INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team EVEREST

Environnements de vérification et sécurité du logiciel

Sophia Antipolis

THEME SYM

Activity Report

2004

Table of contents

1. Team

Head of project-team
Gilles Barthe [Senior research scientist INRIA]

Vice-head of project team
Marieke Huisman [Research scientist INRIA]

Administrative assistant
Nathalie Bellesso

Scientific advisor
Jean-Louis Lanet [INRIA DirDRI]

Software development staff
Lilian Burdy [until May 2004, now at Clearsy]

Ph. D. students
Néstor Cataño [until September 2004, now at U. of York]
Guillaume Dufay [until June 2004, now at U. of Ottawa]
Mariela Pavlova
Tamara Rezk
Sabrina Tarento

Post-doctoral fellows
Florian Kammüller [since April 2004]
Sylvain Lippi [since November 2004]
Igor Siveroni [until February 2004, now at Imperial College London]

Student internships
Fernando Pastawski [Cordoba University, Internship Program, 4 monthes]
Pratik Worah [IIT Kharagpur, Internship Program, 2.5 monthes]
Santiago Zanella [Rosario University, 5 monthes]

2. Overall Objectives

The Everest project aims at ensuring security for mobile and embedded applications. Our privileged application domains are small, trusted and portable devices, such as mobile phones and smart cards, their operating systems and the applications running on these devices.

In a more general sense, our work concentrates on the development and validation of secure software, both at platform and application level. To achieve this, we advocate the use of formal methods and language-based techniques for safe and secure programming.

The project focuses on the following research areas: Specification and verification of components; Secure loading of components; Platform certification.

3. Scientific Foundations

3.1. Type systems

Types are often considered as one of the great successes of programming language theory, and their use permeates modern programming languages. The widespread use of types is a consequence of three crucial factors:

- *Types are intuitive*. Types are a particularly simple form of assertion. They can be explained to the user without the need to understand precise details about why and how they are used in order to achieve certain effects. For example, Fortran and C use type systems to generate memory layout directives at compile time; yet the users of C can write type-correct programs and understand typing errors without knowing exactly how the type concept is related to memory layout.

- *Types are automatic*. In many cases an untyped program can be enhanced with types automatically by type inference. Of course, adherence of a program to even the simplest policy is an algorithmically undecidable property. Type systems circumvent this obstacle by guaranteeing a safe over-approximation of the desired policy. For example, a branching statement with one unsafe branch will usually be considered unsafe by a type system. This not only restores decidability but contributes to the aforementioned intuitiveness and simplicity of type systems.
- *Types scale up*. Besides their simplicity and the possibility to infer types, type systems allow to reduce the verification of a complex system into simpler verification tasks involving smaller parts of the system. Such a compositional approach is a crucial property for making the verification of large, distributed and reconfigurable systems feasible.

Type systems have long been used in programming languages to enforce basic safety properties of programs. For example, the type system of Java is designed to statically detect certain runtime errors such as the application of a string function to a floating point number, or a call to a method that does not belong to a name space of a given class. In addition, the research community has developed numerous type systems that enforce more advanced safety properties dealing with modularity, concurrency... For example, there have been several proposals of type systems to control aliasing or detect race conditions in Java programs.

Type systems are also increasingly being studied as a means to enforce security; in particular, the research community has developed type systems that guarantee information flow and resource control policies.

3.2. Program verification

Research on program logics has a long history, dating back to the seminal work on Floyd-Hoare logics and weakest precondition calculi in the late 1960s and early 1970s. Although this line of research has not yet lead to a breakthrough in the application of program verification, there has been steady progress, resulting in tool-supported program logics for realistic programming languages.

There are a number of reasons to adopt program verification techniques based on logic to guarantee the correctness of programs.

- *Logic is expressive*. During its long development, logic has been designed to allow for greater and greater expressiveness, a trend pushed by philosophers and mathematicians. This trend continues with computer scientists developing still more expressiveness in logic to encompass notions of resources and locality. Today a rich collection of well developed and expressive logics exist for describing computational systems.
- *Logic is precise*. While types generally over-approximate program behavior, logic can be used to provide precise statements of program behavior. Special conditions can be assumed and exceptional behaviors can be described. Via the use of negation and rich quantifier alternation, it is possible to state nearly arbitrary observations about programs and computational systems.
- *Logic allows to combine analyses*. Logic provides a common setting into which the declarative content of a typing judgment or other static analyses can be translated. The results of such analyses can then be placed into a common logic so that conclusions about their combinations can be drawn.

