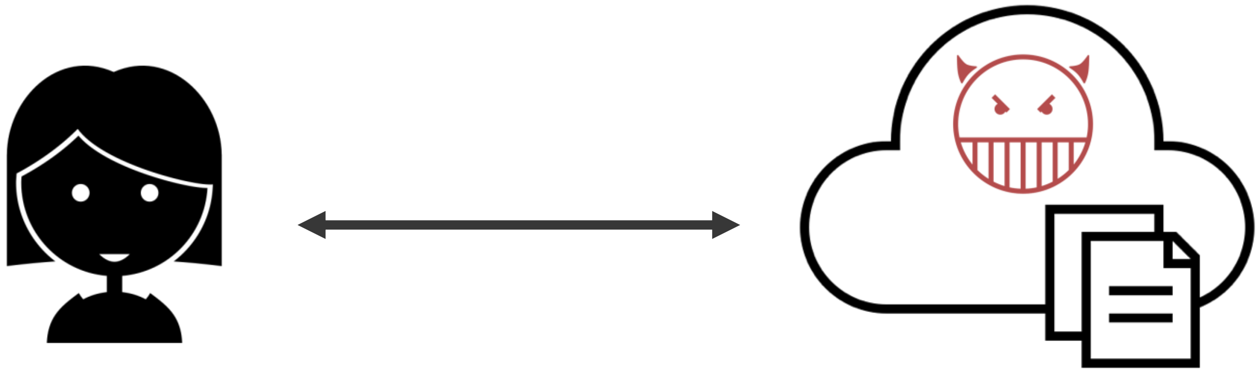
Building an End-to-End Toolchain for Fully Homomorphic Encryption with MLIR

Alexander Viand, Patrick Jattke, Miro Haller, Anwar Hithnawi

ETH zürich

Cloud Computing

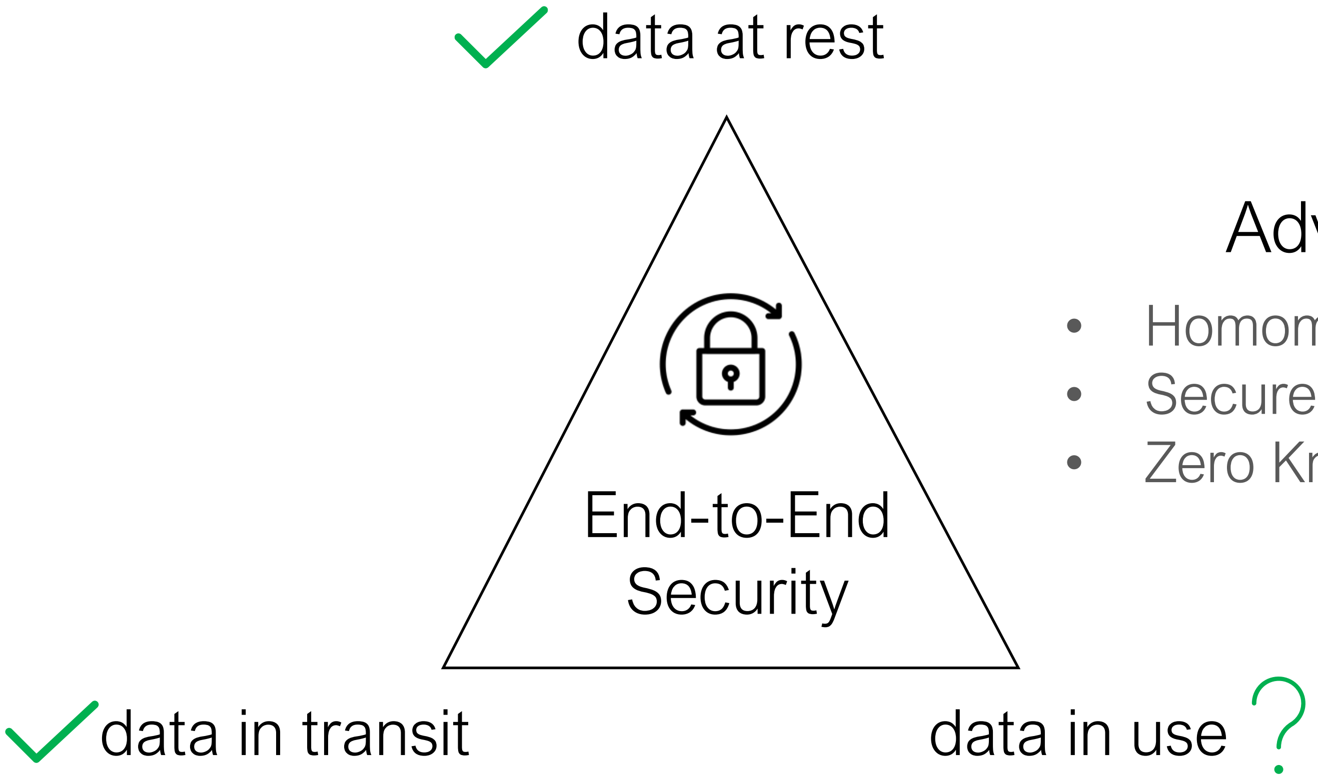


“Where the sensitive information is concentrated, that is where the spies will go. This is just a fact of life.”
former NSA official Ken Silva.

