Compilation for Secure Two-Party Computations*

Martin Franz[1], Andreas Holzer[2], Stefan Katzenbeisser[1],
Christian Schallhart[3], Helmut Veith[2]

[1] TU Darmstadt
Security Engineering Group
Mornewegstrasse 32
D-64283 Darmstadt, DE

[2] Vienna University of Technology
Institut für Informationssysteme 184/4
Favoritenstraße 9-11
A-1040 Vienna, AT

[3] Oxford University
Wolfson Building
Parks Road
OX1 3QD Oxford, UK

Abstract: Secure two-party computation (STC) is a computer security paradigm that enables two distrusting parties to jointly compute on sensitive input data. While both parties are interested in the outcome of the computation, they are not willing to share their data with each other. Until recently, STC was deemed theoretical and impractical. However, novel efficient cryptographic primitives bring STC well within practical reach. Indeed, custom-tailored commercial STC products already appeared on the market. Unfortunately, a widespread application of STC is still hindered by the difficulty to implement STC protocols. While recent work proposed simple programming languages for the specification of STCs, they are still difficult to use for practitioners, and translating existing source code into this format is cumbersome. Similarly, the manual construction of STC protocols is labor intensive and error-prone.

We discuss recent work that forms a significant step towards practically realizable STCs that can be integrated into modern software engineering frameworks. In particular, we discuss the compiler CBMC-GC which uses model checking techniques to automatically generate efficient STC protocols from ANSI C programs. Experimental results demonstrate CBMC-GC's practical usefulness.

Overview

In modern information processing infrastructures, not only data but also code is becoming more mobile, e.g., in cloud services. With an increasing amount of sensitive information processed, there is an increasing demand for technical solutions that assure data secrecy and privacy, even if data is processed on potentially untrusted platforms. These solutions are called *Privacy Enhancing Technologies (PETs)* and the central cryptographic tool enabling such PETs are *Secure Two-Party Computations (STCs)*, allowing two distrusting parties to perform arbitrary computations on sensitive data without ever exposing their input in the clear. Hence, no information on the other party's input is revealed, beyond the information derivable from the commonly computed function output.

As an example, imagine Alice and Bob as two millionaires who want to determine the richer one among them – but *without* revealing how much they own, neither to the other

*This abstract is a summary of [FHK+14] and [HFKV12]. This work was supported by the Vienna Science and Technology Fund (WWTF) grant PROSEED, the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), CASED and EC SPRIDE.

millionaire nor to a trusted or untrusted third party. This is the "millionaires' problem", first described by Yao [Yao82], who thereby initiated research on STCs. Subsequently it has been shown that every computable function over two inputs is also computable in the framework of STC: Two players can evaluate the function on their respective private inputs so that the result of the computation is available to both, without needing to share the inputs in plain text with each other.

After 30 years of mainly theoretical studies, increased computational power and advanced cryptographic protocols make it feasible to evaluate reasonably large functions like secure auctions, data mining, or biometric face and speech recognition in an STC context. The predominant approach to implement STCs are *Garbled Circuits*, as originally proposed by Yao [Yao86]. The typical STC tool chain is as follows: An STC developer takes an algorithm and translates it into a Boolean circuit (oftentimes manually, in rare cases using a domain-specific language). This circuit is then passed on to an STC framework that interprets the circuit using STC. For example, for Yao's Garbled Circuits, party A garbles the circuit including party A's input such that party B can evaluate the circuit without knowing the intermediate truth values computed in the process.

One main obstacle for practical applications and widespread adoption of STC was the lack of support for general programming languages, as only circuit evaluation [HEKM11] or simplified programming languages [MNPS04] were supported. This situation changed recently when CBMC-GC[1], the first STC compiler for full ANSI C, was presented [HFKV12, FHK+14]. It extends the bounded model checker CBMC [CKL04]. CBMC-GC translates a C program into a circuit which is then deployed to the two STC parties A and B. Recent improvements in CBMC-GC's circuit optimization algorithms [FHK+14] bring the sizes of generated circuits close to or even improve over handoptimized circuits.

References

[CKL04] E. Clarke, D. Kroening, and F. Lerda. A Tool for Checking ANSI-C Programs. In *TACAS'04*, 2004.

[FHK+14] M. Franz, A. Holzer, S. Katzenbeisser, C. Schallhart, and H. Veith. CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations. In *CC'14*, pages 244–249, 2014.

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX'11*, 2011.

[HFKV12] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure Two-Party Computations in ANSI C. In *CCS'12*, 2012.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — A Secure Two-Party Computation System. In *SSYM'04*, 2004.

[Yao82] A. C.-C. Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS'82*, 1982.

[Yao86] A. C.-C. Yao. How to Generate and Exchange Secrets. In *FOCS'86*, 1986.

[1] `http://forsyte.at/software/cbmc-gc/`