Recently, there has been a lot of research into program verification of Java, which has culminated in the realization of program verification environments for single-threaded Java.

3.3. Machine-checked semantics and algorithms

Keywords: *cryptographic algorithms*, *proof-assistants*, *semantics*.

We are interested in developing formal, machine-checked semantics of programming languages and of their execution platforms such as virtual machines, run-time environments, Application Programming Interfaces, and of the tools that are used for compiling, verifying, validating programs. In particular, we have strong experience in modeling and verifying execution platforms for smart cards, as well as their main security functions, such as bytecode verifiers and access control mechanisms, and the standard deployment architectures for multi-application smartcards, such as Global Platform.

3.4. Software security

Security is not a technology, but a property of a system. For this reason, there are new security requirements associated with each new technology or architecture. While these security requirements are often expressed from the perspective of the users, they must also be translated to concrete objectives that can be enforced by security mechanisms. For instance, the Java security model assumes that end users trust its runtime environment but not the downloaded code. The concrete objectives include adherence to the typing and policy of the JVM and compliance with the stack inspection mechanism, which are enforced by the Java byte code verifier and the runtime environment.

The ability to derive concrete verification objectives from carefully gathered security requirements is an important step for guaranteeing that a component is secure with respect to a given security policy. Typical requirements include:

- information flow security policies;
- resource control policies;
- framework-specific security; and
- application-specific security.

3.5. Proof-Carrying Code

Keywords: *certificates*, *proof-checker*, *verification condition generation*.

Proof Carrying Code (PCC) is an innovative security framework in which components come equipped with a certificate which can be used by devices (code consumers in PCC terminology) to verify locally and statically that downloaded components issued by an untrusted third party (code producers in PCC terminology) are correct. In order to realize this view, standard PCC infrastructures build upon several elements: a logic, a verification condition generator, a formal representation of proofs, and a proof checker.

- *A formal logic for specifying and verifying policies*. The specification language is used to express requirements on the incoming component, and the logic is used to verify that the component meets the expected requirements. Standard PCC adopts first-order predicate logic as a formalism to both specify and verify the correctness of components, and focuses on safety properties. Thus, requirements are expressed as pre- and post-conditions stating, respectively, properties to be satisfied by the state before and after a given procedure or function is invoked.
- *A verification condition generator (VCGen)*. The VCGen produces, for each component and safety policy, a set of proof obligations whose provability will be sufficient to ensure that the component respects the safety policy. Standard PCC adopts a VCGen based on programming verification techniques such as Hoare-Floyd logics and weakest precondition calculi, and it requires that components come equipped with extra annotations, e.g., loop invariants that make the generation of verification conditions feasible.

- *A formal representation of proofs (Certificates).* Certificates provide a formal representation of proofs, and are used to convey to the code consumer easy-to-verify evidence that the code it receives is secure. In Standard PCC, certificates are terms of the lambda calculus, as suggested by the Curry-Howard isomorphism, and routinely used in modern proof assistants such as Coq.
- *A proof checker that validates certificates against specifications.* The objective of a proof checker is to verify that the certificate does indeed establish the proof obligations generated by the VCGen. In Standard PCC, proof checking is reduced to type checking by virtue of the Curry-Howard isomorphism. One very attractive aspect of this approach is that the proof checker, which forms part of the Trusted Computing Base is particularly simple.

We study other variants of Proof Carrying Code that cover a wide range of security properties that escape the scope of certifying compilers, and that need to be established interactively on source code programs.

4. Application Domains

4.1. Smart devices

Keywords: *mobile phones*, *smart cards*.

Smart devices, including new generation smartcards and trusted personal devices typically contain a microprocessor and a memory chip (but with limited computing and storage capabilities). They are often used by commercial and governmental organizations and are expected to play a key role in enforcing trust and confidence in e-payment and e-government. Current applications include bankcards, e-purses, SIM cards in mobile phones, e-IDs, *etc*,

These devices provide solutions for card application developers by enabling them to program in high-level languages (several dialects of Java exist for this purpose: Java Card, MIDP, *etc.*), on a common base of software (an abstract machine and Application Programming Interfaces, APIs), which isolates their code from specific hardware and operating system libraries. These devices support both the flexibility and the evolution of applications by enabling downloading of executable content onto already deployed smart devices (so-called post issuance), and by allowing several commercially independent applications to run on a single smart card. This open character forms its commercial strength, but also creates the technical challenges of these devices: reliability, correctness and security become crucial issues, since malicious applets might potentially exploit bugs in the smart card platform, with detrimental effects on security and/or privacy.