Software Vulnerabilities | Insider Threats | Physical Attacks

End-to-End Encrypted Systems

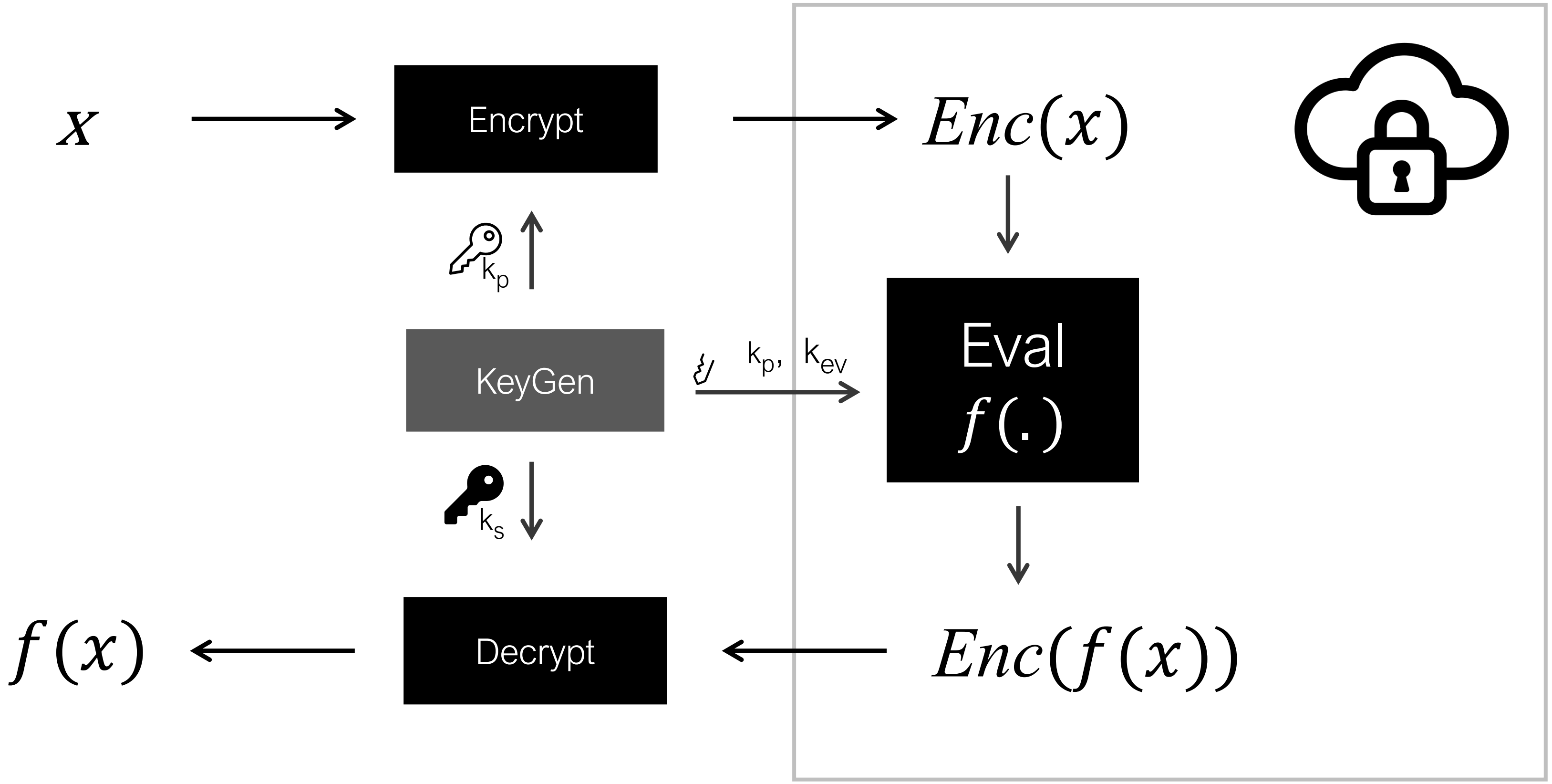
Modern Cryptography



Advanced Crypto

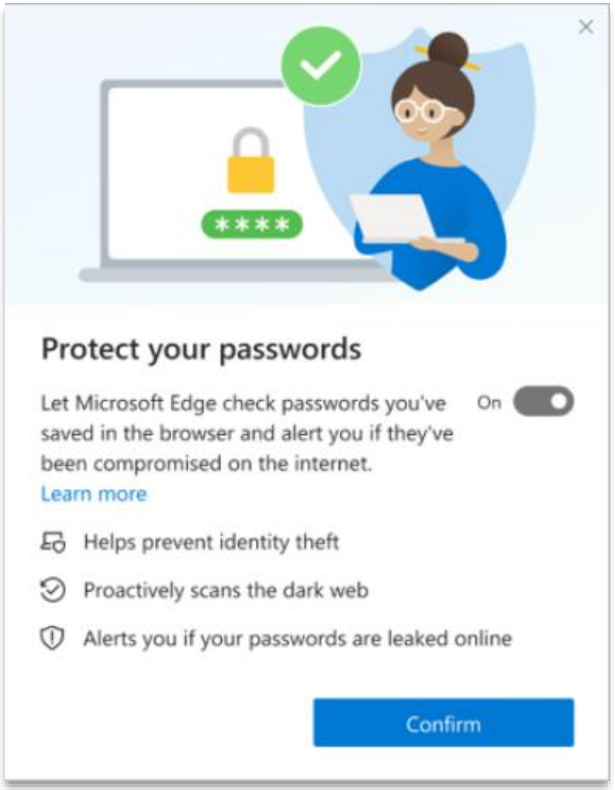
- Homomorphic Encryption
- Secure Multi-party Computation
- Zero Knowledge Proofs

Fully Homomorphic Encryption



Computing on encrypted data

Fully Homomorphic Encryption



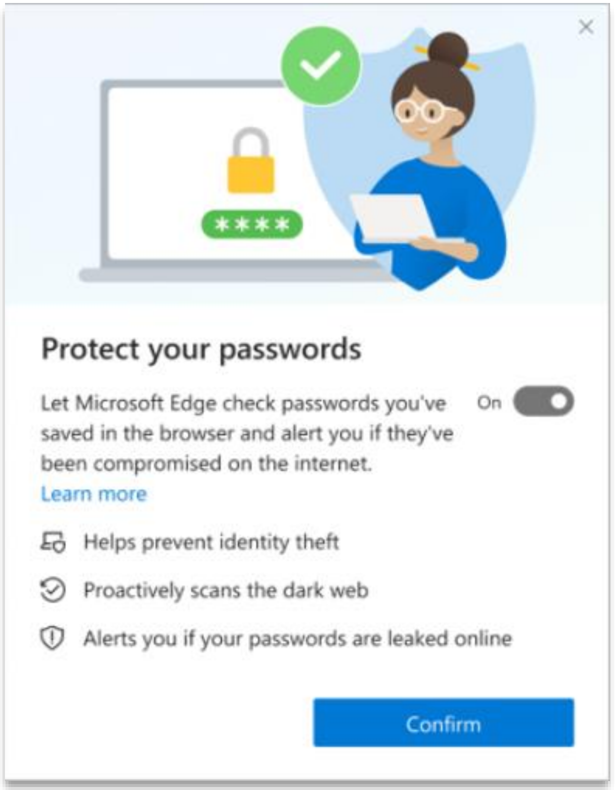
Alice91 | 0791 ****
A.Sample | 1234 ****
Bob | b4dp455 ****
Alice | 12345 ****

Microsoft Azure

Alice | 12345
Bob | wordpass
Eve | pa55word
Joe | correcth..
⋮
Steve | hunter2

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

Fully Homomorphic Encryption

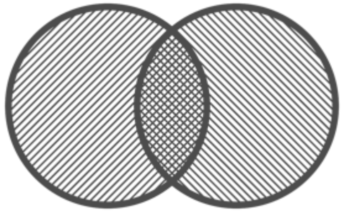


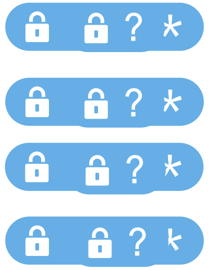
Alice91 | 0791

A.Sample | 1234

Bob | b4dp455

Alice | 12345





Alice | 12345

Bob | wordpass

Eve | pa55word

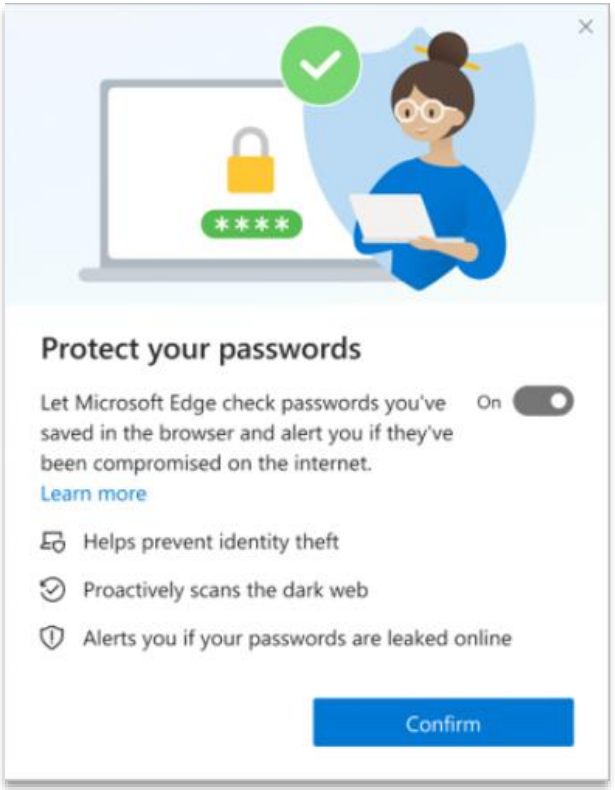
Joe | correcth..

⋮

Steve | hunter2

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

Fully Homomorphic Encryption



Microsoft Azure

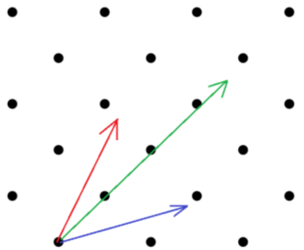
- Alice91 | 0791
- A.Sample | 1234
- Bob | b4dp455
- Alice | 12345

- Alice | 12345
- Bob | wordpass
- Eve | pa55word
- Joe | correcth..
- ⋮
- Steve | hunter2

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

FHE: Theory to Practice

$Enc(0) \circledcirc Enc(1)$

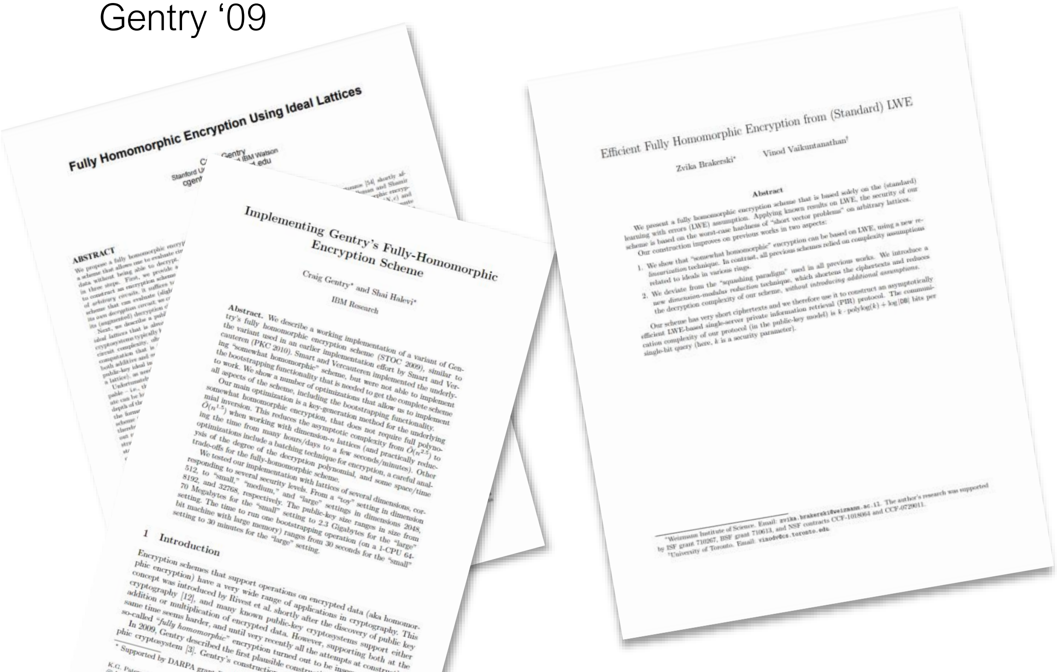


Learning **W**ith **E**rrors

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

$$where\ \boldsymbol{a} \overset{\$}{\leftarrow} \mathbb{Z}_q^n, \boldsymbol{e} \overset{\chi}{\leftarrow} \mathbb{Z}_q, \boldsymbol{s} \in \mathbb{Z}_q^n$$

