Review #410A

===========================================================================

Overall merit

-------------

3. Weak accept

Brief paper summary (2-3 sentences)

-----------------------------------

The paper presents a framework for generating secure and efficient implementations of 2-party protocols. The compiler is evaluated on case studies from machine learning.

Strengths

---------

- Presents a pragmatic solution to a timely problem

- Performance results are promising

Weaknesses

----------

- Limitations of the system are not described precisely

- Proofs are sketchy

- Program partitioning is poorly explained.

Detailed comments for the authors

---------------------------------

The paper presents a general framework for generating efficient implementations of 2-party protocols. The framework builds on ABY, an existing platform that achieves similar goals, but is much harder to use. Additionally, EzPC implements program partitioning, a counterpart to pipelining optimization; this optimization appears in prior work and was discussed but not implemented in ABY. The general idea of the paper is interesting, but the paper has some issues:

- Lack of clarity: the paper is generally well-written but on several occasions it fails to provide high-level intuition for key technical points. This is the case for program partitioning. How general is the approach? Is it fully automated or does it need programmer's input?

- Limitations: the paper should make the limitations of the framework

clear. Which kinds of loops and branching statements are allowed?

How do you deal with programs that branch on secrets or perform

array accesses depending on secrets? As future work, how would you

extend to malicious adversaries? Would it be possible to modify the compiler to deal with 3-party protocols?

- Proofs: the appendix contains sequences of statements. The paper

promises to release full proofs in a technical report. Proofs should

appear in the appendix.

- Evaluation: since it compiles to ABY, the main question seems to be

what would be the overhead between code written directly in ABY and

code output by the compiler. This is not discussed very clearly in

the paper. It is also not always clear whether the comparisons are

legitimate. The relationship with Gazelle is also insufficiently

discussed.

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

Review #410B

===========================================================================

Overall merit

-------------

4. Accept

Brief paper summary (2-3 sentences)

-----------------------------------

While building blocks for secure two-party computation (2PC) are emerging, developing efficient 2PC programs is still a challenging problem. This paper presents EzPC, a framework that allows programmers to write 2PC programs in a high-level imperative language. The EzPC compiler automatically transforms the high-level description to 2PC executables, leveraging existing 2PC libraries in a correct and secure way. Evaluation shows that EzPC offers comparable or even better performance compared with hand-crafted protocols.

Strengths

---------

+ Developing a user friendly language model for 2PC is a promising direction.

The EzPC source language offers the right interface for non-expert

cryptographer to develop 2PC programs.

+ The EzPC framework makes it possible to update backend (e.g., adding

+ new

building blocks) and optimize backend (e.g., adding a more efficient label

inference algorithm) without any changes in the source code.

+ Formalization of the language features and core correctness and

+ security

properties.

+ EzPC is shown to offer comparable or even better performance compared

+ with

hand-crafted protocols.

Weaknesses

----------

- The label inference algorithm is not fully fleshed out

Detailed comments for the authors

---------------------------------

The paper is well-written. It provides a convincing approach to developing 2PC programs for non-expert in cryptography. I liked the way that the paper formalizes the split of computation into arithmetic and boolean parts as a label inference problem in the intermediate language. This design offers great flexibility for future updates to the framework. The evaluation is also thorough and convincing: EzPC not only provides at least comparable performance compared with hand-crafted protocols, it only incurs reasonable programmer effort. Of course, formalization of the language features and core security properties is a big plus.

I don't have any major concerns about the paper. Here are a few comments:

1. Some rules are missing in the paper. For examples, evaluation rule for s1;s2 is missing in Figure 5; evaluation rules for s1;s2 and while loop are missing in Figure 9. Also, since there are multiple nondeterministic choices in Figure 7, I'd like to see how the implementation handles nondeterminism earlier in the paper, rather than waiting until page 9 to discover a simplistic label inference algorithm. Also, the paper mentions that "Only the variables that govern the control flow ... are assigned public labels". This seems too conservative. Does the evaluation show extra cost for creating circuits for public variables?

2. I'm not sure if I follow the motivation for code partitioning. It seems to me that the evaluation of intermediate language to circuits is compositional, in the sense that for s1;s2 where s1 (resp. s2) maps to circuits g1 (resp. g2), the overall circuits are g1+g2. So overall circuit size is roughly linear w.r.t. program size. What brings the scalability issue (I was expecting more severe issues such as exponential increment of circuit size)?