4.2. Global computing

Keywords: *JVM-enabled devices*, *networks*.

A global computer is a *distributed* computational infrastructure that aims at providing a global and uniform access to services. However, global computers consist of large networks of *heterogeneous* devices that differ greatly in their computational infrastructure and in the resources they offer to services. In order to deliver services globally and uniformly, each device in a global computer must therefore be *extensible* with the computational infrastructure, platform or libraries, needed to execute the required services. In that respect, global computers transcend the scope of established computational models such as mobile code, the Grid, or agents, which impose a clear separation between mobile applications and the fixed computational infrastructure upon which they execute.

While global computers may deeply affect our quality of life, security is paramount for them to become pervasive infrastructures in our society, as envisioned in ambient intelligence. Indeed, numerous application domains, including e-government or e-health, involve sensitive data that must be protected from unauthorized parties. In spite of clear risks, provisions to enforce security in global computers remain extremely primitive. Some global computers, for instance in the automotive industry, choose to enforce security by maintaining devices completely under the control of the operator. Other models, building upon the Java security architecture, choose to enforce security via a sandbox model that distinguishes between a fixed trusted computing base and untrusted applications. Unfortunately, these approaches do not embrace the complexity of global computers.

5. Software

5.1. Tool for applet validation

Participants: Gilles Barthe, Lilian Burdy, Marieke Huisman [correspondent], Jean-Louis Lanet, Mariela Pavlova.

We develop a tool for applet validation. The tool takes Java or JVM programs annotated with JML specifications (or JML-like specifications for JVM programs), and generates appropriate proof obligations which can be used as input for different proof assistants (automatic or interactive).

In addition, the tool includes a compiler that takes as input a Java applet annotated with JML annotations, and generates the corresponding bytecode program and JML-like annotations for the bytecode. The next step is to provide an explicit representation of proofs, a.k.a. proof objects, and extend the compiler to proof objects in order to offer support for Proof Carrying Code.

The development of the tool is partly done in the context of an ODL (Software Development Support project) on generation of proof obligations and interactive proof systems.

5.2. Jakarta

Participants: Gilles Barthe [correspondent], Pierre Courtieu, Guillaume Dufay.

We develop a tool for specifying and verifying formally execution platforms, and more particularly bytecode verifiers, as they exist e.g. in the Java Platform.

The tool, which instruments a two-phase methodology that is common to many existing works, intends to provide a very high-level of automation for the mundane tasks inherent to the methodology (namely deriving a virtual machine from another by abstraction techniques, and proving the correctness of the abstraction), and has been used successfully for certifying the correctness of the JavaCard bytecode verifier.

The development of the tool was initiated in the context of the european project Verificard and is partly done in the context of the RNTL project CASTLES which focuses on the certification of execution platforms for smartcards.

6. New Results

6.1. Applet specification and verification

Participants: Gilles Barthe, Lilian Burdy, Marieke Huisman, Jean-Louis Lanet, Igor Siveroni, Kerry Trentelman [ANU, Australia].

The JACK tool has been improved on many points. The correctness of the generated lemmas has been improved concerning notably the treatment of exceptional behaviours. Two plugins have been added to generate proof obligations in Coq and PVS. Further, the automated security audit procedure (developed in 2003), which aims at generating automatically JML annotations from high-level security properties, has been integrated in JACK, and its correctness has been studied formally [20][26]. Also, a tool has been developed to generate automatically some method preconditions concerning modified variables and run-time exceptions. Finally, the JACK tool is also used as a starting point in our work on Proof Carrying Code.

We have developed a technique to reduce the verification of a temporal property for a multi-threaded Java application to the verification of the temporal property for parts of the application [24].

We have pursued our work on compositional verification of applet interactions. In [21], we propose a maximal model construction that supports secure post-issuance loading of applets, as illustrated in [17].

In [23], the method is extended to incorporate the use of public interfaces as a means of encapsulating method implementations, thereby improving significantly the efficiency of our method.

Finally, we have completed two studies on the use of JML-based formal methods for smartcards [9][22].

6.2. Security type systems

Participants: Gilles Barthe, Amitab Basu, David Naumann [Stevens Institute of Technology, USA], Tamara Rezk, Martijn Warnier [University of Nijmegen, Netherlands].

