Integrating MPC and ZK-compiler technology

Brett Hemenway Marcella Hastings
Daniel Noble
{ fbrett, mhast, dgnoble } @cis.upenn.edu

1 Motivation and Overview

In recent years, academic groups have developed many efficient, open-source compilers to translate code from a high-level language into secure multiparty-computation (MPC) protocols. MPC compiler technology has improved dramatically since Fairplay [10], the first practical implementation, was introduced in 2004. Most MPC compilers convert an arbitrary function defined in a high-level language into an optimized circuit representation, which is passed to an implementation of a circuit-based MPC protocol like Garbled Circuits, GMW or BGW.

Compilers for ZK proof systems have similar constraints to MPC protocols, and the ZK programming pipeline is similar. Indeed, the libsnark documentation suggests the following workflow: "Express the statements to be proved as an R1CS (or any of the other languages above, such as arithmetic circuits, Boolean circuits, or TinyRAM). This is done by writing C++ code that constructs an R1CS, and linking this code together with libsnark." [18]

Recognizing these similarities, several groups have attempted to leverage MPC technology: some ZK compilers use the FairPlay infrastructure to convert a high-level language (SFDL) into a circuit [20], and the Pepper project has a similar pipeline, compiling a C-based language into a constraint system suitable for proof using libsnark [13].

We propose integrating modern MPC compilers with zk-SNARK toolchains. MPC compilers like Wysteria [2, 16] and EMP-toolkit [22] provide high-level languages for generating circuits and executing them securely. Circuit generators like Frigate [12, 11] and CBMC-GC [7, 19] convert C code into highly optimized Boolean circuits. The circuits generated by these MPC compilers are intended for consumption by an MPC protocol, but we plan to translate them into R1CS format and execute them with zk-SNARK technology (e.g. libsnark).

As we work to connect MPC compilers to zk-SNARK back-ends, we will also be on the lookout for places where SNARK technology (such as tinyRAM [3, 4, 14]) could be used to improve MPC compilers.

2 Technical Approach

The circuit generators and MPC compilers above are designed to generate circuits for use in MPC protocols, but these representations are essentially application-agnostic, and could be used in proof systems instead of MPC protocols. The challenge in this project is developing the pipeline from generated circuits to ZK compilers. In principle, these pipelines should be straightforward, but significant engineering challenges remain: many of the circuit formats

are undocumented, and the compilers are generally research-quality code with minimal I/O options.

We plan to begin our exploration with Wysteria, EMP-toolkit, Frigate and CBMC-GC. Other advanced MPC compilers like ABY [5, 6] Obliv-C [17], PICCO [15] and SCALE-MAMBA [9] (formerly SPDZ) build and execute arithmetic and Boolean circuits internally, but unlike the compilers listed above, they do not provide an easy method to export the underlying circuit, and would be more difficult to integrate into the zk-SNARK pipeline.

If we are successful in creating a pipeline for generating zk-SNARKs, a possible future direction would be to make this pipeline *verifiable*. In other words, given an established pipeline, can we verify that the resulting circuits and protocols are correct. Integrating tools from verifiable computation would allow us to provide end-to-end assurance of the integrity of the entire pipeline.

3 Team Background and Qualifications

Our team consists of three researchers at the University of Pennsylvania, Brett Falk, Marcella Hastings and Daniel Noble. Over the past year, our team has explored almost every modern open-source MPC compiler with an aim to assess the usability, limitations and efficiency of the field. The knowledge we have gained from this exploration puts us in a unique position to leverage advances in MPC-compiler technology in the ZK space.

Brett Hemenway Falk is a research assistant professor in the CIS department at Penn. Dr Falk's work focuses on cryptography and multiparty computation. Marcella Hastings is a third-year PhD student at the University of Pennsylvania. Her work focuses on applied cryptography, particularly on leveraging cryptographic tools to better serve developers. Daniel Noble is a first-year PhD student at the University of Pennsylvania, focusing on practical applications of secure computation protocols.

4 Evaluation Plan

The long-term goal of this project is to improve both MPC and zk-SNARK compiler technology by develop software connectors between compilers that allow developers to take advantage of breakthroughs across fields. In the short-term, there are many concrete goals that would indicate some level of success. These goals include:

1. Compatibility: Can we execute any circuit produced by an MPC compiler with the libSNARK back-end?

2. Circuit size: How do circuit sizes compare when using MPC compilers vs SNARK compilers?

3. Execution time: Can we improve the concrete proving time for a specific computation using MPC compilers?

4. (Stretch goal) Using SNARK front-ends for MPC: Can we extract a circuit generated by a SNARK compiler like Pequin or xjSNARK and execute it with an MPC back-end like ABY?

5 Security Considerations

Better front-end compilers for zk-SNARKS should lower the barriers to building zk-SNARKS and facilitate wider zk-SNARK deployments. Wider adoption of ZK technology should improve user-privacy and the security of the system as a whole. Hand-coding and optimizing circuits (or R1CS formulae) is a tedious, error-prone task, and allowing programmers to write in a high-level language should reduce implementation errors and improve overall security privacy.

