Domain-Polymorphic Language for Privacy-Preserving Applications[*]

Dan Bogdanov
Cybernetica AS
dan.bogdanov@cyber.ee

Peeter Laud
Cybernetica AS
peeter.laud@cyber.ee

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

ABSTRACT

We present SecreC, a programming language for specifying privacy-preserving applications using a mix of techniques for secure multiparty computation. Building on the concept of *protection domain* as an abstraction of resources used to ensure the privacy of data, the SecreC language allows the specification of protection domains for different pieces of data, and the specification of the computation in *domain-polymorphic* manner. We have implemented the compiler for the language, integrated it with the existing SMC framework SHAREMIND, and are currently using it for new privacy-preserving applications.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Polymorphism*

Keywords

Secure Multiparty Computation; Software Engineering

1. INTRODUCTION

Many different secure multiparty computation (SMC) techniques exist, based on garbled circuits [14, 11, 10] or decision diagrams [8], secret sharing [13, 2, 5, 3], homomorphic encryption [4, 7]. When developing an application making use of SMC techniques, we may want to use more than one technique simultaneously, and/or we may want to defer the choice of particular SMC techniques to a later stage of development. The main reason for this is efficiency — different operations may be fastest using different techniques, even when considering the costs of translating between data representations [9]. Confidentiality policies may compound this issue, stating that different pieces of data must be treated with techniques providing protection against different kinds of adversaries (passive vs. active; the size of coalitions it's able to form). In this case, we may use faster techniques for data needing less protection. Also, in our quest for speed, we may want to try out and profile different SMC techniques; this should be possible without rewriting the application.

SHAREMIND's [2] secure computation runtime responds to this wish through modular and the ease of integration of new techniques. The application programmer, in order to make full use of the capabilities of the runtime, needs a language to express the functionality of the application and the possible choices of SMC techniques, without being forced to commit to particular techniques too early. In this paper we present SecreC, a language that supports such development methods. The design of the language has been somewhat inspired by Jif [12]. Several design choices have also been affected by our unique practical experience in developing SMC applications.

2. PROTECTION DOMAINS

A *protection domain kind* (PDK) is a set of data representations, algorithms and protocols for storing and computing on protected data. A *protection domain* (PD) is a set of data that is protected with the same resources and for which there is a well-defined set of algorithms and protocols for computing on that data while keeping the protection.

Each PD belongs to a certain PDK and each PDK can have several PDs. Protection domains are a fundamental concept in SecreC, partitioning the techniques and security resources available to the program.

A typical example of a PDK is secret sharing, with implementations for sharing, reconstruction, and arithmetic operations on shared values. A PD in this PDK would specify the actual parties doing the secret sharing, and the number of cooperating parties for reconstruction. Another example of a PDK is a fully homomorphic encryption [6] scheme with operations for encryption, decryption, as well as for addition and multiplication of encrypted values. Here different keys correspond to different PD-s. Non-cryptographic methods for implementing PDK-s may involve trusted hardware or virtualization. *Public* is also a PDK, containing a single PD. In general, a PDK has to provide a list of data types it operates with, and functions that operate on them. The cryptographic implementations of these functions are beyond the scope of the application programmer. These

[*]This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. Distribution Statement A (Approved for Public Release, Distribution Unlimited).

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/10.1145/2517872.2517875.

functions are made available by the SHAREMIND engine to applications, written in SecreC, through *system calls*. The available functions may be different for different PDKs and data types. It is possible that certain functionality that is provided through a dedicated system call in one PDK is implemented as a SecreC program for another PDK.

3. THE PROGRAMMING LANGUAGE

The execution of a privacy-preserving application consists of invoking the system calls in the correct order, correctly passing data between them. The task of a SecreC-program is to specify this order and the flow of data.

SecreC is an imperative, strongly and statically typed language, with the concrete syntax strongly influenced by C++. All variables and values in the program are typed. A type consists of the data type, and the PD. One can write *PD-polymorphic* code, possibly with restrictions to specific PDKs. In this paper we only show the abstract syntax while the concrete sytax of the language is significantly richer.

