Specifying Sharemind’s Arithmetic Black Box

Peeter Laud
Cybernetica AS
peeter.laud@cyber.ee

Alisa Pankova
Cybernetica AS
Software Technologies and Applications
Competence Centre
University of Tartu, Institute of Computer Science
alisa.pankova@cyber.ee

Martin Pettai
Cybernetica AS
Software Technologies and Applications
Competence Centre
University of Tartu, Institute of Computer Science
martin.pettai@cyber.ee

Jaak Randmets
Cybernetica AS
University of Tartu, Institute of Computer Science
jaak.randmets@cyber.ee

ABSTRACT
In this paper, we discuss the design choices and initial experiences with a domain-specific language and its optimizing compiler for specifying protocols for secure computation. We give the rationale of the design, describe the translation steps, the location of the compiler in the whole SHAREMIND protocol stack, and the results we have obtained with the system.

Categories and Subject Descriptors
D.4.6 [Operating Systems]: Security and Protection—Cryptographic controls; D.3.4 [Programming Languages]: Processors—Code generation

Keywords
Secure Multiparty Computation; Protocol optimization

1. THE PROTOCOLS OF SHAREMIND
Existing secure multiparty computation (SMC) frameworks use different protocol sets for achieving privacy. Several frameworks implement the arithmetic black box (ABB) [4], the methods of which are called during the runtime of a privacy-preserving computation by the SMC engine in the order determined by the specification of the computation. An ABB must at least contain the methods for linear combination and multiplication of private integers, but it contains more in typical implementations.

SHAREMIND SMC framework [2] features an exceptionally large ABB. Besides the operations listed above, it also contains comparison, bit extraction, widening, division of arbitrary-width integers [3], as well as a full set of floating-point [5] and fixed-point operations, including the implementations of elementary functions. More protocol sets on top of different SMC methods are planned.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PETShop’13, November 4, 2013, Berlin, Germany.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2489-2/13/11 ...$15.00.
http://dx.doi.org/10.1145/2517872.2517874.

Different protocols of the ABB form a hierarchy, with more complex protocols invoking simpler ones (multiplication, widening, certain bit-level operations) for certain tasks [3]. The implementation of protocols for ABB operations is an error-prone and repetitive task. Repetition is caused by implementing the protocols to work with values of different bit-length. Attempts to optimize complex protocols over the composition boundaries introduce errors and make the library of protocols unmaintainable. The task of building and maintaining implementations of protocols is naturally answered by introducing a domain-specific language (DSL) for specifying them.

The DSL allows us to specify the protocols in a manner similar to their write-up in papers on SMC protocols. This specification is compiled and linked with the SHAREMIND platform. There is a different language [1] for specifying the privacy-preserving applications as a composition of these protocols. Having different languages for implementing different levels of the privacy-preserving computation allows us to apply optimizations most suitable for each level, and improves the user experience by allowing us to tailor the languages for the specific domain. Protocols are specified and implemented in a declarative style, but applications are implemented in imperative style as a sequence of protocol invocations.

2. THE LANGUAGE FOR PROTOCOLS
Our protocol DSL is a functional language, mimicking the style of the pseudocode used to present protocols. A program in this language states, which party computes which values from which previously available values. Computations used several times can be abstracted as functions.

The language allows to state only once similar computations performed by different parties. Actually, this is the default mode and each variable x in the program denotes a separate value at each party. Defining x=f(y1,...,yn) causes each party to apply f to its own values of y1,...,yn and denote the result as x. To access the value of x at a particular party no. i, one may write x from i. In party i’s code, the pseudo-numbers Prev and Next denote the parties no. (i−1) and (i+1) (modulo the number of parties). There are specific syntactic constructs to state that certain computations have to be made only by a subset of parties.

Fig. 1 gives the specification for multiplying numbers u,v ∈ Z_2^*, additively shared between three parties [2] (i.e. a private value x ∈...
def reshare (u : uint[n]) : uint[n] = {
    let r = rnd (),
        w = u - r + (r from Prev);
    return w;
}

def mult (u : uint[n], v : uint[n]) : uint[n] = {
    let u' = reshare (u),
        v' = reshare (v),
        w = u' * v' + u' * (v' from Prev) + (u' from Prev) * v';
    return reshare (w);
}

Figure 1: Multiplication protocol and its DAG
come “circles”). This does not make a secure protocol insecure be-
cause it does not make the view of any party richer than it was. We
are currently developing a comprehensive library of optimizations
for such distributed arithmetic circuits.

We have tried out the optimizations on certain protocols described
in [3]. We have managed to reduce the amount of communication
of the largest protocols by around 4% (7% when not consider-
ing randomness that could be predistributed). Also, the number of
rounds the protocols need is reduced by 1–2, compared to [3]. The
composition of protocols also creates many places where constant
propagation and merging are useful.

4. ACKNOWLEDGEMENTS

This work was supported by the European Social Fund through
the ICT Doctoral School programme, and by the European Re-
gional Development Fund through the Estonian Center of Excel-
dence in Computer Science, EXCS, and through the Software Tech-
nologies and Applications Competence Centre, STACC. It has also
received support from Estonian Research Council through project
PUT2.

5. REFERENCES

programming of privacy-preserving applications. Cryptology

http://eprint.iacr.org/

framework for fast privacy-preserving computations. In
S. Jajodia and J. López, editors, ESORICS, volume 5283 of
Lecture Notes in Computer Science, pages 192–206. Springer,
2008.

High-performance secure multi-party computation for data

efficient multiparty computation from threshold homomorphic
encryption. In D. Boneh, editor, CRYPTO, volume 2729 of
Lecture Notes in Computer Science, pages 247–264. Springer,
2003.


computation with malicious adversaries. In Proceedings of the
21st USENIX conference on Security symposium, pages

verification of high-grade cryptographic applications. In
P. Ning, V. Atluri, V. D. Gligor, and H. Mantel, editors, FMSE,