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 [8].

Homomorphic encryption is a special type of encryption that supports computation on encrypted data without the need to decrypt it first. FHE is the a specific type homomorphic encryption that supports multiple operations to an unlimited depth.

Practical FHE can open doors to new applications that operate on sensitive data. Concerns over privacy violations make sensitive user data, such as geographic data, medical records, and video footage, hard to use in untrusted environments, such as the cloud [5]. However, FHE can lift these restrictions by allowing computations to be performed on encrypted data. Therefore, malicious adversaries in the untrusted environment cannot gain access to any of the underlying plaintext data.

Unfortunately, modern day FHE has some glaring weaknesses. First, running FHE computations is extremely slow. Running computations on homomorphically encrypted data is orders of magnitude slower than running computations on unencrypted data. Second, developing FHE programs is unfriendly and 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, a shared compiler infrastructure extensible to a variety of cryptographic back-ends. The key abstraction of CirC is the CirC Intermediate Representation (CirC-IR). Designers can implement front-ends (i.e. C, Zokrates) that compile to the CirC-IR and add cryptographic back-ends (i.e. SMT Solver, Proof Systems) that compile from the CirC-IR. Additionally, optimization passes applied to CirC-IR, such as determining optimal relinearization placement, can greatly improve the performance of FHE programs [8].

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. The extensible front-end of CirC will allow developers to easily integrate CirC into their system. Also, 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, integrating a FHE back-end into CirC will enable new opportunities to explore applications that require primitive mixing, using two or more cryptographic primitives to improve the performance of secure computations.

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 an optimization pass - 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 further optimizations - Integrate an FHE + MPC program

4 Literature Search

To understand the CirC infrastructure, I will need to read "Unifying Compilers for SNARKs, SMT, and More" [8], which is a paper that describes the CirC infrastructure. "Porcupine: A Synthesizing Compiler for Vectorized Homomorphic Encryption" [2], "An Encrypted Vector Arithmetic Language and Compiler for Efficient Homomorphic Computation" [3], and "SoK: Fully Homomorphic Encryption Compilers" [9] will be good references 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

For my code editor, I will use Visual Studio Code. I will also need a server to test the benchmarks. For this, the CMU SNAP group has a cluster of servers that I will use to perform the benchmarks.

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

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