A Proof Technique for Noninterference In Open Systems: An extended version (Extended Abstract)

Enrico Scapin
University of Trier, Germany

In [3], a framework has been proposed which allows tools that can check standard noninterference properties but a priori cannot deal with cryptography, in particular probabilities and polynomially bounded adversaries, to establish cryptographic indistinguishability properties, such as privacy properties, for Java programs. The framework combines techniques from program analysis and cryptography, more specifically, universal composability [1], a well-established concept in cryptography. The idea is to first check noninterference properties for the Java program to be analyzed where cryptographic operations (such as encryption) are performed within so-called ideal functionalities. Such functionalities typically provide guarantees even in the face of unbounded adversaries and can often be formulated without probabilistic operations and, therefore, they can be carried out by tools that a priori cannot deal with cryptography.

At the core of the framework, there are results linking the notion of (termination-insensitive) noninterference [4] with the notion of cryptographic indistinguishability: in order to assert that two systems using cryptographic operations are computational indistinguishable, it is enough to show that these systems are noninterferent when the cryptographic operations are replaced by their corresponding ideal functionalities.

As to checking non-interference, many program analysis tools can only deal with closed Java programs. The systems to be analyzed are, however, often open: they interact with a network or use some libraries which are not necessarily trusted and, hence, are not part of the code to be analyzed; instead, they are considered as part of the environment with unspecified behavior. Therefore, [3] intruduces the notion of *noninterference in an open system*, i.e., in a system not completely defined: An open system S is noninterferent if for each environment E this system can be composed with, the resulting close system $S \cdot E$ is also noninterferent. As part of the framework, a *proof technique* was proposed to reduce the problem of checking noninterference in an open system to checking noninterference for a single (almost) closed system. Technically, this result shows how to construct, for an open system S, a family of environments $\tilde{E}_{\vec{u}}$ parametrized by an input sequence $\vec{u}$, such that S is noninterferent if and only if S composed with $\tilde{E}_{\vec{u}}$ is noninterferent for all $\vec{u}$. Importantly, the latter property can be verified using existing tools for program analysis.

Our Contribution. The framework is formulated for a language called Jinja+ and is proven w.r.t. the formal semantics of this language. Jinja+ is a Java-like language that extends the language Jinja [2] with, among others, arrays, the type `byte`, and the `abort` primitive. In this work, we further extend its syntax and semantics with: (a) java-interfaces, (b) abstract classes, (c) strings. Except for the result discussed below, all definitions and results of the framework carry out easily to the extended language. That is, the new types of values and the new rules of the augmented small-step semantics do not affect the proofs in a significant way.

One result, however, namely the proof technique for proving noninterference in open systems, required non-trivial modifications to model the exchange of data between the system and the environment when also strings are involved. In particular, the exchange of data through string references introduces subtle changes in the original result and, technically, invalidates the main assumption the result is based on, i.e., the separation between the state of the system and the state of the environment. Therefore, we extend the construction of $\tilde{E}_{\vec{u}}$ to handle exchange of string references. Furthermore, relying on the fact that the Java (Jinja+) strings are *immutable*, we relax the state separation assumption and adopt the proof in a non-trivial way to work with the new (relaxed) assumption. Based on this premise, we then reshaped the proof technique for proving noninterference in open systems taking into account string references, too. We refer the reader to [5] for all the details.

References

[1] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001).*

[2] Gerwin Klein and Tobias Nipkow. A Machine-Checked Model for a Java-Like Language, Virtual Machine, and Compiler. *ACM Trans. Program. Lang. Syst.*, 2006.

[3] Ralf Küsters, Tomasz Truderung, and Jürgen Graf. A Framework for the Cryptographic Verification of Java-like Programs. In *25th IEEE Computer Security Foundations Symposium (CSF 2012).*

[4] Andrei Sabelfeld and Andrew C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications, special issue on Formal Methods for Security*, 2003.

[5] Enrico Scapin. A proof technique for noninterference in open systems: An extended version. Available at http://infsec.uni-trier.de/publications/paper/Scapin-Inoninterference-ext-2015.pdf.