Gentry '09



Fully Homomorphic Encryption Using Ideal Lattices

Implementing Gentry's Fully-Homomorphic Encryption Scheme

Craig Gentry and Shai Halevi

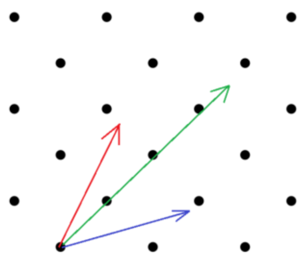
IBM Research

Efficient Fully Homomorphic Encryption from (Standard) LWE

Zvika Brakerski Vinod Vaikuntanathan

FHE: Theory to Practice

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



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

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

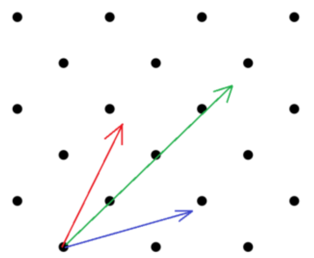
$where\ \boldsymbol{a} \overset{\$}{\leftarrow} R_q, \boldsymbol{e} \overset{\chi}{\leftarrow} R_q, \boldsymbol{s} \in R_q$

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

Gentry '09

FHE: Theory to Practice

$Enc(0) \circledcirc Enc(1)$



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

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

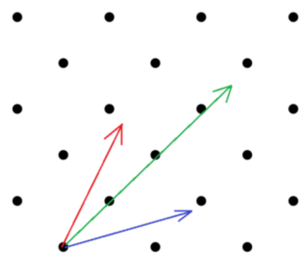
$where\ \boldsymbol{a} \xleftarrow{\$} R_q, \boldsymbol{e} \xleftarrow{\chi} R_q, \boldsymbol{s} \in R_q$

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

Gentry '09

FHE: Theory to Practice

$Enc(0) \circledcirc Enc(1)$ **(R**ing-) **L**earning **W**ith **E**rrors



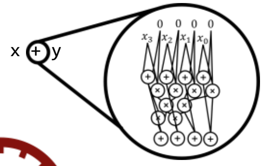
$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}$$