We have defined an information flow type system for a sequential JVM-like language that includes classes, objects, and exceptions. Furthermore, we show that it enforces non-interference. Our work provides, to our best knowledge, the first analysis that has been shown to guarantee non-interference for a realistic fragment of the JVM [16].

An important issue is to ensure that security type systems for low-level languages relate appropriately to their counterparts for high-level languages. We have proven that compilation preserves security typing for a simple imperative language [10][7]. In collaboration with David Naumann, we are extending this result to a richer fragment of Java.

These works deal with a weak form of non-interference that does not consider covert channels, such as resource consumption (memory/cpu) and timing behavior. In collaboration with Martijn Warnier, we have introduced a program transformation method that uses transactions for preventing timing leaks in sequential object-oriented programs.

6.3. Security by logic

Participants: Gilles Barthe, Pedro D'Argenio [University of Marseille and Cordoba], Marieke Huisman, Tamara Rezk, Pratik Worah [IIT Kharagpur].

While information flow type systems provide a common means to enforce non-interference, such type systems are inherently imprecise, and reject many secure programs, even for simple programming languages. We have investigated logical formulations of non-interference that allow a more precise analysis of programs. It appears that such formulations are often sound and complete, and also amenable to interactive or automated verification techniques, such as theorem-proving or model-checking [12]. We have also investigated logical formulations of observational determinism by Myers and Zdancewic.

6.4. Proof Carrying Code

Participants: Gilles Barthe, Lilian Burdy, Mariela Pavlova, Tamara Rezk, Ando Saabas [Tallinn Technical University].

Program verification techniques provide a means to guarantee that source code programs are correct with respect to a formal specification. However, the benefits of source code verification are not immediately exploitable in the context of mobile code, in which the code consumer downloads a compiled program originating from an untrusted code producer. The goal of our work is to show how to bring the benefits of program verification techniques to the code consumer, by proposing a proof-carrying code (PCC) architecture in which the code producer writes, together with the program, a specification and a proof that the program meets the specification. Then, the program, specification, and proofs are compiled and sent to the code consumer that can verify them.

In order to illustrate our ideas, we have defined a weakest precondition calculus for a simple assembly language (SAL), demonstrated its correctness w.r.t. operational semantics, and studied its relationship with a weakest precondition calculus for a simple imperative language (SIL) that can be compiled into SAL. We have shown that the proof obligations generated from the source program are syntactically equivalent to those generated from the assembly program obtained by compiling (without optimizations) the original source program. Current work aims at adapting these results to optimizing compilation.

We have also initiated the development of a Proof Carrying Code tool on top of JACK: the tool can be used to validate compiled Java applications that are annotated with JML-like specifications. The tool takes as input a class file annotated with JML-like annotations, and generates thanks to a weakest precondition calculus appropriate proof obligations which can be used as input for different proof assistants (automatic or interactive). In addition, we have developed a compiler that takes as input a Java applet annotated with JML annotations, and generates the corresponding bytecode program and JML-like annotations for the bytecode.

6.5. Machine-checked semantics

Participants: Gilles Barthe, Guillaume Dufay, Florian Kammüller, Jean-Louis Lanet, Leonor Prensa [LORIA], Santiago Zanella [University of Rosario, Argentina].

We have completed our models of bytecode verification [13], and are applying our methodology to prove formally the correctness of information flow type systems. To this end we have developed an abstract proof of non-interference in Coq, and are instantiating the proof to different fragments of the JVM. We have also pursued our work on machine-checked proofs of information flow type systems for high-level concurrent languages [15].

We have also developed a formal model in B of the GlobalPlatform Card Specification v2.1.1. The formal model provides an abstract reference implementation of the card specification, and will potentially facilitate GlobalPlatform's future development of functional tests for the specification by automating the generation of tests [25].

6.6. Verification of cryptographic algorithms

Participants: Gilles Barthe, Jan Cederquist, Sabrina Tarento.

We have pursued our work on the formalization of the generic model (GM) and the random oracle model (ROM) in COQ. The aim of this work is to verify the correctness of cryptographic algorithms, without making the perfect cryptography assumption.

In [11], we have shown an upper bound to the probability of a non-interactive adversary (that adheres to the GM) to find a certain secret. We have also extended this result to the case of an interactive adversary (that adheres to the GM and to the ROM), and are currently considering the case of parallel attacks.