3. The paper focuses on Machine Learning applications, while the source language is clearly a general-purpose one. From the discussion on page 1, it seems the reason is to avoid declassification of intermediate values and indexing into arrays at secret indices. But the paper also mentions that "Some of our benchmarks require accessing arrays at secret indices" (page 9). I wonder what makes the authors believe that ML applications are at least less likely to have secret indices? Are there applications beyond ML (e.g., privacy-preserving bidding) that will benefit from EzPC? In other words, I think the EzPC could be repurposed for general computation, with some limitations on issues like declassification of intermediate values and secret indices.

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

Review #410C

===========================================================================

Overall merit

-------------

4. Accept

Brief paper summary (2-3 sentences)

-----------------------------------

Multi-party computation (MPC) provides strong protections for data, but is hard to use by all but a small number of experts. This paper proposes the EzPC compiler, which makes it easier for non-experts to efficiently incorporate such privacy-preserving primitives into their programs.

Strengths

---------

-Enabling non-experts to have a shot at efficient MPC is an important next step for this community.

-The authors pick a wealth of applications to test against, showing convincing results.

Weaknesses

----------

-While it was nice that a non-cryptographer was able to use your system, it would have been nice to see stronger evidence that the proposed technique really was easier for non-experts.

Detailed comments for the authors

---------------------------------

It is pretty amazing to look back at the progress made in the MPC over the course of the last decade. While long viewed as nothing more than a theoretical curiosity, our community has helped to make advances in computation across a number of different primitives (e.g., Yao's garbled circuits, fully homomorphic encryption, etc) that have moved the use of such constructions into the realm of practicality.

The challenge for many now is to find ways to make those advances accessible to non-expert programmers. That is, a programmer should not have to be cryptographers who specialize in this particular topic in order to be able to incorporate these mechanisms into their programs.

However, the state of the art has long been very much this.

That's one reason that I like this paper - it pushes hard on the integration of this superb line of research to ensure that it can actually be used by a wider set of developers than a limited number of researchers. This is the next critical step if the area is really to take flight.

The authors did a particularly nice job on their evaluation, in my view.

I have read too many papers in this space that focus on just a single application, providing too narrow a view of the performance envelope of a technique. This work, however, looks at a wide array of learning algorithms, and I enjoyed going through the results.

Two complaints, but I believe that both can be handled with writing changes. The first is that the authors say that this approach is easier, and that even one of their co-authors (who is a non-cryptographer) can use their system. That's not exactly a strong metric - chances are, that person has a pretty good idea of what needs to be done to build a program that links/uses the results of one of the many compilers out there. You probably need to soften the language here, or at least promise publicly to perform a use study going forward to back up the claim. This approach is different, it's a step forward, but let's not claim usability without the experiment.

Finally, the authors should expand their list of applications listed in the related work (last sentence) to include the following well-known custom applications:

De Cristofaro and G. Tsudik, Practical Private Set Intersection Protocols, Financial Cryptography and Data Security Conference (FC), 2010.

Nipane et al., "Mix-In-Place" Anonymous Networking Using Secure Function Evaluation, Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2011.

De Cristofaro, P. Gasti and G. Tsudik, Fast and Private Computation of Cardinality of Set Intersection and Union, Conference on Cryptology and Network Security (CANS), 2012.

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

Review #410D

===========================================================================

Overall merit

-------------

2. Weak reject

Brief paper summary (2-3 sentences)

-----------------------------------

The paper presented EzPC, a compiler to facilitate constructing semi-honest 2PC protocols. Programmers can write an imperative program in EzPC source language, compile it using EzPC, and run it as a secure 2-party computation using ABY framework. They evaluated EzPC's performance using matrix factorization and a few classifiers including linear, naive Bayes, decision tree, and DNN.

Weaknesses

----------

My main concerns is that all the work described in this paper is reimplementing what have been done and published (quite a few years ago). More specifically,

1. the idea of compiling 2PC from high-level custom PL for \*\*easy programming\*\* has been practiced for years. Notably, existing works like Wysteria and ObliVM also offered the same \*\*formal security guarantees\*\* as those described in Section 4, and provided even more language features (such as secure declassification and supporting oblivious memory access) than EzPC. When the target applications (like circuits for machine learning) don't need those features, there is no penalty cost to use previous compilers.

2. the idea of \*\*cryptographic-cost aware compiler\*\* is not new. Several papers (such as [34] and a few more recent ones) practiced exactly this idea using ABY framework. By the way, the authors never clarify in the paper how the program partition and protocol assignment process are automated. What optimization algorithm does EzPC use? How does the optimization scale up with the application complexity?

3. the idea of \*\*Scalability\*\* through code partitioning has been used in [34], OblivC, and ObliVM.