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

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

Gentry '09

Binary FHE

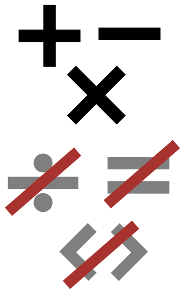


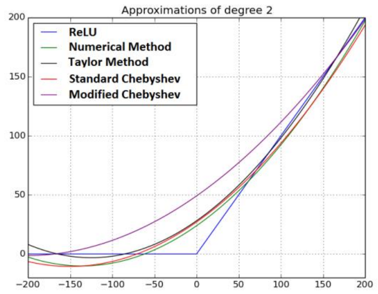
~1 OoM more operations

FHEW'14, TFHE '16

BGV '12, B/FV '12, CKKS '16

Arithmetic FHE



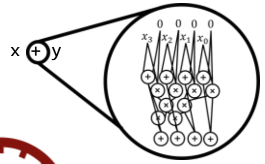


(a) Approximation of ReLU using different methods

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

FHE: Theory to Practice

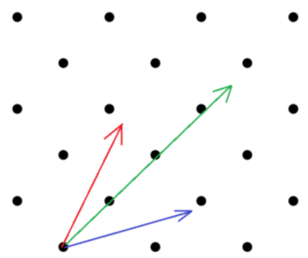
Binary FHE



~1 OoM more operations

FHEW'14, TFHE '16

$Enc(0) \circledcirc Enc(1)$



(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

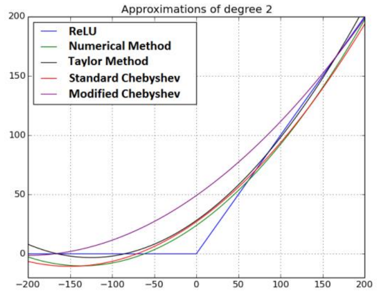
Hardware Acceleration

FHE

intel. Duality galois SRI International

BGV '12, B/FV '12, CKKS '16

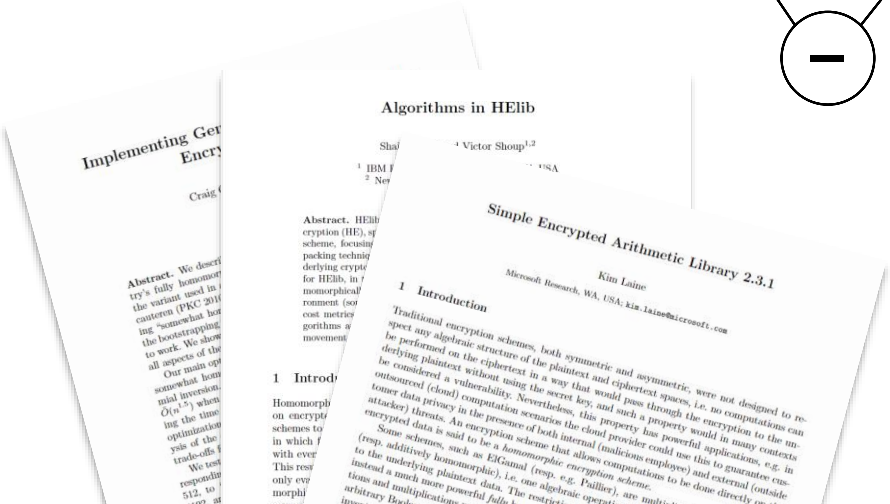
Arithmetic FHE

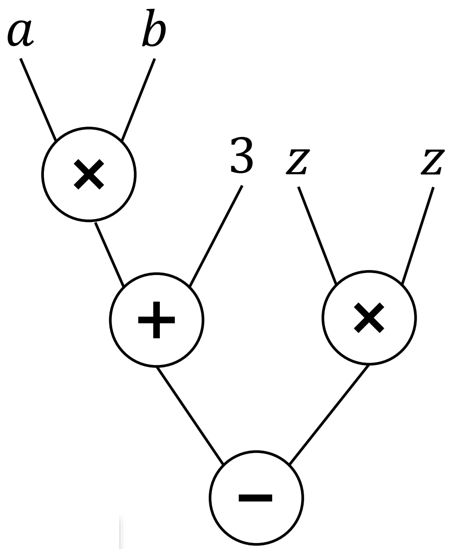


(a) Approximation of ReLU using different methods

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

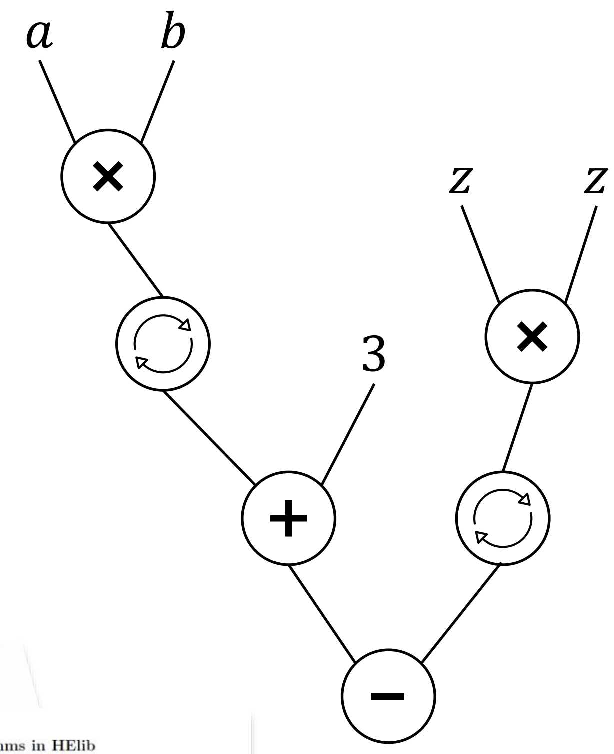
Evolution of FHE Tools



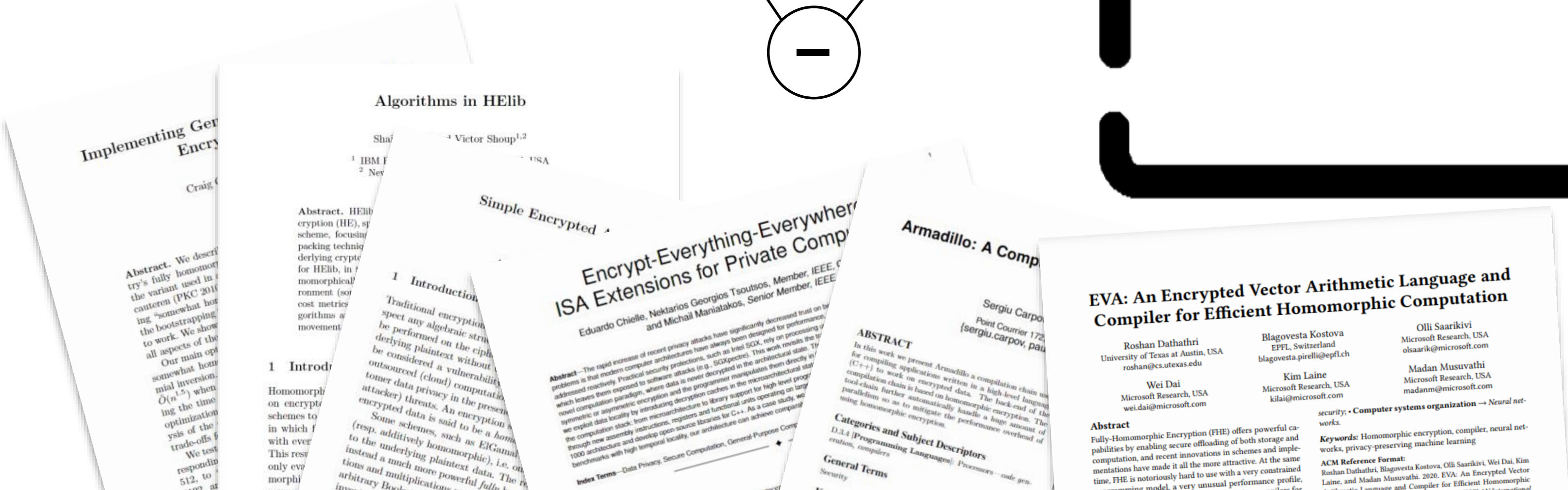



```
void f(...)
{
 mul_inp(a,b);
 relin_inp(a);
 add_plain_inp(a,3)
 square_inp(z,z);
 relin_inp(a);
 sub_inp(a,z);
 return a;
}
```

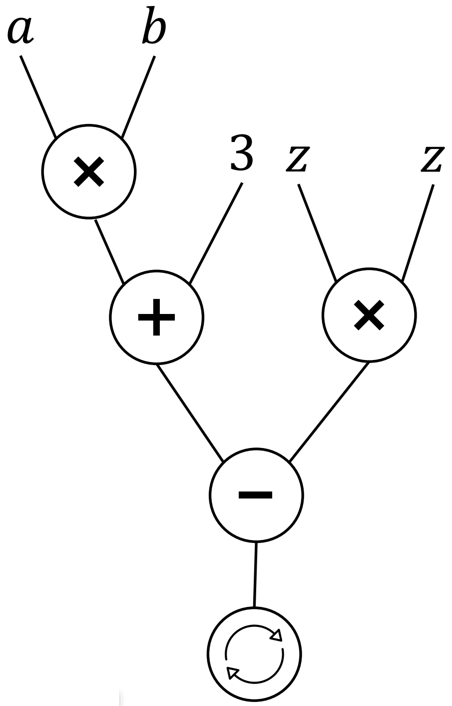
Evolution of FHE Tools



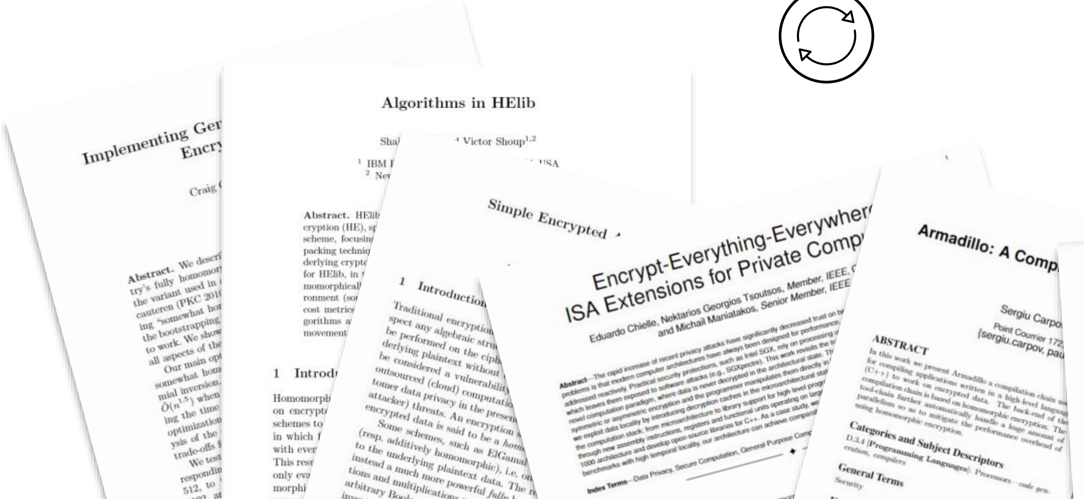

```
void f(...)
{
 ctxt ab = a*b + 3;
 return ab – z*z;
}
```

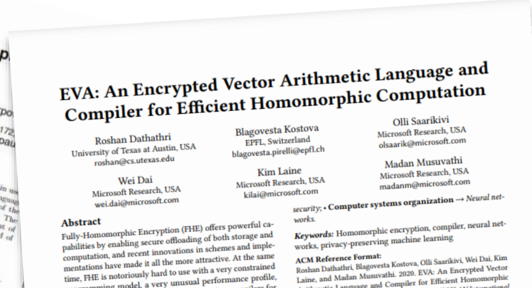


Evolution of FHE Tools




```
void f(...)
{
 ctxt ab = a*b + 3;
 return ab – z*z;
}
```





EVA: An Encrypted Vector Arithmetic Language and Compiler for Efficient Homomorphic Computation

Roshan Dathathri, University of Texas at Austin, USA, roshan@cs.utexas.edu
Blagovesta Kostova, EPFL, Switzerland, blagovesta.pirelli@epfl.ch
Olli Saarikivi, Microsoft Research, USA, olsaarik@microsoft.com
Wei Dai, Microsoft Research, USA, wei.dai@microsoft.com
Kim Laine, Microsoft Research, USA, kilai@microsoft.com
Madan Musuvathi, Microsoft Research, USA, madanm@microsoft.com

SoK: Fully Homomorphic Encryption Compilers

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

Abstract—Fully Homomorphic Encryption (FHE) allows a third party to perform arbitrary computations on encrypted data, learning neither the inputs nor the computation results. Hence, it provides resilience in situations where computations are carried out by an untrusted or potentially compromised party. This powerful concept was first conceived by Rivest et al. in the 1970s. However, it remained unrealized until Craig Gentry presented the first feasible FHE scheme in 2009.

The advent of the massive collection of sensitive data in cloud services, coupled with a plague of data breaches, moved highly regulated businesses to increasingly demand confidential and secure computing solutions. This demand, in turn, has led to a recent surge in the development of FHE tools. To understand the landscape of recent FHE tool developments, we conduct an extensive survey and experimental evaluation to explore the current state of the art and identify areas for future development.

In this paper, we survey, evaluate, and systematize FHE tools and compilers. We perform experiments to evaluate these tools' performance and usability aspects on a variety of applications. We conclude with recommendations for developers intending to develop FHE-based applications and a discussion on future directions for FHE tools development.

I. Introduction

Recent years have seen unprecedented growth in the adoption of cloud computing services. More and more highly regulated businesses and organizations (e.g., banks, governments, insurances, health), where data security is paramount, move their data and services to the cloud. This trend has led to a surge in demand for secure and confidential computing solutions that protect data confidentiality while in transit, rest, and in-use. This is an amply justified and expected demand, particularly in the light of the numerous reports of data breaches [1], [2]. Fully Homomorphic Encryption (FHE) is a key technological enabler for secure computation and has recently matured to be practical for real-world use [3]–[9].

FHE allows arbitrary computations to be performed over encrypted data, eliminating the need to decrypt the data and expose it to potential risk while in use. While first proposed in the 1970s [10], FHE was long considered impossible or impractical. However, thanks to advances in the underlying theory, general hardware improvements, and more efficient implementations, it has become increasingly practical. In 2009, breakthrough work from Craig Gentry proposed the first feasible FHE scheme [11]. In the last decade, FHE has gone from a theoretical concept to reality, with performance improving by up to five orders of magnitude. For example, times for a multiplication between ciphertexts dropped from 30 minutes to less than 20 milliseconds. While this is still around seven orders of magnitude slower than an IMUL instruction on a modern CPU, it is sufficient to make many applications practical. Additionally, modern schemes introduced SIMD-style parallelism, encoding thousands of plaintext values into a single ciphertext to further improve throughput [12].

These advances have enabled a wide range of applications covering a wide range of domains. These include mobile applications, where FHE has been used to encrypt the back end of a privacy-preserving fitness app [13], while continuing to provide a real-time experience. In the medical domain, FHE has been used to enable privacy-preserving genome analysis [14] applications over large datasets. More generally, FHE has been used to solve various well-known problems like Private Set Intersection (PSI) [15], outperforming previous solutions by 2× in running time. In the domain of machine learning, FHE has been used for tasks ranging from linear and logistic regression [16] to Encrypted Neural Network inference [17], which can be used to run privacy-preserving ML-as-a-Service applications, for example, for private phishing email detection [18]. As a consequence, there has been increasing interest in FHE-based secure computation solutions [3]–[9]. Gartner projects [19] that "by 2025, at least 20% of companies will have a budget for projects that include fully homomorphic encryption."