A SecreC program P consists of a sequence of declarations followed by the body of the main function. Every declaration is either a protection domain kind declaration, protection domain declaration (stating the PDK into which it belongs), or a function declaration. The names of the PDKs and PDs are made visible in the compiled code. During deployment, the SHAREMIND engine checks that it supports the necessary PDKs. At the same time, the PDs of the compiled program are matched with the actual resources of the runtime.

$$P \quad ::= (\texttt{pdk}\ k\texttt{;} \mid \texttt{pd}\ d\ \texttt{:}\ k\texttt{;} \mid F)^*\ B$$

A function definition F states the name of the function f, its return type (consisting of the PD d_0 and the data type t_0), the names and types of its arguments, and its body B. The function may be polymorphic, and the PDs can be type variables that are universally quantified either over all PDs, or over PDs of a certain PDK. Different arguments of the function may have different PDs; in particular, this is the case for (de-/re-)classification functions that conver the values from one PD to another. The type of a system call is defined similarly; but instead of its body, only its name in PDK implementations is mentioned.

$$\begin{array}{lcl} F & ::= & [Q]\ f\texttt{(}x_1 : d_1\, t_1\texttt{,}\ldots\texttt{,}x_n : d_n\, t_n\texttt{)}\ \texttt{:}\ d_0\, t_0\ (B \mid S) \\ Q & ::= & \forall\, \alpha_1, \ldots, \alpha_n\,\texttt{.} \qquad \alpha \ ::= \ d \mid d : k \\ S & ::= & \texttt{syscall(}f\texttt{)} \qquad B \ ::= \ \texttt{\{}(x_i : d_i\, t_i\texttt{;})^*\ s\texttt{\}} \end{array}$$

Importantly, SecreC supports *function overloading*, where several different functions with the same name, but different type may coexist in the same program. These are foremost used to express “the same” operations in different PDKs (e.g. arithmetic and boolean operations). These are also used to provide PDK-specific implementations for some generic functionality, where the PDK supports an efficient protocol for this functionality. At a call site, the compiler selects the called function according to its type, preferring functions with more precise domain types. If several functions with types of incomparable precision are available, the compiler either makes a reasonable default choice, or raises an error. The compiler also raises an error if the called polymorphic function attempts to use a function that does not exist in a concrete PDK (in C++ terms type errors may arise during template instantiation).

Data types, denoted with t, are not fixed, but definitely include integers `int`, booleans `bool`, and vectors of integers `int[]` and booleans `bool[]`. The statements s and expressions e in SecreC, shown below, are the usual WHILE-language constructs, complemented with operations to manipulate arrays.

$$\begin{array}{lcl} s & ::= & \texttt{skip} \mid s_1\ \texttt{;}\ s_2 \mid x\ \texttt{=}\ e \mid x\texttt{[}e_1\texttt{]}\ \texttt{=}\ e_2 \\ & \mid & \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2 \mid \texttt{while}\ e\ \texttt{do}\ s \mid \texttt{return}\ e \\ e & ::= & x \mid c_t \mid e\ \texttt{::}\ d \mid f\texttt{(}e_1\texttt{,}\ldots\texttt{,}e_n\texttt{)} \\ & \mid & e_1\texttt{[}e_2\texttt{]} \mid \texttt{mkarr(}e_1\texttt{,}e_2\texttt{)} \mid \texttt{length(}e\texttt{)} \end{array}$$

The expressions include function calls. The expressions and statements are typed; in an assignment $x := e$, the types of x and e are the same (as syntactic sugar, the compiler can automatically insert a call to classification function if the PD of e is *public*, and the PD of x is something else). In places where the compiler cannot infer the protection domain of some expression, it is possible to state it explicitly. Importantly, one does not attempt to hide the control flow of the program. Hence all branching decisions must be made based on public data.

Example.