6.7. Type theory

Participants: Gilles Barthe, Benjamin Grégoire, Fernando Pastawski.

We have studied type isomorphisms in dependent type theory [6], and inconsistent pure type systems [8].

We have pursued our work on type-based termination, which enforces termination of recursive definitions through an extended type system in which types carry information about the respective size of input and output of functions. The main new result is an algorithm for automatic size inference, so that users do not need to provide any size information on terms [14]. We have also implemented a prototype of type-based termination for the Calculus of Inductive Constructions.

7. Contracts and Grants with Industry

7.1. Castles

RNTL Castles (Development of Static Analyses and Test for Secure Embedded Software, accepted in 2003, started January 2004). Other participants are Lande (Rennes), AQL and Oberthur. More information is available via http://www-sop.inria.fr/everest/projects/castles/. The goal of the project is to define an environment to support the formal specification and verification of low-level execution platforms.

8. Other Grants and Activities

8.1. International collaborations

- PAI Parrot with the University of Tallinn in Estonia. Collaboration on (co-)inductive types, modal logics, type theory, proof-carrying code.
- SICS and KTH, Stockholm, Sweden: compositional verification of control-flow security properties for applets.
- University of Nijmegen, Netherlands and ANU, Canberra, Australia: specification languages for Java (extending of JML towards high-level security properties).
- Stevens Institute of Technology, and Kansas State University, USA: language-based security.
- GlobalPlatform: formal models of the GlobalPlatform specifications.

8.2. National initiatives

- ACI Sécurité GECCOO (Generation of Certified Code for Object-Oriented Applications - Specifications, refinement, proof and error detection, 2003 - 2006). Other participants are TFC (Besançon), Cassis (Nancy), LogiCal (LRI/Futurs) and Vasco (IMAG, Grenoble). For more information, see http://geccoo.lri.fr/.
- ACI Sécurité SPOPS (Secure Operating Systems for Trusted Personal Devices, 2003 - 2006). Other participants are POPS (Lille) and SSIR (Supélec, Rennes). For more information, see http://www-sop.inria.fr/everest/projects/spops/.

8.3. European initiatives

8.3.1. INSPIRED

We participate in the European IST project Inspired (December 2003 - December 2006). The partners in this project are all major industrial actors in the domain of smart cards, INRIA (also the projects Metiss and POPS) and the Universities of Louvain (Belgium) and Twente (Netherlands). The goal of the project is to define the new generation of Trusted Personal Devices. The role of INRIA is to develop appropriate formal methods for those.

8.3.2. Thematic Networks

EVEREST participates in the networks Types (type theory) and Appsem (Applied Semantics).

8.3.3. Other european projects

EVEREST participates in the EU Global Computing 1 project PROFUNDIS.

9. Dissemination

9.1. Conference and workshop attendance, travel

- Gilles Barthe participated to ETAPS 2004, CSFW 2004 and PLID 2004. He visited Stevens Institute of Technology, Hoboken, USA, in July 2004.
- Guillaume Dufay presented his work at ETAPS 2004. He visited the University of Ottawa in March 2004.
- Florian Kammüller visited Technische Universitaet Berlin, Germany, in July 04 and Oxford University, United Kingdom, in December 04.

- Marieke Huisman presented her work at ETAPS 2004 and participated to ECOOP 2004 and FTfJP 2004.
- Jean-Louis Lanet presented his work at the International Workshop on Formal Method and Security, Nanjing, China. He also participated to E-Smart 2004.
- Mariela Pavlova presented her work at CARDIS 2004. She participated in the Marktoberdorf summer school, Germany.
- Tamara Rezk presented her work at VMCAI 2004, CSFW 2004, CASSIS 2004, and participated to PLID 2004. She visited the University of Verona in April 2004, the University of Tallinn in April and December 2004, Kansas State University, USA, in May and June 2004, Stevens Institute of Technology, Hoboken, USA, in July 2004.
- Sabrina Tarento presented her work at a DIMACS Workshop on Security Analysis of Protocols, at IJCAR 2004, and at TYPES 2004. She participated in the summer school EJCP 2004.