Despite these recent breakthroughs, building secure and efficient FHE-based applications remains a challenging task. This is largely attributed to the differences between traditional programming paradigms and FHE's computation model, which poses unique challenges. For example, virtually all standard programming paradigms rely on data-dependent branching, e.g., if/else statements and loops. On the other hand, FHE computations are, by definition, data-independent, or they would violate the privacy guarantees. Working with FHE also introduces significant engineering challenges in practice. Different schemes offer varying performance tradeoffs, and optimal choices are heavily application-dependent. To address some of the engineering challenges in this space, we have seen a surge of work on tools that aim to improve accessibility and reduce barriers to entry in this field.

Without tool support, realizing FHE-based computations by implementing the required mathematical operations directly or using an arbitrary-precision arithmetic library is complex, requiring considerable expertise in both cryptography and high-performance numerical computation. Therefore, FHE libraries like the Simple Encrypted Arithmetic Library (SEAL) [20] or the Fast Fully Homomorphic Encryption Library over the

1

S&P 2021, arXiv 2101.07078

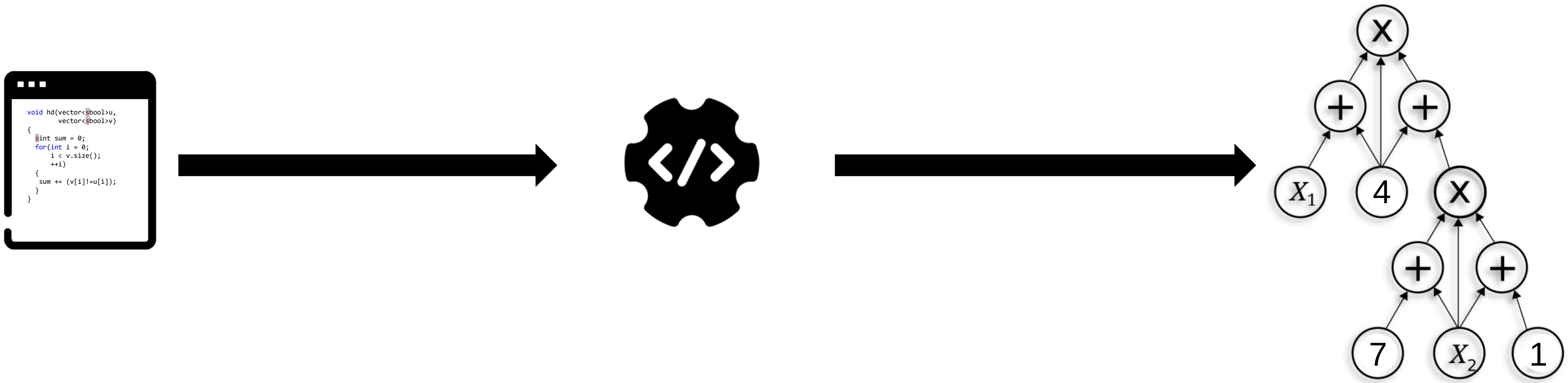
Existing tools make important contributions,
but are too **narrowly focussed**

