen

Server : Scheduler

```
#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);
  ...
  bootsAND(&temp_nodes[4], temp_nodes[461], temp_nodes[256], 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);
  }
}
```

Execution Engine : Transpiler Codegen

Client: Encryption/Decryption API

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

absl::Status SumSimpleStruct_SCHEDULER(absl::Span<LweSample> result,
       absl::Span<const LweSample> value,
       const TFheGateBootstrappingCloudKeySet* bk);

absl::Status SumSimpleStruct(TfheRef<signed int> result
       const TfheRef<SimpleStruct> value,
       const TFheGateBootstrappingCloudKeySet* bk) {
  return SumSimpleStruct_SCHEDULER(result.get(), value.get(), bk);
}
```

```
#include <tfhe_simple_struct>

Tfhe<SimpleStruct> fhe_simple_struct(SimpleStruct(2, 3, 4));
fhe_simple_struct.SetEncrypted(simple_struct, key);
```

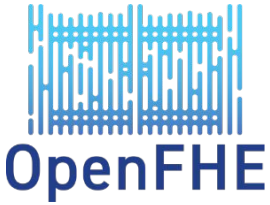
Circuit Schedulers

Computer Architecture & Distributed Systems

- Simple Single Threaded Scheduler
- Multi-core CPU: Multi threaded Scheduler
- GPU: SIMD Scheduler
