Building an End-to-End Toolchain for Fully Homomorphic Encryption with MLIR

Alexander Viand, Patrick Jattke, Miro Haller, Anwar Hithnawi
Cloud Computing

“Where the sensitive information is concentrated, that is where the spies will go. This is just a fact of life.”

former NSA official Ken Silva.
End-to-End Encrypted Systems
Modern Cryptography

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

End-to-End Security

- data at rest
- data in transit
- data in use?
Fully Homomorphic Encryption

$f(x) \leftarrow \text{Decrypt} \rightarrow \text{Enc}(f(x))$
Delegate the **processing** of data without giving away **access** to it.
Fully Homomorphic Encryption

Delegate the **processing** of data without giving away **access** to it
Delegating the processing of data without giving away access to it
Learning With Errors

\[ c = \sum_{i=1}^{n} a_i s_i + e \]

where \( a \in \mathbb{Z}_q^n, e \in \mathbb{Z}_q, s \in \mathbb{Z}_q^n \)
$Enc(0) \otimes Enc(1)$ \textbf{(Ring-)} Learning With Errors

$c = a \cdot s + e$

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

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

Gentry ‘09
$c = a \cdot s + e$

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

$R = \mathbb{Z}[X]/(X^n + 1)$
FHE: Theory to Practice

\[ \text{Enc}(0) \otimes \text{Enc}(1) \quad (\text{Ring-}) \text{Learning With Errors} \]

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

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

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

Gentry ‘09

FHE’14, TFHE ‘16

Arithmetic FHE

BGV ‘12, B/FV ‘12, CKKS ‘16

Binary FHE

\[ \sim 1 \text{ OoM more operations} \]

Figure from CryptoDL: Deep Neural Networks over Encrypted Data [HTG17]
FHE: Theory to Practice

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

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

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

Gentry ‘09

\[ \text{Enc}(0) \otimes \text{Enc}(1) \] (Ring)-Learning With Errors

Binary FHE

\[ \sim 1 \text{ OoM more operations} \]

FHEW’14, TFHE ‘16

FHE’12, B/FV ‘12, CKKS ‘16

Arithmetic FHE

Hardware Acceleration

BGV ‘12, B/FV ‘12, CKKS ‘16

Figure from CryptoDL: Deep Neural Networks over Encrypted Data [HTG17]
Evolution of FHE Tools

```
void f(...) {
    mul_inp(a,b);
    relin_inp(a);
    add_plain_inp(a,3)
    square_inp(z,z);
    relin_inp(a);
    sub_inp(a,z);
    return a;
}
```
Evolution of FHE Tools

```c
void f(...) {
    ctxt ab = a*b + 3;
    return ab - z*z;
}
```
Evolution of FHE Tools

```c
void f(...)
{
    ctxt ab = a*b + 3;
    return ab - z*z;
}
```
Existing tools make important contributions, but are too narrowly focussed
void hd(vector<bool> u, vector<bool> v) {
  s int sum = 0;
  for (int i = 0; i < v.size(); ++i) {
    sum += (v[i] != u[i]);
  }
}
End-to-End FHE Toolchain

Computer Program

FHE Compiler

Arithmetic Circuit

```c
void hd(vector<bool> u, vector<bool> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
    {
        sum += (v[i] != u[i]);
    }
}
```
void hd(vector<bool> u, vector<bool> v) {
    int sum = 0;
    for (int i = 0; i < v.size(); ++i) {
        sum += (v[i] != u[i]);
    }
}
End-to-End FHE Toolchain

Computer Program → FHE Compiler → Arithmetic Circuit

Loss of Information
End-to-End FHE Toolchain

```
void hd(vector<bool> u, vector<bool> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
    {
        sum += (v[i] != u[i]);
    }
}
```
End-to-End FHE Toolchain

Computer Program

FHE Compiler

Secure and Efficient FHE Solutions

void hd(vector<bool> u, vector<bool> v) {
    int sum = 0;
    for (int i = 0; i < v.size(); ++i) {
        sum += (v[i] != u[i]);
    }
}
End-to-End FHE Toolchain

void hd(vector<int> u, vector<int> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
        sum += (v[i] != u[i]);
}
End-to-End FHE Toolchain

void hd(vector<bool> u, vector<bool> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
    {
        sum += (v[i] != u[i]);
    }
}
End-to-End FHE Toolchain

Computer Program

FHE Compiler

Secure and Efficient FHE Solutions

void hd(vector<bool> u, vector<bool> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
    {
        sum += (v[i] != u[i]);
    }
}
End-to-End FHE Toolchain

Computer Program

FHE Compiler

Secure and Efficient FHE Solutions

```c
void hd(vector<bool> u, vector<bool> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
        sum += (v[i] != u[i]);
}
```
End-to-End FHE Toolchain

Computer Program → FHE Compiler → Secure and Efficient FHE Solutions

Computer Program

FHE Compiler

Secure and Efficient FHE Solutions

void hd(vector<bool> u, vector<bool> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
    {
        sum += (v[i] != u[i]);
    }
}
End-to-End FHE Toolchain

Computer Program → FHE Compiler → Secure and Efficient FHE Solutions

FHE Compiler

- **Prog Opt.**
- **Circuit Opt.**
- **Crypto Opt.**
- **Target Opt.**

**FHE Compiler**

**Computer Program**

```cpp
void hd(vector<bool> u, vector<bool> v)
{
    int sum = 0;
    for (int i = 0; i < v.size(); ++i)
    {
        sum += (v[i] != u[i]);
    }
}
```
HECO Architecture

Python
... other frontends
C-like DSL

HeCo
::fhe
// custom MLIR Dialect
Types
secret<T>
batchedsecret<T>
Operations
+
-
*
+/=
mul
rot

mlir::tensor
// builtin MLIR Dialect
Types
tensor<V>
// e.g., V = secret<T>
Operations
+
-
*
+/=
mlir::arithmetic
// iadd, imul, ...

HeCo
::bfv
// ops in bfv scheme

HeCo
::bgv
// ops in bgv scheme

HeCo
::ckks
// ops in ckks scheme

HeCo
::poly
// ops in polynomial ring + number theoretic transform

HeCo
::rns
// ops in residue number system representation

mlir::emitc
// integrate legacy libs

C++

FHE Libraries
(e.g. SEAL)

mlir::llvm
// MLIR interface to LLVM IR
dprive::isa
// hardware ISA

Junyoung Lee, Prototyping a compiler for homomorphic encryption in MLIR, EuroLLVM'22
HECO: Modular End-to-End Design

Computer Program → FHE Compiler → Secure and Efficient FHE Solutions

Abstractions

Automated Batching

FHE Schemes Standardization
- Γ
- Λ

FHE Intermediate Representation Standardization
- 2018 draft API standard not adopted/implemented
- Significant FHE compiler efforts are accelerating
  - Need to re-visit standardization of abstractions
  - Expand beyond “FHE API” abstraction
SIMD-like Parallelism

Standard C++

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

Batched FHE

```cpp
int foo(int[] a, int[] b) {
    return a * b;
}
```

No efficient free permutation or scatter/gather
SIMD-like Parallelism

**Standard C++**

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

**Batched FHE**

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

- **SIMD Batching**
- **No efficient free permutation or scatter/gather**
- **Only cyclical rotations**
shuffle

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

return r;

int foo(int[] a, int[] b) {
    return a * b;
}

SIMD-like Parallelism

Standard C++

Batched FHE

Only cyclical rotations

No efficient free permutation or scatter/gather