End-to-End FHE Toolchain

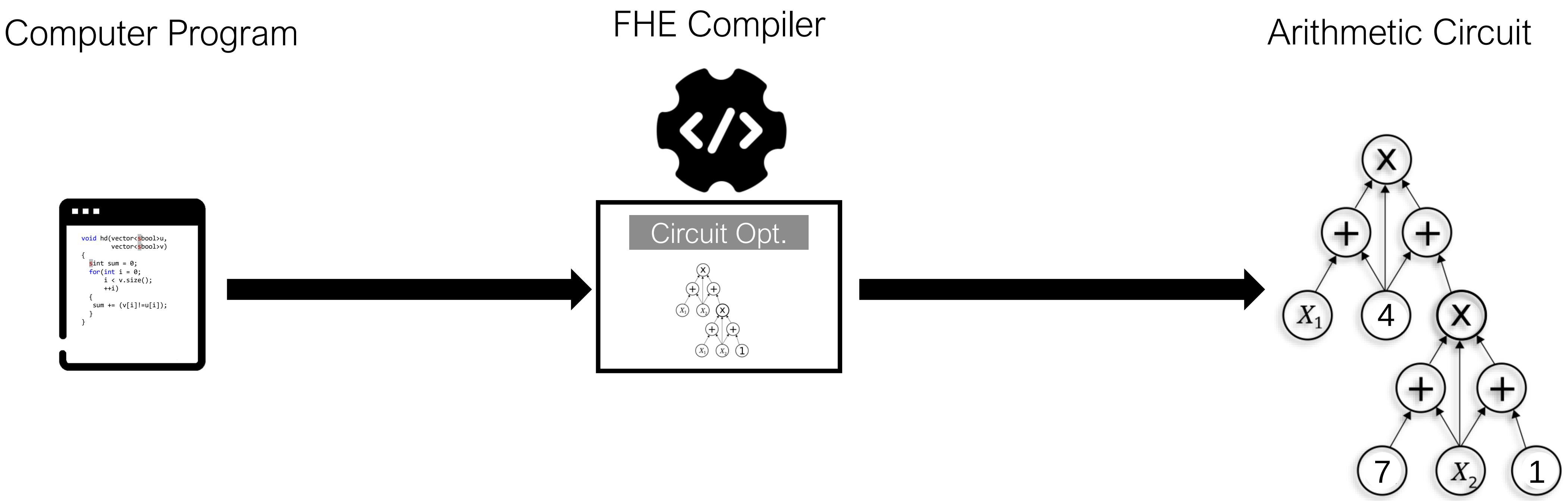
Computer Program

FHE Compiler

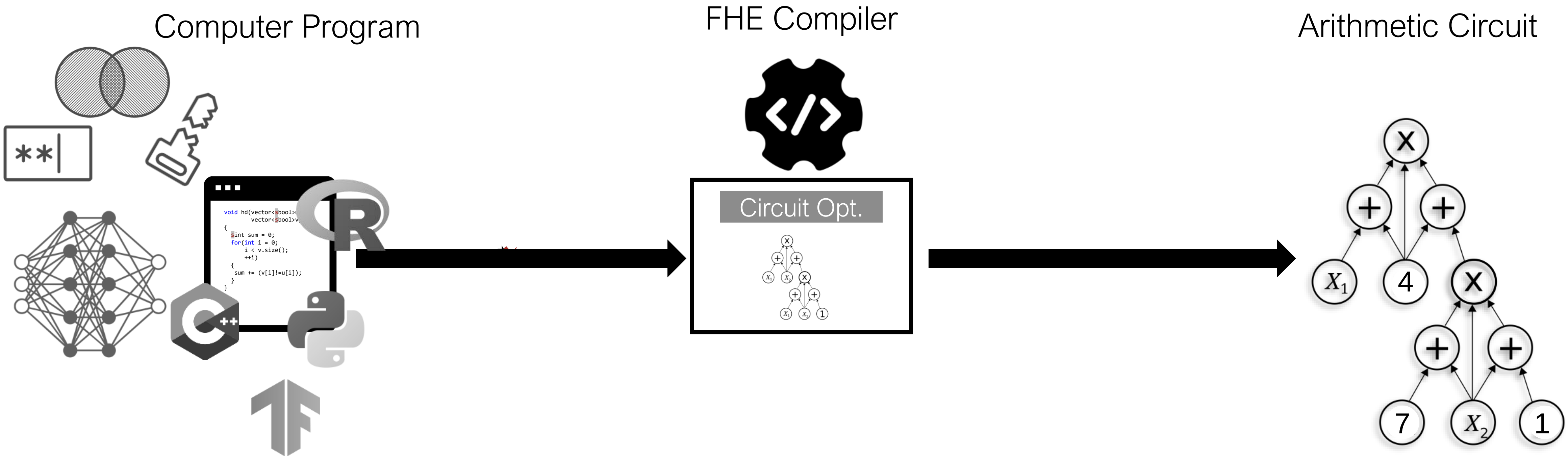
Arithmetic Circuit



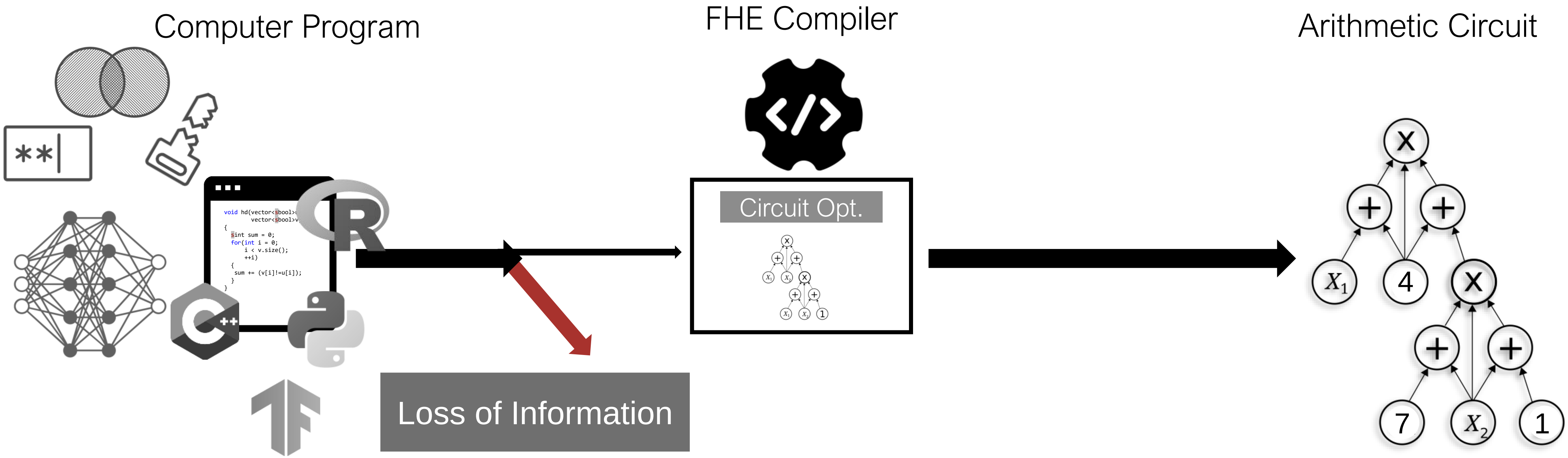
End-to-End FHE Toolchain



End-to-End FHE Toolchain

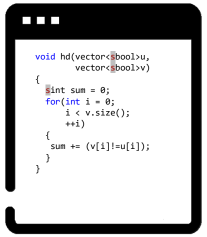


End-to-End FHE Toolchain

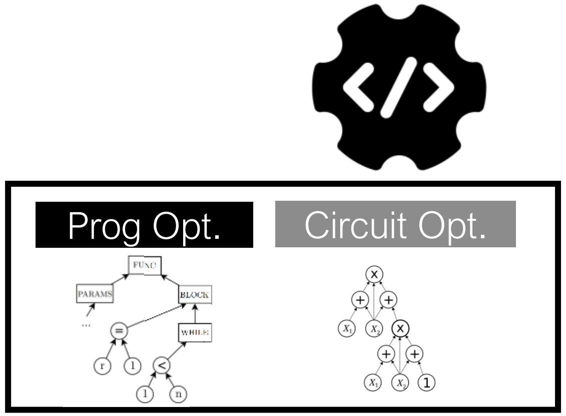


End-to-End FHE Toolchain

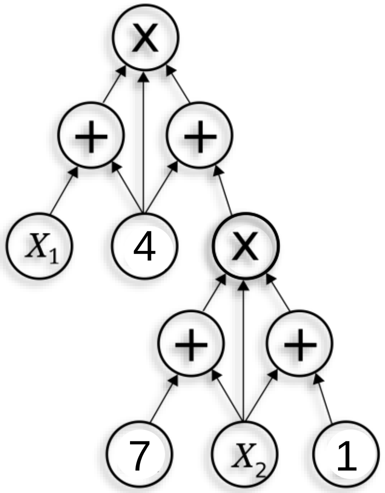
Computer Program



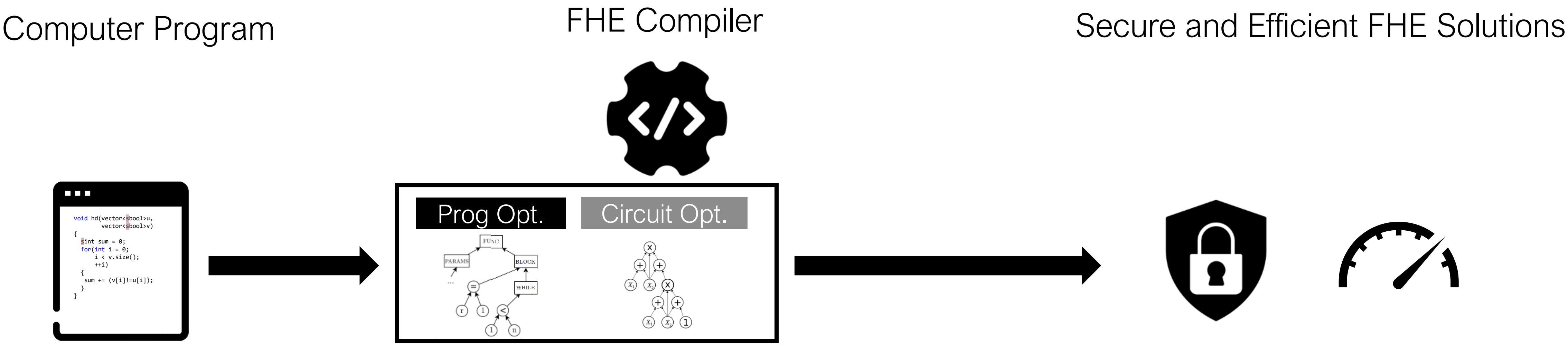
FHE Compiler



Arithmetic Circuit



End-to-End FHE Toolchain

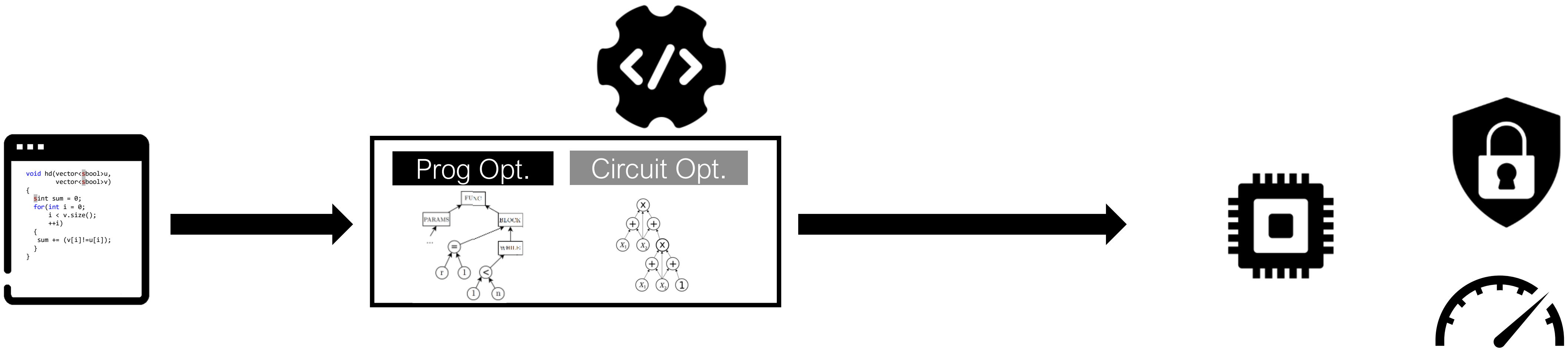


End-to-End FHE Toolchain

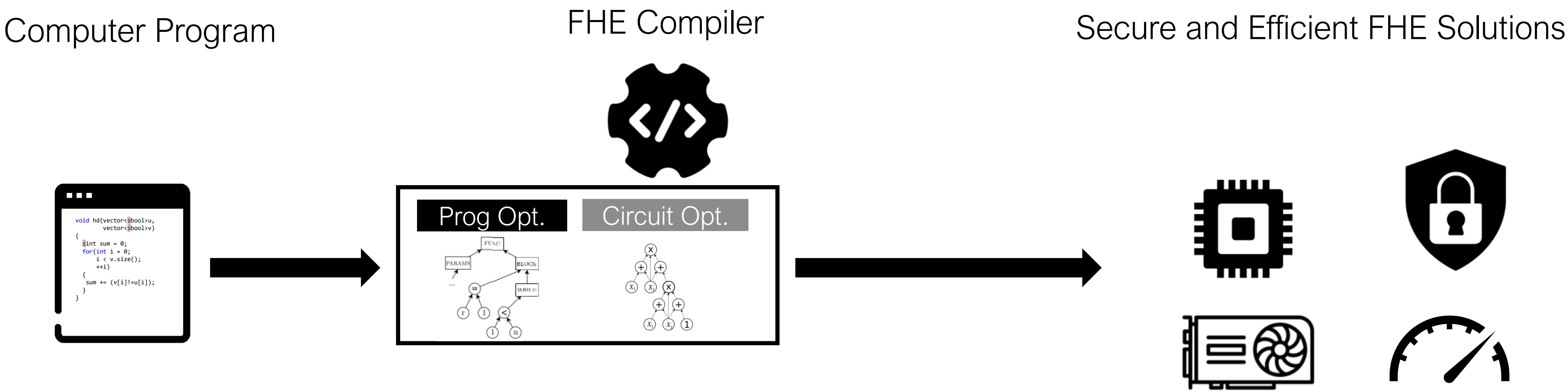
Computer Program

FHE Compiler

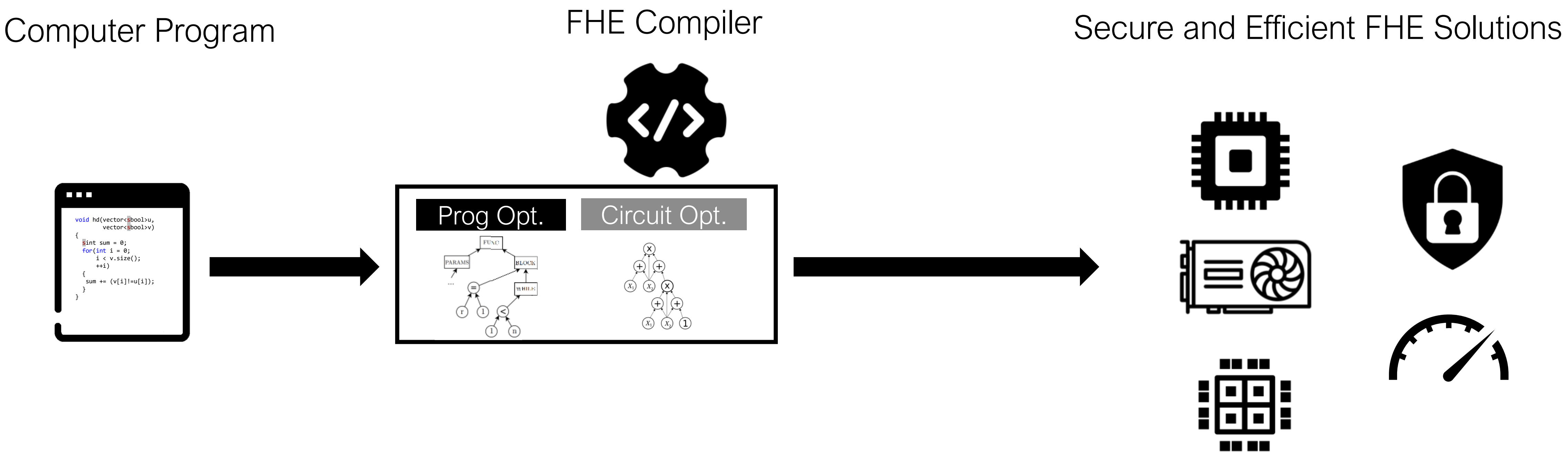
Secure and Efficient FHE Solutions