- TPU: SPMD Scheduler
- Fleet: Distributed Scheduler?

FHE Library

Cryptosystem Library Implementation

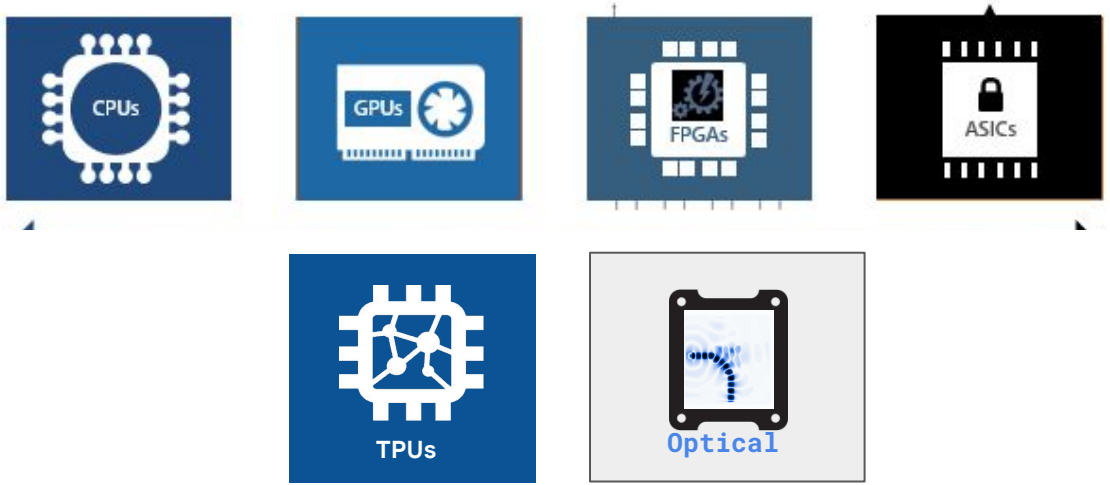


ZAMA
Concrete

Micro Instruction Scheduler

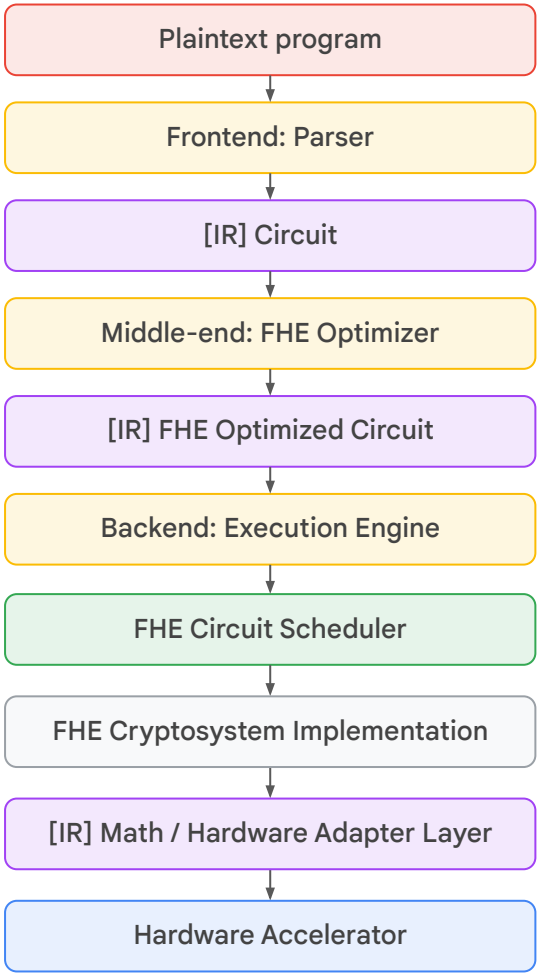
Computer Architecture & Hardware Co-design

- Accelerate Math/Crypto Primitives: Polynomial Math
- Benchmark across Hardware accelerators

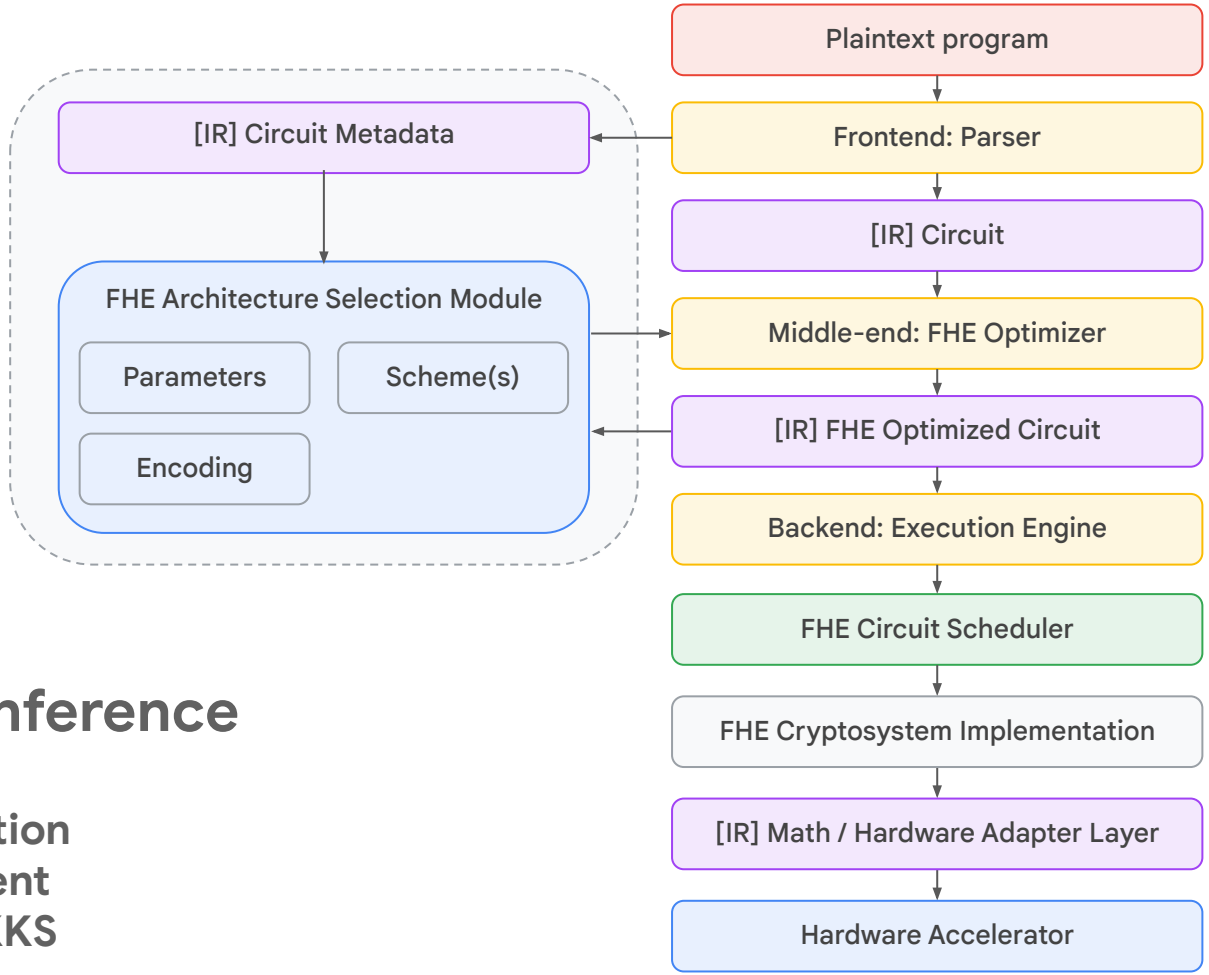


FHE Transpiler

C++ Compilation



FHE Stack



Private ML Inference

Parameter Selection
Noise Management
TFHE-LUTs or CKKS

Blocker to FHE Stack

Extensible Intermediate Representations

4. HEIR

Homomorphic Encryption Intermediate Representation

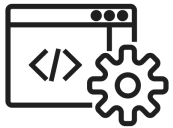
What is HEIR?

- Compiler toolchain for FHE (+ more!)
- Benchmarking platform
- Built in LLVM's MLIR framework

MLIR: Multi Level Intermediate Representation

A collection of ***modular and reusable*** software components that enables the ***progressive lowering*** of operations, to efficiently ***target hardware in a common way,*** *built at Google for TensorFlow.*

MLIR : A great fit for FHE

