An Intermediate Language for Garbled Circuits

William Melicher
University of Virginia
Email: wrm2ja@virginia.edu

Samee Zahur
University of Virginia
Email: samee@virginia.edu

David Evans
University of Virginia
Email: evans@cs.virginia.edu

***Abstract*—Secure two-party computation allows two parties to evaluate a function of their private inputs without revealing their own inputs to the other party. The garbled circuit technique, developed by Andrew Yao, is a generic approach to secure computation, but has traditionally been viewed as impractical due to lack of efficient frameworks for generating and executing garbled circuits. Our group has been working on ways to make garbled circuit execution more efficient and scalable. Prior to this work, that has required low level circuit descriptions and extensive manual optimization effort. This work introduces GCParser, a modular intermediate level language for easily optimizing and executing garbled circuits. We demonstrate that this language can efficiently take advantage of optimizations for garbled circuits.**

I. Introduction

Secure two-party computation allows two parties to evaluate a function of their private inputs without revealing their own inputs to the other party. In 1982, Andrew Yao [1] developed a technique called garbled circuits to implement secure computation for generic functions. Later this technique was proven to be secure by Lindell and Pinkas in 2009 [2]. Secure computation has been an area of theoretical research, but has seen limited applications, due in part to lack of an efficient and flexible framework. Previous approaches to garbled circuit development, most significantly Fairplay [3], were viewed as being too inefficient. Indeed, secure computation via garbled circuits was, and still is, orders of magnitude more difficult than using a trusted third party. However, using a trusted third party is not always appropriate, especially in privacy sensitive applications.

Since their creation, garbled circuit techniques have been incrementally improving, allowing more practical applications. Recent work has mitigated many of these problems, and resulted in significant gains in efficiency and scalability [4], [5], [6], [7]. These gains make practical applications of secure computation more tractable on diverse platforms. Natural applications of secure computation have historically been focused on preserving privacy, and include biometric identification [8], and private auction systems[9].

It is difficult to generate efficient secure computations directly from programs in high-level languages. Our goal is to provide a standard intermediate language that can be used with a variety of front ends and back ends. Because GCParser is neither too high level, nor too low level, a range of optimizations can be applied. High level languages can have difficulty taking advantage of low level optimizations like preferring XOR gates, which can be used with no overhead [10]. However, low level languages do not have enough high level semantics to perform optimizations like reordering expressions to favor completely local computation.

Fairplay [3] was an early system to create a high level language for describing garbled circuits. Since its creation, many incremental optimizations have been made that require low level knowledge of the specific operations to be done. Optimizations such as determining which operations can be done locally without loss of privacy[11] can be difficult without finer grained information about the circuit. This finer-grained information is not available with a high level language. Incorporating these low level optimizations into a high level language like Fairplay is difficult without heavily modifying the language. Part of the motivation for GCParser was to avoid these difficulties with creating an efficient high level language for garbled circuits.

II. The GCParser Approach

This poster presents GCParser, an intermediate language for describing and executing garbled circuits. It was designed to allow optimized circuits to be shared across a variety of both front ends and back ends. GCParser builds on our previous low level framework for executing garbled circuits. Our previous work [4] implemented techniques for more efficient and scalable garbled circuits, by pipelining circuit generation and execution, which reduced the exorbitant memory constraints required by large circuits. However, GCParser is independent of the particular circuit execution framework, and allows execution of other back ends for secure computation. It is intended to be used as a generated language by other utilities, allowing them to take advantage of any optimizations that act on GCParser. GCParser operates at the level of variables and expressions, but does not have higher level language constructs such as looping. This makes GCParser easy to optimize, and allows code to be easily generated for GCParser. Here are some of the more notable features of GCParser language.

(a) *Agnostic to Higher Level Front ends*
GCParser is independent of high level frameworks, and can therefore be used with different interfaces to the language. Although there are no current high level front ends for GCParser, this will be a focus of future work. Any front end that supports the GCParser language interface will implement any optimizations which act on the GCParser language for free. This layering approach to a language for garbled circuits allows a variety of optimizations to be made, both at the high level and the low level.