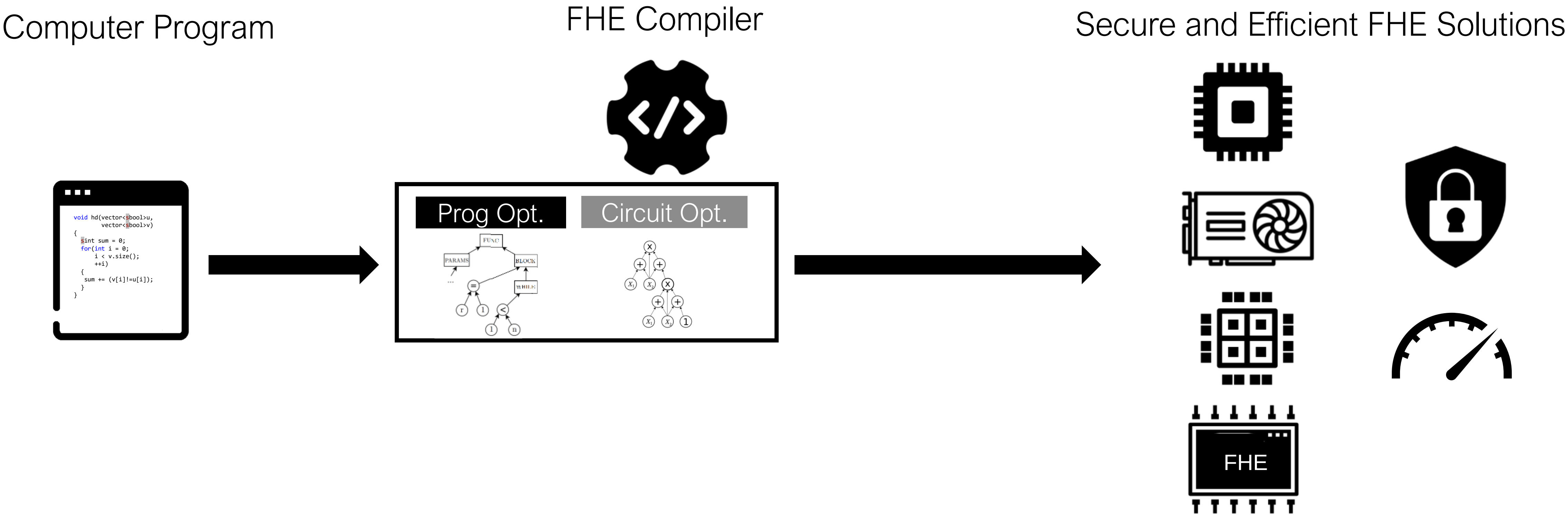
End-to-End FHE Toolchain



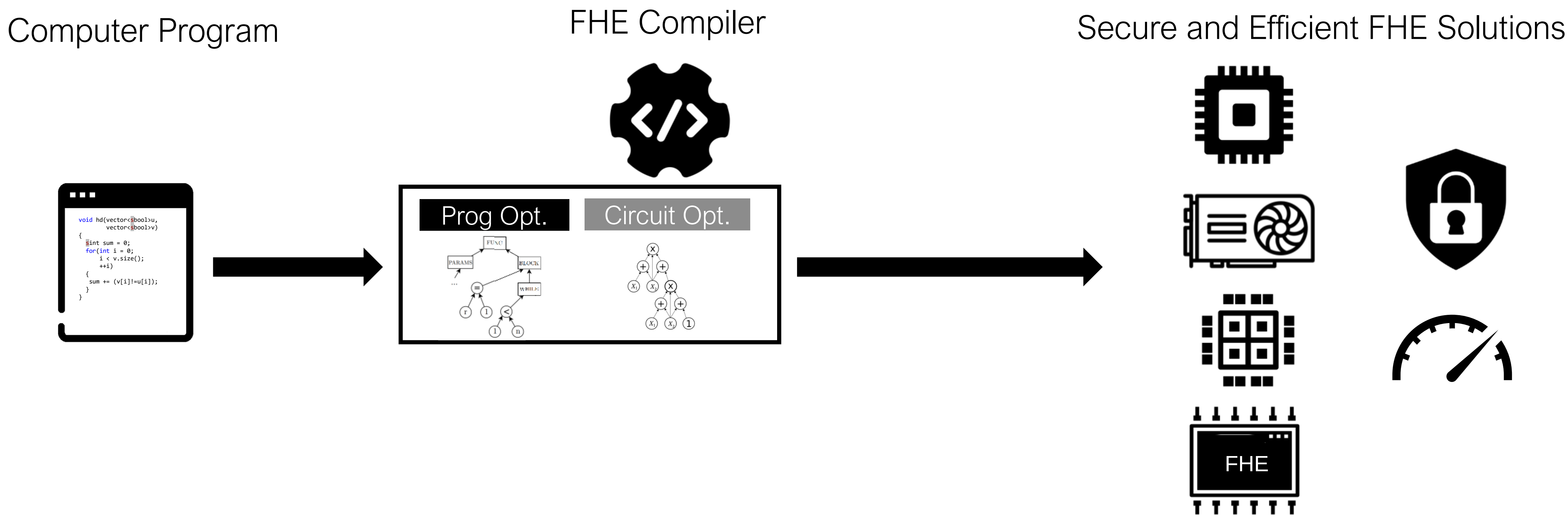
End-to-End FHE Toolchain



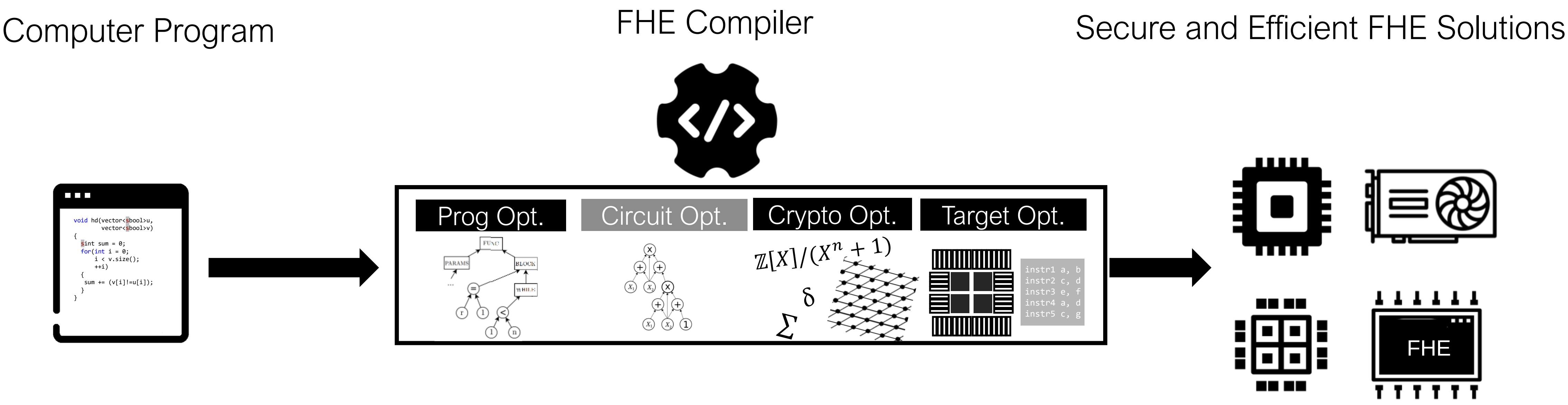
End-to-End FHE Toolchain



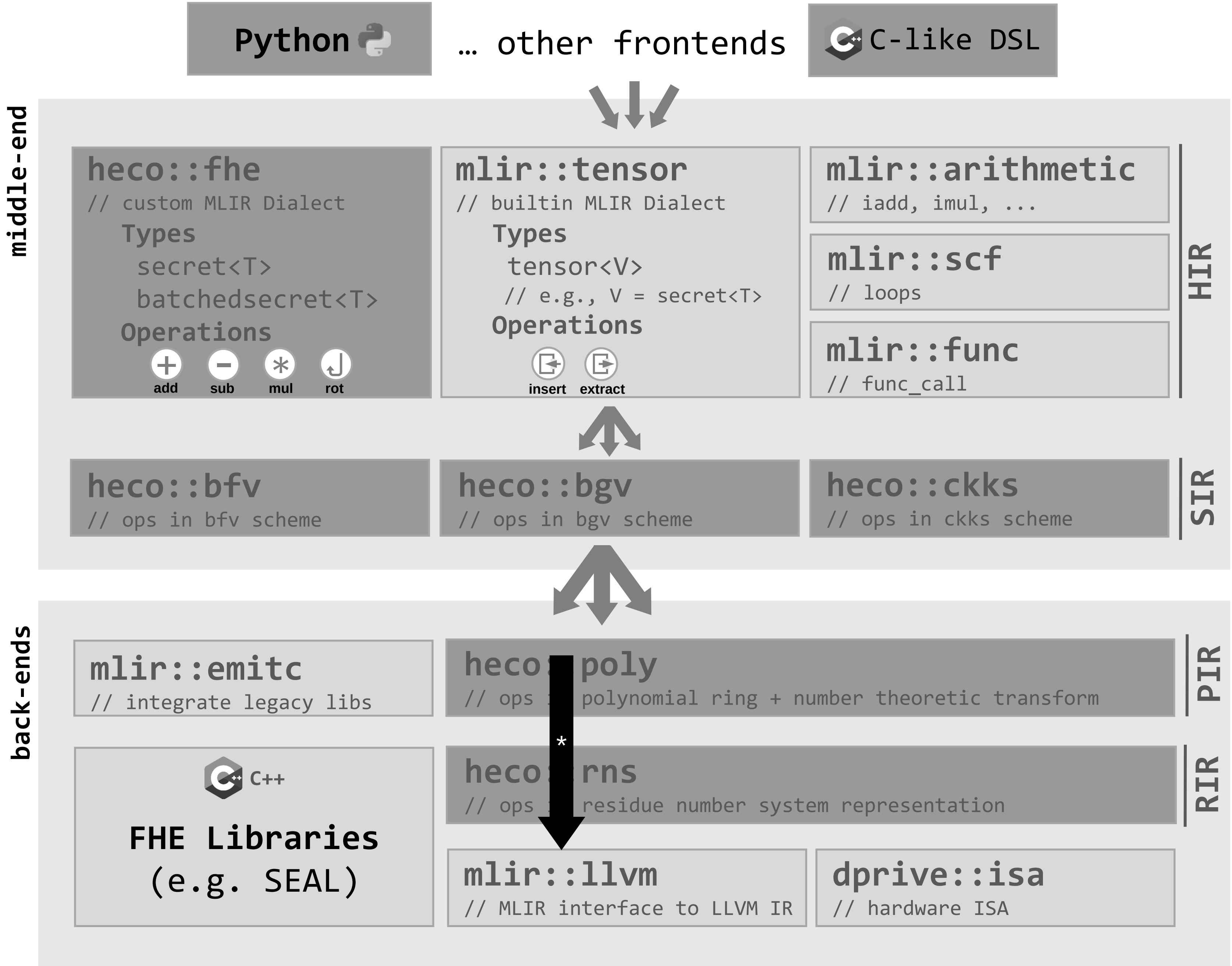
End-to-End FHE Toolchain



End-to-End FHE Toolchain

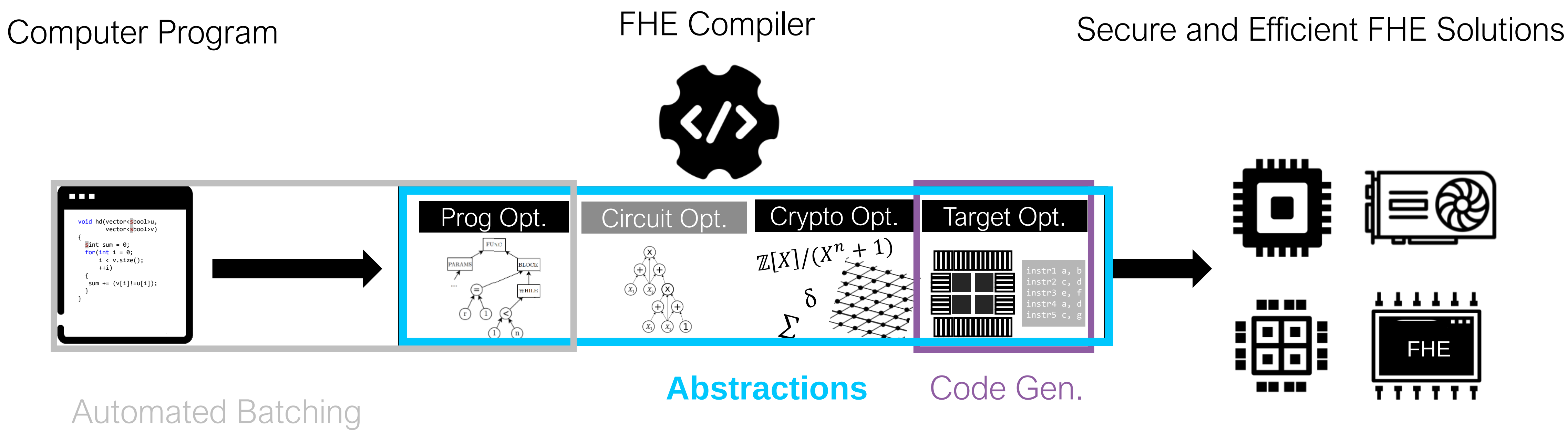


HECO Architecture



* Juneyoung Lee, Prototyping a compiler for homomorphic encryption in MLIR, EuroLLVM'22

HECO: Modular End-to-End Design



FHE Schem ardization

- H
- N 8)
- 2.8)

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

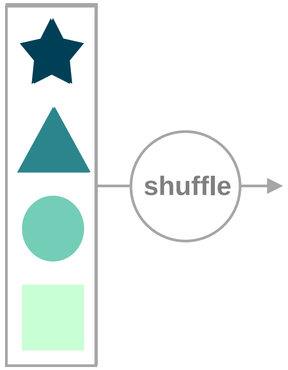
Standard C++

```
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

SIMD-like Parallelism

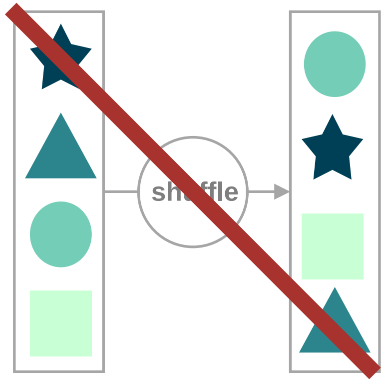
Standard C++

