FORMAL VERIFICATION OF SECURITY POLICIES OF CRYPTOGRAPHIC SOFTWARE

Bárbara Vieira
Departamento de Informática
barbarasv@di.uminho.pt

September 3, 2010

Keywords

Formal verification, cryptography

Abstract

In this paper we present CAOVerif, a deductive verification tool for the CAO language. CAO is a domain-specific language for cryptography with interesting challenges for formal verification. It introduces not only a rich mathematical type system, but it also offers cryptography-oriented language constructions. The toolchain encompasses different transformations of the source code in order to get the VCs and is based in the Jessie plug-in of the Frama-C framework.

INTRODUCTION

Program verification refers to techniques aiming at establishing that a program is in accordance with its specification. These techniques usually rely on verification platforms, which can be used to prove complex program properties. Often these are based on variations of *Hoare logic* [6], whose axioms and inference rules capture the semantics of imperative programming languages. Such verification platforms are usually structured around the following components: an *annotation Language*, a *Verification condition generator (VCGen)* and *proof generation*.

In this report we describe a deductive verification tool for CAO. This is a language tuned to the implementation of low-level cryptographic primitives in a way which is close to the notation used in scientific papers and standards. In a nut-shell, CAO is an imperative language that includes the typical set of imperative statements, but offers a very rich set of native types that are used recurrently in cryptography. The idea is to take advantage of the characteristics of this programming language to construct a domain-specific verification tool, allowing for the same verification techniques that can be applied over general-purpose languages such as C, but simplifying the verification of security-relevant properties, and hopefully providing a higher degree of automation.

Experience in applying general purpose verification tools to cryptographic software shows that there is a great potential for improvement in this regard [1, 2], and justifies the motivation for this work.

CAOVERIF

Frama-C [3], is verification framework for `C` programs based on Hoare Logic. Frama-C provides static analyzers implemented as plug-ins, granting a fine-grained collaboration of analysis techniques. The weakest precondition plug-in, called Jessie, allows proving that `C` functions satisfy their specification as expressed in ACSL (a specification language based on `JML`[7]). The Jessie plug-in is built on the top of the Why tool, which is a VCGen that supports a large number of provers as back-end. From an annotated Why program, the tool generates a set of verification conditions that can be discharged by a series of proof assistants [8] and automatic provers [4, 5].

We build on the Frama-C framework, and build CAOVerif on top of the Jessie plug-in to reduce development time. We take advantage of the fact that Jessie already incorporates many features to reason about program correctness, such as a first-order logic mechanism that facilitates the design of language extensions. Based on this mechanism, we were able to rapidly develop a appropriate models of CAO primitive types and memory handling mechanisms. The architecture for the CAO deductive verification tool is shown in Figure 1. An annotated CAO program is first checked for syntax and type consistency and is then translated into the Jessie input language by the `CAO2Jessie` tool. The corresponding code in the Why input language is then generated by running the Jessie tool. A multitude of proof assistants and automatic provers can then be used to discharge the proof obligations which are generated by the Why VCGen.

Because we are building on existing tools, we could fo-

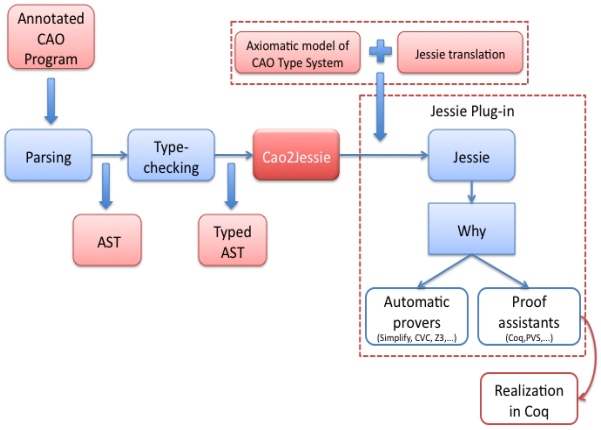

Figure 1: Tool architecture

cus our attention on the CAO-to-Jessie translation and, in particular, on the Jessie *model* that captures the semantics of CAO programs. Here we benefit from the considerable amount of work that has already been carried out to support the translation of C programs into Why. In particular, most of the standard imperative language constructions supported by CAO can be directly translated into equivalent Jessie constructions. Most CAO types are not native Jessie types, and so our translation includes a model (i.e. an incomplete axiomatization in first-order logic) of the CAO type system. The first-order theory created for each CAO type typically comprises a type signature and a set of axioms that partially describe the operations allowed over that type. It would be pointless to include a complete axiomatization in first-order logic of the semantics of the entire CAO type-system, particularly for the most complex types dealing with mathematical types such as rings and finite fields, as these would be of little use to off-the-shelf automatic provers. Instead we propose to refine our models in Coq, over which we can show that our axiomatization is consistent, and where users can interactively discharge more intricate verification conditions. The translation of CAO programs into Jessie also ensures the automatic generation of verification conditions to validate the safe execution of the verified code. Program safety in CAO has two dimensions: memory safety and safety in arithmetic operations. Memory safety is reduced to making sure that all indices used in vector, bit-string and matrix accesses are within the proper range. The safety of arithmetic operations implies making sure that all expressions refer to well-defined mathematical computations.

WORK-PLAN

As future work, we intend to implement a dedicated prover over Coq that will provide support for CAO types where our current models have clearer limitations. We also aim to establish the correctness of CAOVerif, to guarantee that our *VCGen* generates proof-obligations that indeed provide assurance as to the input CAO program's safety and correctness.

References

[1] J. B. Almeida, M. Barbosa, J. S. Pinto, and B. Vieira. Verifying cryptographic software correctness with respect to reference implementations. In M. Alpuente, B. Cook, and C. Joubert, editors, *FMICS*, volume 5825 of *Lecture Notes in Computer Science*, pages 37–52. Springer, 2009.

[2] J. B. Almeida, M. Barbosa, J. S. Pinto, and B. Vieira. Deductive verification of cryptographic software. *NASA Journal of Innovations in Systems and Software Engineering*, 2010.

[3] P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. *ACSL: ANSI/ISO C Specfication Language*. CEA LIST and INRIA, 2008. Preliminary design (version 1.4, December 12, 2008).

[4] S. Conchon, E. Contejean, and J. Kanig. Ergo : a theorem prover for polymorphic first-order logic modulo theories, 2006.

[5] L. de Moura and N. Bjørner. *Z3: An Efficient SMT Solver*, volume 4963/2008 of *Lecture Notes in Computer Science*, pages 337–340. Springer Berlin, April 2008.

[6] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.

[7] G. T. Leavens, C. Ruby, K. R. M. Leino, E. Poll, and B. Jacobs. JML (poster session): notations and tools supporting detailed design in Java. In *OOPSLA '00: Addendum to the 2000 proceedings of the conference on Object-oriented programming, systems, languages, and applications (Addendum)*, pages 105–106, New York, NY, USA, 2000. ACM.

[8] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.2*, 2008. `http://coq.inria.fr`.

AUTHOR BIOGRAPHY

BÁRBARA VIEIRA was born in Braga, Portugal and went to Universidade do Minho where she studied Matemática e Ciências da Computacão in 2002. She started the PhD in Computer Science in 2008 being supervised by Manuel Bernardo Barbosa. She also visited INRIA Saclay - Île de France in 2009 for 3 moths to work with Jean-Christophe Filliâtre and Claude Marché.