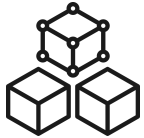


Multi-Level Abstraction

Shared abstractions and specialized expertise

Not Opinionated

Choose level of abstractions for different targets

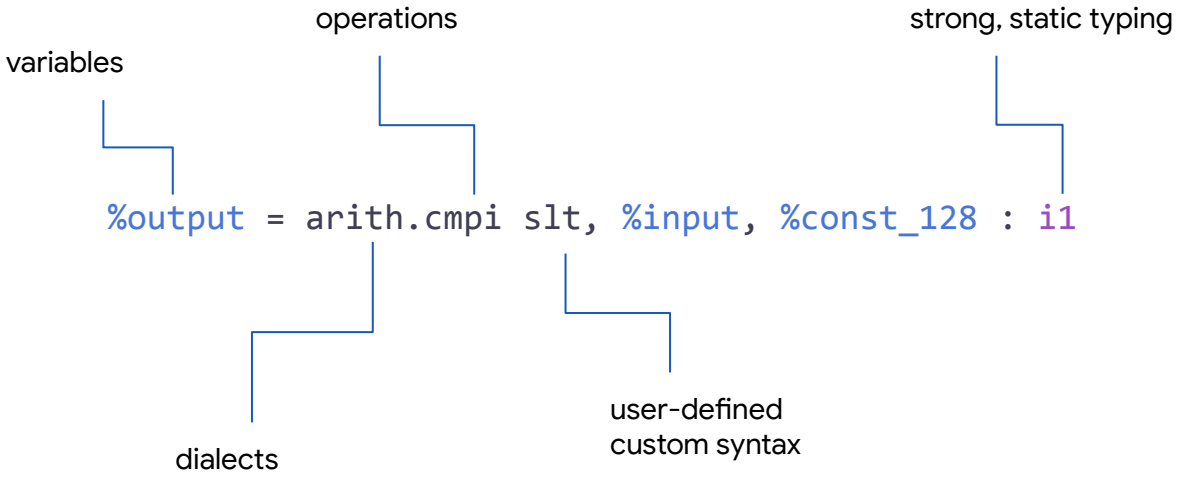


Modular & Extensible

Adapt to new research and mix and match

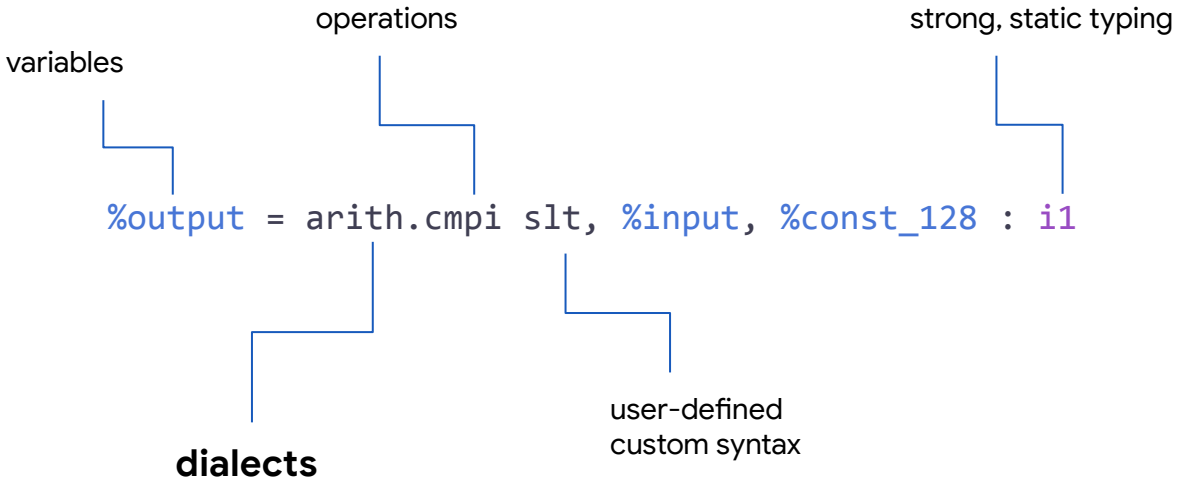
MLIR: Multi Level Intermediate Representation

Common Syntax, basic building blocks

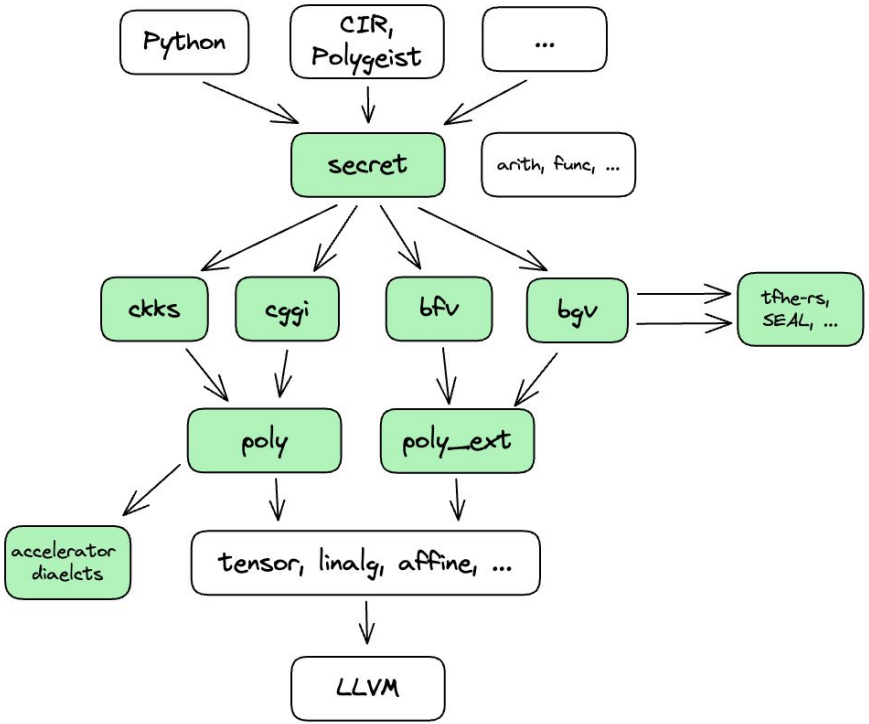


MLIR: Multi Level Intermediate Representation

Common Syntax, basic building blocks



What we're working on in HEIR



4.1 HEIR Goal

To build a compiler toolchain

with all the established ideas in FHE

to accelerate FHE research

and lower the barrier to production FHE

Using the MLIR framework

We're building a compiler toolchain

with all the established ideas in FHE

to accelerate FHE research

and lower the barrier to production FHE

Using the MLIR framework

We're building a compiler toolchain

Every FHE scheme, fast polynomial math, optimizations, noise models, ...

with all the established ideas in FHE

to accelerate FHE research

and lower the barrier to production FHE

Using the MLIR framework

We're building a compiler toolchain

Every FHE scheme, fast polynomial math, optimizations, noise models, ...

with all the established ideas in FHE

Focus on your novel idea, not compiler infra

to accelerate FHE research

and lower the barrier to production FHE

Using the MLIR framework

We're building a compiler toolchain

Every FHE scheme, fast polynomial math, optimizations, noise models, ...

with all the established ideas in FHE

Focus on your novel idea, not compiler infra

to accelerate FHE research

Benchmarking, hardware targets, frontends (C++, Python, Tensorflow)

and lower the barrier to production FHE

Working group!

- github.com/google/heir
- Open design meeting every two weeks
- Participants from across industry and academia
  - Bigco: Google, Intel
  - Startups: Zama, Cryptolab
  - University: ETH Zurich, KU Leuven
  - Hardware startups: Optalysys, Niobium
  - **you?**

Meeting calendar

galois | Duality | ZAMA | HEAAN CRYPTO LAB | Google

Thank you

google/heir

[Working Group]

https://google.github.io/heir/community