Consider the task of sorting a vector of integers in privacy-preserving manner. The sorting method is polymorphic over the PD — the only restriction is that the given PDK supports basic arithmetic and comparison. The generic sorting function operates by constructing a sorting network and obliviously performing compare-and-swap on pairs in the order defined by the network.

```
∀ D. sort(src : D int[]) : D int[] {
  i : public int;
  a, b, c : D int;
  alength : public int = length(src);
  sn : public int[] = makeSortingNetwork(alength);
  for (i=0; i < length(sn)-1; i = i + 2) {
    a = src[sn[i+0]]; b = src[sn[i+1]];
    c = isLessThan(a, b); // from PDK implementation
    src[sn[i+0]] = c*a + (1 - c)*b;
    src[sn[i+1]] = c*b + (1 - c)*a;
  }
  return src;
}
```

Listing 1: Generic sort

However, we can do better if we have more information about the given PDK. In particular, if a PDK `K` provides a fast method to shuffle vectors, an efficient method for sorting can be implemented. Comparison results of shuffled vector can be declassified and control flow of the program can depend on the declassified results [15]. Sorting can be overloaded for this special case and when sorting a vector the overload resolution mechanism selects the appropriate implementation.

```
∀ D:K. sort(src : D int[]) : D int[] {
  dest : D int[] = shuffle(src);
  // Sort dest vector using public comparisons:
  // declassify(isLessThan(dest[i], dest[j]))
  return dest;
}
```

Listing 2: Specialized sort

Other features.

The SecreC language also contains a multitude of other necessary features which make the programs easier to write. Among these are the module system, multi-dimensional arrays, arithmetic operators and overload resolution. The implementation of these is orthogonal to the semantics of protection domains and not described here.

Compilation.

The compiler first resolves the polymorphism, instantiating polymorphic functions as necessary. Differently from C++, we can argue about correctness of this step as we have a well-defined semantics for our polymorphic language. The result of this step is a monomorphic program where the protection domains of all pieces of data are statically known.

At this stage, optimizations can be applied to the program. Beside the usual compiler optimizations, there can also be optimizations specific to the PDK used in a certain place of the program, or optimizations involving the change of used PDKs [9]. After optimizing, the program is further compiled to Sharemind bytecode which is executed by the engine.

SecreC is already in active use. The standard library of Sharemind written in SecreC currently consisting of ca. 10000 lines of code, and correctness tests span ca. 25000 lines.

The semantics of SecreC is described in more detail in [1].

4. ACKNOWLEDGEMENTS

This work was supported by the European Social Fund through the ICT Doctoral School programme, and by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and through the Software Technologies and Applications Competence Centre, STACC. It has also received support from Estonian Research Council through project PUT2, and from Defense Advanced Research Projects Agency (contract no. HR0011-11-9-0002).

5. REFERENCES

[1] D. Bogdanov, P. Laud, and J. Randmets. Domain-Polymorphic Programming of Privacy-Preserving Applications. Cryptology ePrint Archive, Report 2013/371, 2013. `http://eprint.iacr.org/`.

[2] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In S. Jajodia and J. Lopez, editors, *Proceedings of the 13th European Symposium on Research in Computer Security, ESORICS '08*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.

[3] M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, pages 223–240. USENIX Association, 2010.

[4] I. Damgård and J. B. Nielsen. Universally composable 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.

[5] M. Geisler. *Cryptographic Protocols: Theory and Implementation.* PhD thesis, Aarhus University, February 2010.

[6] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[7] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security. CCS'10*, pages 451–462. ACM, 2010.

[8] Y. Ishai and A. Paskin. Evaluating Branching Programs on Encrypted Data. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[9] F. Kerschbaum, T. Schneider, and A. Schröpfer. Automatic Protocol Selection in Secure Two-Party Computations. In *20th Network and Distributed System Security Symposium (NDSS)*, 2013.

[10] B. Kreuter, abhi shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security*, 2012.

[11] L. Malka. Vmcrypt: modular software architecture for scalable secure computation. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 715–724. ACM, 2011.

[12] A. C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In A. W. Appel and A. Aiken, editors, *POPL*, pages 228–241. ACM, 1999.

[13] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[14] A. C.-C. Yao. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science. FOCS'82*, pages 160–164. IEEE, 1982.

[15] B. Zhang. Generic Constant-Round Oblivious Sorting Algorithm for MPC. In X. Boyen and X. Chen, editors, *ProvSec*, volume 6980 of *Lecture Notes in Computer Science*, pages 240–256. Springer, 2011.