Verifying C Cryptographic Protocol Implementations by Symbolic Execution

Supervisors: Cristian Ene* Laurent Mounier†

October 2015

1 Context

Many applications rely on complex cryptographic protocols for communicating over the insecure Internet (e.g., online banking, electronic commerce, social networks, etc). The C programming language is largely used in writing cryptographic software. Both the design of protocols and their C implementation are error prone. Recent years have seen a real progress in the formal verification of cryptographic protocols as illustrated by the development of several tools both in the symbolic model (Proverif [Bla01], AVISPA [ABB+05], Hermes [BEJ+05]) and in the computational model (CryptoVerif [Bla08], CertiCrypt [BGZB09], EasyCrypt [BGHB11]). There remains however a large gap between what we verify (the protocol usually described in a process algebra, as pi calculus for example) and what we rely on (the implementation which is usually done in a «real» language, like C).

The need to verify the code is now well recognized, but only a few recent works try to propose solutions.

One of the first attempts at cryptographic verification of C code is the CSur ([GLP05]): one extracts from a C program a set of Horn clauses that are then solved using a theorem prover. Some limits of this approach: 1) it can be used to prove secrecy, but it is not clear how one can apply the tool to handle authentication properties (the order of instructions is completely ignored); 2) the results that are obtained are sound only in the symbolic (Dolev-Yao) model of cryptography.

A second line of research, based on symbolic execution ([Kin76]), is that of [CM11], [CM12]: [CM11] extends the KLEE test-generation tool ([CDE08]) by treating certain concrete functions, like cryptographic primitives, as symbolic functions, that is, their execution is avoided, and their behaviour is modelled via rewriting rules. However, their work does not extend the class of properties supported by KLEE, in particular they do not take into account how the inputs provided by an adversary depend on the knowledge learnt by the same adversary. [CM12] extends the previous work with a tainting mechanism that tracks information flows of data, but this work suffers from several limitations:

*Cristian.Ene@imag.fr
†Laurent.Mounier@imag.fr

it cannot handle authentication properties, and it is done only in the Dolev-Yao model.

Another work based on symbolic execution is that of [AGJ11]. The authors first use the CIL ([NMRW02]) tool to compile the C program down to CVM (a simple stack-based instruction language). Then, they symbolically execute the CVM program in order to eliminate memory accesses and destructive updates, and get an equivalent program in an intermediate model language (IML), actually a version of applied pi calculus enriched with bitstring manipulation operations. Then, they abstract the bitstring operations, and get a process in the fragment of applied pi calculus supported by ProVerif (Proverif [Bla01]), and hence they are able to use ProVerif for checking some security properties in the symbolic model. In a subsequent work [AGJ12], the authors changed the last part of their approach: they translate the extracted IML program to a CryptoVerif protocol description, such that a successful verification with CryptoVerif implies the security of the original C implementation in the computational model. One limitation of this work is that their current method and prototype can analyze only a single execution path, so it is limited to programs/protocols with no significant branching.

2 Goals

The goal of this thesis is to develop a method and a prototype tool in order to verify security properties of C code that uses cryptographic primitives. The objective is to overcome the limitations of the existing methods. More specifically, the aim is to develop a method that starting with a C program (possibly with branching and loops), first extracts an equivalent program in a higher level model, and then use an existing tool in order to verify for security properties.

The work can be organized as follows:

- compile the C program down to a simpler language (CVM or LLVM [LA04] for example)
- define an appropriate intermediate language (denoted here by IL) and give an automatic method based on symbolic execution that allows to extract an equivalent IL program from the compiled version of the initial C program
- give an automatic method to extract an equivalent model EM from the above obtained IL program, such that EM can be used as input for an existing tool (if possibly, EasyCrypt or CryptoVerif in order to get computational soundness)
- implement the method in a prototype tool.

3 Bibliography

References

[ABB+05] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-

Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification*, pages 281–285. Springer, 2005.

[AGJ11] Mihhail Aizatulin, Andrew D Gordon, and Jan Jürjens. Extracting and verifying cryptographic models from c protocol code by symbolic execution. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 331–340. ACM, 2011.

[AGJ12] Mihhail Aizatulin, Andrew D Gordon, and Jan Jürjens. Computational verification of c protocol implementations by symbolic execution. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 712–723. ACM, 2012.

[BEJ+05] Liana Bozga, Cristian Ene, Romain Janvier, Yassine Lakhnech, Laurent Mazaré, and Michaël Périn. Automatic verification of security properties based on abstractions. In *Proceedings of the NATO Advanced Research Workshop Verification of Infinite State Systems with Applications to Security VISS*, volume 1 of *NATO Security through Science Series D: Information and Communication Security*, pages 23–53. IOS Press, 2005.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology–CRYPTO 2011*, pages 71–90. Springer, 2011.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. *ACM SIGPLAN Notices*, 44(1):90–101, 2009.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *csfw*, page 0082. IEEE, 2001.

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *Dependable and Secure Computing, IEEE Transactions on*, 5(4):193–207, 2008.

[CDE08] Cristian Cadar, Daniel Dunbar, and Dawson R Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, volume 8, pages 209–224, 2008.

[CM11] Ricardo Corin and Felipe Andrés Manzano. Efficient symbolic execution for analysing cryptographic protocol implementations. In *Engineering Secure Software and Systems*, pages 58–72. Springer, 2011.

[CM12] Ricardo Corin and Felipe Andrés Manzano. Taint analysis of security code in the klee symbolic execution engine. In *Information and Communications Security*, pages 264–275. Springer, 2012.

[GLP05] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real c code. In *Verification, Model Checking, and Abstract Interpretation*, pages 363–379. Springer, 2005.

[Kin76] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[LA04] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*, pages 75–86. IEEE, 2004.

[NMRW02] George C Necula, Scott McPeak, Shree P Rahul, and Westley Weimer. Cil: Intermediate language and tools for analysis and transformation of c programs. In *Compiler Construction*, pages 213–228. Springer, 2002.