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

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

}
```

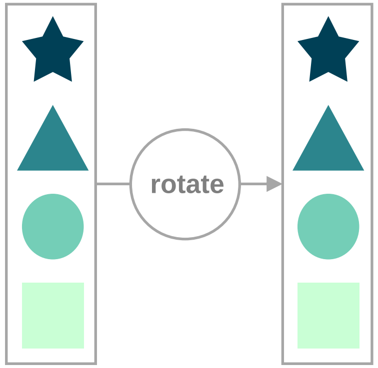
Batched FHE

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

   return a * b;

}
```

SIMD Batching



No efficient free permutation or scatter/gather



Only cyclical rotations

SIMD-like Parallelism

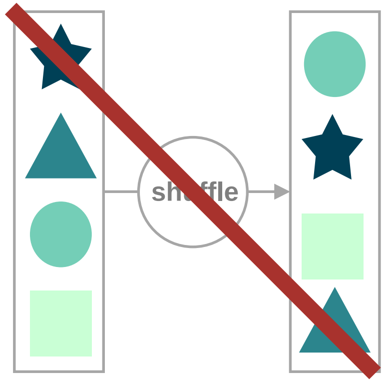
Standard C++

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

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

}
```

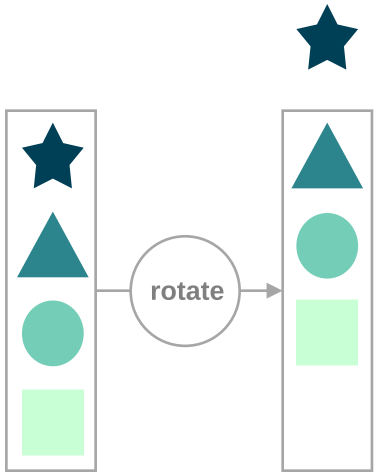
Batched FHE

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

   return a * b;

}
```

SIMD Batching



No efficient free permutation or scatter/gather



Only cyclical rotations

arxiv.org/abs/2202.01649

github.com/MarbleHE/HECO