So I cannot really find what's novel in this paper.

The authors claims "up to 19x" cost savings. Unfortunately, it is far from a fair comparison. The numbers are measured on completely different hardware and mostly using very different cryptographic techniques. There is no sign that the 19x performance improvements can be attribute to any thing novel in EzPC.

Detailed comments for the authors

---------------------------------

On page 2, the foot note of the scalability paragraph claims that using swap and disk space causes huge slowdown. I doubt this is the case considering the fact that high-performance SSD has become commonplace nowadays and that secure computation is typically computation and network bound while disk IO can happen in parallel with those expensive computation/network communications. It would be necessary to substantiate your claim with experiments. Also, it blames the scalability limitations of existing compilers without providing good justifications. Do you actually try some applications using existing compilers and find it doesn't work because of scalability issues?

Response by Rahul Sharma <[rahsha@microsoft.com](mailto:rahsha@microsoft.com)>

---------------------------------------------------------------------------

We thank the reviewers for their insightful comments. We acknowledge the feedback about improving the writing and adding details; we promise to incorporate it in the final version.

\*\*Comparison against previous compilers (ReviewerD)\*\*:

The comparison is in Section 2.1. To summarize, CBMC-GC times out after 5 hours when multiplying two tiny matrices; ObliVM scales better but is still 25x slower than EzPC on the same hardware.

\*\*Novelty of EzPC (ReviewerD)\*\*

The novelty of EzPC is in implementing cryptographic cost awareness inside a compiler, allowing the programmers to program in a high-level language absent of all the cryptographic details, while the compiler chooses the best circuit representation (boolean or arithmetic) for different parts of the program and also handles interconversions between them -- all in a provably secure manner. More specifically:

\*\*w.r.t previous compilers\*\*: Wysteria, ObliVM, and CBMC-GC, use boolean (or arithmetic) circuits only backends. And so, the challenges associated with multiple representations and interconversions are absent. Moreover, as shown in Section 2.1, they perform poorly on ML tasks that involve a mix of arithmetic and boolean computations.

\*\*w.r.t. [34]\*\*: [34] operates over circuits represented as a sequence of dyadic operations, while EzPC provides a high-level programming language with loops and branches. EzPC compilation is automatic (Section 3) and sub-quadratic in the size of generated ABY programs (e.g. Figure 3) after switching off memory safety checking. In sharp contrast, [34] describes approaches that would be doubly exponential and exponential in the length of ABY programs.

\*\*w.r.t Yao pipelining\*\*: Unlike prior work, which is specific to and works on Yao’s garbled circuits, our partitioning technique works at the source level and is agnostic to the cryptographic backend.

These features (ease-of-programming, use of multiple crypto backends, and scalability through partitioning) enable us to write and evaluate a wide array of ML algorithms such as matrix factorization with efficient generic 2PC techniques. Without EzPC, this would have been a near-impossible task and has never been done.

\*\*Scalability experiments for Partitioning (ReviewerD)\*\*:

These are in Section 7, Figure 11. For DNNs with 6 layers or more, swap causes slowdown and large programs terminate with SIGBUS.

\*\*Loops, branches, and secret array indices (ReviewerA and ReviewerB)\*\*

In our experiments, assigning public labels to control flow variables is critical for performance and/or security. However, we have observed that conservatively creating circuits for other public variables incurs minimal cost in practice.

At the expression level, EzPC provides secret conditionals (using the '? :' operator) which are implemented using multiplexers. Secret dependent array accesses can also be implemented using the '? :' operator.

\*\*Other comments\*\*:

ReviewerA:

Partitioning is fully automated and compatible with any 2PC protocol. The EzPC compiler, with suitable changes to operator costs, can target other backends including 3PC or MPC, or protocols secure against malicious adversaries. We are currently pursuing these directions.

The proofs follow by straightforward induction once the key lemmas (Appendix C) have been identified. We will include full proofs in a Technical Report.

We have evaluated against handwritten ABY code for the examples in the ABY repository and EzPC has comparable performance. However, these examples are not representative of practical ML tasks. Writing sophisticated ML applications such as neural networks in ABY is a very tedious and nearly impossible engineering task for non-crypto experts -- automating the same is our main contribution.

ReviewerB:

Scalability issues: when the circuit size is larger than the memory size, it either runs slowly or not at all (Figure 11). Partitioning enables decomposing g1;g2 that does not fit in memory into g1 and g2, both of which are individually small enough to fit in memory.

ReviewerC:

Thanks for additional interesting functionalities!