Fully Homomorphic Encryption Back-End for CirC
07-300, Fall 2021

William Seo
https://northrim.github.io/07400-s22

November 12, 2021

1 Project Description

I will be working with Professor Wenting Zheng and her PhD student Edward Chen to develop a fully homomorphic encryption (FHE) back-end to CirC, "an infrastructure for building compilers to" existentially quantified circuits, an abstraction suitable for implementing cryptographic primitives [9].

Homomorphic encryption is a special type of encryption that supports computation on encrypted data without the need to decrypt it first. FHE is the highest level of homomorphic encryption, which supports multiple operations to an unlimited depth. Efficient and accurate FHE is highly sought due to its many valuable applications. In the world today, there exists a vast amount of data. Additionally, there exist many useful ways to utilize data, especially with the development of machine learning. However, "concerns over privacy legal issues" make a lot of useful data such as customer data, medical records, and video footage unavailable to use [5][10]. FHE can lift this restriction on data by allowing any data to be used in computation without violating its privacy.

Unfortunately, modern day FHE has some glaring weaknesses. Firstly, it is slow; running computations on homomorphically encrypted data is orders of magnitude slower than running computations on unencrypted data. Secondly, it is difficult to develop software using FHE, as it requires a strong cryptographic background. "Even for experts, this process is laborious", as it entails the careful use of "addition, multiply, and rotation instructions while minimizing noise accumulation" [2].

Our goal with this project is to overcome the weaknesses of FHE by implementing an FHE back-end into CirC, an infrastructure analogous to LLVM, which is an infrastructure for compiling front-end languages to machine code [8]. "The key abstraction is its intermediate representation (LLVM IR)". Similarly to LLVM, "language designers can add new front-ends that compile to CirC-IR", and also add "back-ends that compile from CirC-IR to a given EQC." Additionally, optimization passes can be made in the CirC-IR to improve efficiency [9].

Implementing an FHE back-end will allow developers to write FHE programs using front-end languages such as C without the requirement of having a strong cryptographic background. It will remove the necessity of having to worry about minimizing noise, as this will be done automatically by the compiler. Additionally, IR-based optimization passes can improve the speed of FHE programs compiled using CirC. Finally, the FHE back-end can enable many applications such as Gazelle [6], which require multiple cryptographic back-ends simultaneously. Through this project, I hope to make FHE more accessible for a wider range of applications.

2 Project Goals

75% Project Goal	- Develop a working FHE back-end for CirC
100% Project Goal	- Incorporate EVA optimizations [3] into CirC - Pass the benchmark tests
125% Project Goal	- All of the above - Implement IR optimizations - Apply CirC in FHE + MPC program (i.e. Gazelle) [6]

3 Project Milestones

- Yellow: Goals for before the spring semester
- Green: 75% Goals
- Blue: 100% Goals
- Red: 125% Goals

Prerequisites	- Learn Rust and Git - Familiarize myself with the CirC code base - Familiarize myself with the backend compiler passes - Complete starter compiler tasks
February 1st	- Incorporate Microsoft SEAL into the CirC repository Refer to EVA compiler [3] / ABY back-end [4] - Identify which benchmarks we want to run
February 15th	- Begin incorporating binary operations - Write corresponding test cases - Pass the test cases for FHE addition
March 1st	- Continue incorporating additional binary operations i.e. negate, subtract, multiply, rotate - Pass test cases for additional binary operations
March 15th	- Continue incorporating additional binary operations i.e. relinearize, mod switch, rescale, encode - Pass test cases for additional binary operations
March 29th	- Incorporate optimizations defined in EVA [3] - Add graph based optimizations that determine the best locations to add mod switches/relinearization/bootstrapping
April 12th	- Run benchmarks on FHE compiler backend
April 26th	- Prepare for presentation
Stretch	- Add optimizations - Integrate an FHE + MPC program

4 Literature Search

To understand the CirC infrastructure, I will need to read [9]. [2], [3], and [11] will be a good reference to how to implement a compiler for FHE and how to add optimizations. As the back-end will be written in Rust, the Rust Handbook [7] will be essential. Finally, the Dragon Book [1] will be helpful in learning about the workings of compilers.

5 Resources Needed

To write my code, I will use VSCode. I will also need a server to test the benchmarks. For this I will use CMU SNAP.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] Meghan Cowan, Deeksha Dangwal, Armin Alaghi, Caroline Trippel, Vincent T. Lee, and Brandon Reagen. Porcupine: A synthesizing compiler for vectorized homomorphic encryption. *CoRR*, abs/2101.07841, 2021.

[3] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madanlal Musuvathi. EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. *CoRR*, abs/1912.11951, 2019.

[4] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby - a framework for efficient mixed-protocol secure two-party computation, 01 2015.

[5] Alexandre Gonfalonieri. Homomorphic encryption & machine learning: New business models, Oct 2020.

[6] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, Baltimore, MD, August 2018. USENIX Association.

[7] S. Klabnik and C. Nichols. *The Rust Programming Language (Covers Rust 2018)*. No Starch Press, 2019.

[8] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[9] Alex Ozdemir, Fraser Brown, and Riad S. Wahby. Unifying compilers for snarks, smt, and more. *IACR Cryptol. ePrint Arch.*, 2020:1586, 2020.

[10] William Seo. Implementing homomorphic encryption. *07300 Critique Essay*, 2021.

[11] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. Sok: Fully homomorphic encryption compilers. *CoRR*, abs/2101.07078, 2021.