9.2. Leadership within scientific community

- Sabrina Tarento and Tamara Rezk took care of the organization of the seminar.
- Gilles Barthe and Lilian Burdy and Marieke Huisman and Jean-Louis Lanet and Traian Muntean (Marseilles) co-organized the International Workshop on Construction and Analysis of Safe, Secure and Interoperatble Smart Devices, Marseilles, March 2004. The proceedings will appear as Volume 3362 of Lecture Notes in Computer Science.
- Gilles Barthe was a member of the program committee for AMAST 2004, FCS 2004, FMICS 2004 and LOPSTR 2004.
- Marieke Huisman was a member of the program committee for FTfJP 2004, and .NET 2004.
- Jean-Louis Lanet was member of the program committee for CARDIS 2004. He was expert for the Austrian Research Agency IT-FIT for evaluating security projects.
- Gilles Barthe, Peter Dybjer (Göteborg) and Peter Thiemann (Freiburg) co-edited a special issue of Journal of Functional Programming on Dependent Type Theory meets Programming Practice.

9.3. Visiting scientists

We had several visiting scientists, many of whom gave a talk in our seminar. Ando Saabas (Tallinn University) and Martijn Warnier (Nijmegen University) both visited for several weeks.

9.4. Supervision of Ph.D. projects

- Gilles Barthe supervises the Ph.D. project of Maria João Frade (U. Minho, Portugal, completed March 2004), Mariela Pavlova, Tamara Rezk and Sabrina Tarento.
- Marieke Huisman supervised the Ph.D. project of Nestor Cataño, completed November 2004.

9.5. Ph.D. committees

- Gilles Barthe was member of the jury for the theses of Jean-Louis Lanet, Liana Lazar, and Ricardo Medel (USA).
- Jean-Louis Lanet was member of the jury for the thesis of Jean-Christophe Voisinet, Besancon and for the thesis of Pierre Bontron, LSR, Grenoble. He was also member of the jury of CNAM engineer diploma.

9.6. Supervision of internships

- Gilles Barthe and Benjamin Grégoire co-supervised Fernando Pastawski.
- Gilles Barthe and Jean-Louis Lanet co-supervised Santiago Zanella.
- Marieke Huisman supervised Pratik Worah.

9.7. Teaching

- Gilles Barthe taught: *Sémantique et Sécurité* (Semantics and Security), Mastère Recherche, University of Nice; and *Méthodes formelles pour cartes à puce* (Formal methods for smart cards), Ecole Jeunes Chercheurs en Programmation 2004, Le Croisic; *Sécurité des cartes à puce* (Smartcard security, short course), at Ecole des Mines de Paris.
- Guillaume Dufay taught: *Algorithmique et Informatique Théorique* (Algorithms and Theoretical Computer Science), DEUG MASS 2 *Programmation, C et outils* (Programming, C and tools), Licence Informatique.
- Marieke Huisman taught: *Sémantique des langages de programmation* (Programming language semantics), Maîtrise Informatique, University of Nice.
- Jean-Louis Lanet taught: *Sécurité des systèmes embarqués* (Embedded systems security), Mastère OTSI-SI Université de Savoie and Ecole d'Ingénieur de Luminy, *Sûreté de fonctionnement et modélisation formelle* (Software safety and formal modeling), Mastère TIIR, Université de Lille, *Technologie de la carte à puce* (SmartCard Technology, short course), Ecole des Mines de Paris.
- Sabrina Tarento taught: *Algorithmique et Programmation en C* (Algorithmics and Programming in C), DEUG MASS 1;*Calcul Formel* (Computer Algebra), DEUG MASS 1.

10. Bibliography

Doctoral dissertations and Habilitation theses

[1] G. BARTHE. *De la théorie des types à la vérification formelles des petits objets portables de sécurité*, Habilitation à Diriger des Recherches, Université de Nice Sophia-Antipolis, 2004.

[2] N. CATANO. *Formal methods for Java Programs*, Ph. D. Thesis, Université de Paris, 2004.

[3] G. DUFAY. *Vérification formelle de la plate-forme Java Card*, Ph. D. Thesis, Université de Nice Sophia-Antipolis, 2004.

[4] M. FRADE. *Type-Based Termination of Recursive Definitions and Constructor Subtyping in Typed Lambda Calculi*, Ph. D. Thesis, University of Minho, 2004.

[5] J.-L. LANET. *Produire des logiciels sûrs*, Habilitation à Diriger des Recherches, Université de Marseille, 2004.

Articles in referred journals and book chapters

[6] G. BARTHE. *Type Isomorphisms and Back-and-Forth Coercions in Type Theory*, in "Mathematical Structures in Computer Science", To appear, 2005.