This project will not directly impact end-users of the ZCash system, and thus will not require any widespread training. Instead, this project is aimed at building tools for developers. Unfortunately, we are still a long way from building compilers that are easy enough to use that they require no background knowledge, and current compiler technology is still aimed at users who have significant cryptographic expertise. Thus the developers (who would be the users of our tools) would still require some amount of practice or training to use them correctly. If we are successful in this project, a possible future direction would be to make these compilers verifiable, which would further reduce the risk of developer errors and improve the overall security of the system further.

6 Schedule

This project is requires some research exploration, and thus some of the concrete goals may shift slightly as we learn more about the capabilities and limitations of existing SNARK compilers. Survey existing SNARK compilers: We will begin by examining the state-of-the-art SNARK compilers, beginning with: Pequin [13] Snårkl [21] and xJsnark [1, 8].

- Understand the libSNARK input formats: libSNARK supports many different input types, and linking libSNARK with MPC compilers will require a detailed understanding of the input formats supported by libSNARK.
- Link MPC compilers to libSNARK: Our connection task will be to link the circuit output of an MPC compiler with libSNARK. We will begin with:
  - Frigate: Takes a high-level, C-like language and outputs optimized (boolean) circuits. In our previous tests, we have found Frigate extremely efficient and easy to use.
  - CBMC-GC: Takes a subset of ANSI-C, and outputs an optimized Boolean circuit. An interesting feature of CBMC-GC is that the circuits are highly optimized. CBMC-GCs optimization techniques are fairly computation-intensive and are thus not suitable for compiling large circuits. It will be interesting to see whether these aggressive optimization techniques can improve performance in some of the smaller circuits that are common in the ZK field.
- Benchmark: We will benchmark circuit sizes and running times

7 Budget and Justifications

The primary cost of this project is researcher time. The overall scope of this project – linking MPC and SNARK compiler technology for mutual benefit – is large, and could

easily motivate an enormous research effort. We believe, however, that with 2 months of effort for each of the members of our research team, we could begin to make the connections outlined in the timeline above. Two months of salary coverage, including benefits, tuition and university overhead for the three team members would cost about $86K.

References

[1] akosba. xjsnark. GitHub. `https://github.com/akosba/xjsnark`. Accessed 8 June 2018.

[2] aseemr. Wysteria. BitBucket. `https://bitbucket.org/aseemr/wysteria/`. Accessed: 18 May 2018.

[3] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying progrma executions succinctly and in zero knowledge. In *Lecture Notes in Computer Science*, CRYPTO'13, pages 90–108, 2013.

[4] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 781–796, Berkeley, CA, USA, 2014. USENIX Association.

[5] D. Demmler, T. Schneider, and M. Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Proceedings of the 2015 Network and Distributed System Security Symposium*, NDSS'15, 2015.

[6] Encrypto Group. ABY. GitHub. `https://github.com/encryptogroup/ABY`. Accessed: 17 May 2018.

[7] Andreas Holzer, Martin Franz, Stefan Katzenbeisser, and Helmut Veith. Secure two-party computations in ansi c. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 772–783, New York, NY, USA, 2012. ACM.

[8] A. Kosba, C. Papamanthou, and E. Shi. xjsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 543–560, 2018.

[9] KULeuven-COSIC. SCALE-MAMBA. GitHub. `https://github.com/KULeuven-COSIC/SCALE-MAMBA`. Accessed 14 June 2018.

[10] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 20–20, Berkeley, CA, USA, 2004. USENIX Association.

[11] B. Mood. Frigate. BitBucket. `https://bitbucket.org/bmood/frigaterelease`, Accessed: 17 May 2018.

[12] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *Proceedings of the IEEE European Symposium on Security and Privacy*, 2016.

[13] Pepper Project. Pequin. GitHub. https://github.com/pepper-project/pequin. Accessed: 17 May 2018.

[14] Pepper Project. TinyRam. GitHub. https://github.com/pepper-project/tinyram. Accessed: 18 May 2018.

[15] PICCO-Team. PICCO. GitHub. https://github.com/PICCO-Team/picco. Accessed 17 May 2018.

[16] Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 655–670, Washington, DC, USA, 2014. IEEE Computer Society.

[17] samee. Obliv-C. GitHub. https://github.com/samee/obliv-c. Accessed 17 May 2018.

[18] Scipr-Lab. libSNARK project. GitHub. https://github.com/scipr-lab/libsnark. Accessed: 17 May 2018.

[19] TU Darmstadt Security Engineering Group. CBMC-GC. https://www.seceng.informatik.tu-darmstadt.de/research/software/cbmc-gc/. Accessed: 18 May 2018.

[20] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 12–12, Berkeley, CA, USA, 2012. USENIX Association.

[21] Gordon Stewart, Samuel Merten, and Logan Leland. Snårkl: Somewhat Practical, Pretty Much Declarative Verifiable Computing in Haskell. In *PADL'18: The 20th International Symposium on Practical Aspects of Declarative Languages*, LNCS. Springer, 2018.

[22] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. GitHub, 2016. https://github.com/emp-toolkit. Accessed: 17 May 2018.