(b) *Simple Parsing and Semantics*
The language specification itself is easily generated and parsed. The simplicity of the grammar allows many different front ends to compile to the GCParser language. A simple program written in the GCParser language to compute the minimum of two sums of integers is shown in Figure 1. You can see that the program has a simple structure; first describing input to the circuit, next output, and finally describing the operations of the circuit. Inputs are marked with numbers denoting the supplying party, and their bit length. Calculation lines have a single variable, created at that step, followed by the operation name, then the operands.

```
.input a1 1 16
.input a2 1 16
.input b1 2 16
.input b2 2 16
.output minimum
sum1 add a1 b1
sum2 add a2 b2
minimum min sum1 sum2
```

Fig. 1. A Small GCParser Program

(c) *Optimizations can Act on Intermediate Code*
GCParser is designed with the ability to easily perform optimizations that control the evaluation of expressions. This is due to the simplicity of the syntax and expressions which allow analysis of the computation. Because nearly all operations in the GCParser language are in a static single assignment form, information flow can be tracked through a program[12]. This allows optimizations described by Kerschbaum[11] to infer which operations do not hide data, and can therefore be done non-privately. An example of this would be computing the minimum of x and x+2. This expression will always be equal to x, but this transformation is difficult to do in a low level language where it is unclear which gates perform which arithmetic operations. Similarly, it is difficult to perform in a high level language where this form only exists in certain instances of an unrolled loop.

(d) *Flexibility of Operations and Back ends*
The GCParser language is able to easily support new primitive operations by extending it with a short java custom circuit. Operations can act at different levels of abstraction, either on bits, or on variables. Because you can add primitive operations to the language, GCParser can be used with different garbled circuit back ends while still benefiting from GCParser optimizations.

III. Preliminary Results

We have developed prototype code to run a secure string difference algorithm on top of the GCParser language. We plan to extend GCParser with a high level front end interface. This prototype test indicates that this will both increase performance and allow easier development of efficient garbled circuit applications.

IV. Conclusion

For garbled circuits to be practical, there must be continued increases in performance. Significant performance gains have been made, but secure computation using garbled circuits is still orders of magnitude slower than using a trusted third party to perform the computation. However, for many applications, secure computation using garbled circuits is feasible, and improved language tools can help both to increase the performance of secure computation, and allow for easier development. GCParser explores a new way of designing languages for garbled circuits by incorporating a layered architecture. GCParser is an initial intermediate language, but future work will be focused on creating a high level language front end without sacrificing performance.

References

[1] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *FOCS*. IEEE Computer Society, 1982, pp. 160–164.

[2] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *J. Cryptol.*, vol. 22, no. 2, pp. 161–188, Apr. 2009. [Online]. Available: http://dx.doi.org/10.1007/s00145-008-9036-8

[3] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay - secure two-party computation system," in *USENIX Security Symposium*. USENIX, 2004, pp. 287–302.

[4] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *USENIX Security Symposium*. USENIX Association, 2011.

[5] R. Wang, X. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong, "Privacy-preserving genomic computation through program specialization," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 338–347. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653703

[6] J. K. Yan Huang, David Evans, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.

[7] D. E. Yan Huang, Jonathan Katz, "Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution," in *Oakland*. The Internet Society, 2012.

[8] Y. Huang, L. Malka, D. Evans, and J. Katz, "Efficient privacy-preserving biometric identification," in *NDSS*. The Internet Society, 2011.

[9] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *CANS*, ser. Lecture Notes in Computer Science, J. A. Garay, A. Miyaji, and A. Otsuka, Eds., vol. 5888. Springer, 2009, pp. 1–20.

[10] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, L. Aceto, I. Damgrd, L. Goldberg, M. Halldrsson, A. Inglfsdttir, and I. Walukiewicz, Eds. Springer Berlin / Heidelberg, 2008, vol. 5126, pp. 486–498, 10.1007/978-3-540-70583-3-40. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70583-3-40

[11] F. Kerschbaum, "Automatically optimizing secure computation," in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 703–714. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046786

[12] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently computing static single assignment form and the control dependence graph," *ACM Trans. Program. Lang. Syst.*, vol. 13, no. 4, pp. 451–490, Oct. 1991. [Online]. Available: http://doi.acm.org/10.1145/115372.115320