[7] G. BARTHE, A. BASU, T. REZK. *Security Types Preserving Compilation*, in "Journal of Computer Languages, Systems and Structures", To appear, 2005.

[8] G. BARTHE, T. COQUAND. *On the equational theory of non-normalizing Pure Type Systems*, in "Journal of Functional Programming", 2004.

[9] C. BREUNESSE, N. CATAÑO, M. HUISMAN, B. JACOBS. *Formal Methods for Smart Cards: an experience report*, in "Science of Computer Programming", To appear, 2005.

Publications in Conferences and Workshops

[10] G. BARTHE, A. BASU, T. REZK. *Security Types Preserving Compilation*, in "Proceedings of VMCAI'04", Lecture Notes in Computer Science, vol. 2934, Springer-Verlag, 2004, p. 2–15.

[11] G. BARTHE, J. CEDERQUIST, S. TARENTO. *A Machine-Checked Formalization of the Generic Model and the Random Oracle Model*, in "Proceedings of IJCAR'04", Lecture Notes in Computer Science, vol. 3097, 2004, p. 385-399.

[12] G. BARTHE, P. D'ARGENIO, T. REZK. *Secure Information Flow by Self-Composition*, in "Proceedings of CSFW'04", IEEE Press, 2004, p. 100-114.

[13] G. BARTHE, G. DUFAY. *A Tool-Assisted Framework for Certified Bytecode Verification*, in "Proceedings of FASE'04", Lecture Notes in Computer Science, vol. 2984, Springer-Verlag, 2004, p. 99-113.

[14] G. BARTHE, B. GRÉGOIRE, F. PASTAWSKI. *Practical inference for typed-based termination in a polymorphic setting*, in "Proceedings of TLCA'05", Lecture Notes in Computer Science, To appear, Springer-Verlag, 2005.

[15] G. BARTHE, L. PRENSA-NIETO. *Formally Verifying Information Flow Type Systems for Concurrent and Thread Systems*, in "Proceedings of FMSE'04", ACM Press, 2004, p. 13-22.

[16] G. BARTHE, T. REZK. *Non-interference for a JVM-like language*, in "Proceedings of TLDI'05", ACM Press, 2005.

[17] M. HUISMAN, D. GUROV, C. SPRENGER, G. CHUGUNOV. *Checking Absence of Illicit Applet Interactions: a Case Study*, in "Fundamental Approaches to Software Engineering, FASE 2004", Lecture Notes in Computer Science, n^o 2984, Springer-Verlag, 2004, p. 84–98.

[18] F. KAMMÜLLER, J. W. SANDERS. *Heuristics for Refinement Relations*, in "2nd IEEE Int. Conf. on Software Engineering and Formal Methods SEFM04", 2004.

[19] F. KAMMÜLLER, J. W. SANDERS. *Idempotent Relations in Isabelle/HOL*, in "First International Colloquium on Theoretical Aspects of Computing. ICTAC 2004", LNCS, vol. 3407, 2004, p. 312-326.

[20] M. PAVLOVA, G. BARTHE, L. BURDY, M. HUISMAN, J.-L. LANET. *Enforcing High-Level Security Properties For Applets*, in "Proceedings of CARDIS'04", Kluwer, 2004.

[21] C. SPRENGER, D. GUROV, M. HUISMAN. *Compositional Verification for Secure Loading of Smart Card Applets*, in "Second ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'04)", IEEE, 2004, p. 211–222.

[22] L. DU BOUSQUET, Y. LEDRU, O. MAURY, C. ORIAT, J.-L. LANET. *An experiment in JML-based software validation*, in "Proceedings of ASE'04", IEEE, 2004, p. 294-297.

Internal Reports

[23] D. GUROV, M. HUISMAN. *Abstraction over Public Interfaces*, Technical report, n^o RR-5330, INRIA, 2004, http://www.inria.fr/rrrt/rr-5330.html.

[24] M. HUISMAN, K. TRENTELMAN. *Factorising temporal specifications*, To appear in Computing: The Australasian Theory Symposium (CATS 2005), Technical report, n^o RR-5326, INRIA, 2004, http://www.inria.fr/rrrt/rr-5326.html.

Miscellaneous

[25] S. Z. BÉGUELIN. *Formal specification of GlobalPlatform Card Security Requirements*, Manuscript, 2004.

[26] I. SIVERONI. *Security Automata, Program Annotations and High-level Security Properties*, Manuscript, 2004.