Privacy-aware programming language SecreC
Research Seminar in Cryptography
Roman Jagomägis
University of Tartu, Institute of Computer Science
November 2nd, 2009

1. Introduction
Every time we give away some information, we put at risk the safety of our or other people’s interests. It is important to keep secrets, but they often can be used for good intentions. There exist computer systems that are built to solve this problem.

A secure multi-party computation system called SHAREMIND [BLW08?] is able to process secret data without disclosing. It uses the additive secret scheme in the ring \(\mathbb{Z}/2^{32}\) and is proven to give a strong privacy guarantee in the honest-but-curious security model. The system resembles a hybrid virtual machine – it contains a processor unit that can sequentially and in parallel execute a number of operations on the given secret or public data. In fact, this allows us to create algorithms to describe the way the data should be processed. SHAREMIND already has an assembly programming language [Jag08?] to serve this purpose. However, complex algorithms are very difficult and tedious to program using low-level instructions.

To relieve the burden of crafting advanced algorithms we decided to design a higher level privacy-aware programming language called SecreC (pronounced as ‘secrecy’). The syntax of the language is based on C, but it omits several features and adds some new ones. Most importantly, we decided not to support pointers because of our concern about their unknown effect on the privacy of the overall system. Programs written in SecreC translate into existing SHAREMIND assembly and then run on the SHAREMIND virtual machine to accomplish data mining tasks. The design of the language in question is a compromise between various comfort and security factors. In this seminar paper we will review the main principles and features of SecreC, and discuss its good programming practices with security in mind.
2. Language principles

2.1. Data privacy and flow

Every piece of information can be a secret or common knowledge. Let us denote secrets as private data, and common knowledge as public data. It is clear that in order to guarantee the privacy of our secrets these kinds of data may not be mixed together in the public computation environment (Figure 1). Otherwise, one could witness the secrets while working with common knowledge. The flow of private data into public data domain is not limited anyhow, and this is a great security risk that has to be taken care of.

To reduce security risks there has to be a distinction between how public and private data is used. It is possible to construct an alternative hybrid execution model, where public and private data is processed in separate environments (Figure 2). The environments can be two computing systems that are specially built to handle the specific type of data. Having such a setup it is very important to limit the publishing of information derived from private data into public data domain. Although original secrets should never be published, some aggregated values such as median or a sum can usually be published without leaking too much info. For this some strict means of controlling the data flow are mandatory.
The concept of hybrid execution model is a generalization of the SHAREMIND virtual machine, which may be considered a hybrid virtual machine, that handles public data and basic flow control in the assembly interpreter (the public virtual machine) while private data is processed in the SHAREMIND processing unit (the private virtual machine) by core privacy-preserving computation protocols. The data can be transformed and exchanged between the machines using assembly instructions for all SHAREMIND protocols, the public data manipulation and flow control. While public data is stored and processed in public memory, the private data is computed on a special private stack. Instructions for core protocols read the input from the stack, execute corresponding protocols and push the results back on the stack.

The system described contains all the necessary parts for the more secure model. However, it lacks the ease of algorithm implementation and the means of data flow markup. The design of SecreC aims at simplifying the programming task and preventing the developer of privacy-preserving data mining algorithms from making trivial privacy leaks. SecreC introduces a special ‘declassify’ operator to explicitly publish the private information and allow the data flow analysis of algorithms.

Additionally, all decisions during the program flow in the hybrid execution model have to be done on the public data. Otherwise, it would be necessary to publish the observable secret value, which is a threat of a side channel attack.

2.2. Vectorization

The SHAREMIND framework is known to be more efficient when executing several operations in parallel. This is due to the internal architecture of SHAREMIND – during each operation the system gets private inputs from the private stack, and then communicates over the network to perform some computations. Generally, the less system communicates over the network, the faster it is. Intuitively, to minimize the network traffic, we have to send a smaller number of bigger packets instead of a large number of smaller packets. A bigger packet take almost the same amount of time to transfer than a smaller packet.

On the basis of these qualities it is advantageous to execute the same operation on a vector of private data rather than on single values. The extensive
use of operation vectorization is one of the main objectives in the SecreC design. The language must support intuitive and simple means of operation parallelization and allow optimization of the algorithms to take advantage of it.

3. Language features

3.1. Data types
- The main innovation in this language is the use of secret data types.

3.2. Variables and expressions
- Expressions with private subexpressions become private.
- Declassify operator.

3.3. Vectors and matrices

3.4. Functions

3.5. Branching
- if..else

3.6. Cycles
- for, while, do..while
- continue, break

4. Writing algorithms in SecreC
- how to write privacy-preserving algorithms
- declassify as little as possible, techniques for doing so
- Since there’s no branching on private conditions, an alternative could be oblivious selection.

5. Conclusion