INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Project-Team EVEREST*

# *Vérification et sécurité du logiciel*

*Sophia Antipolis - Méditerranée*

THEME SYM

Activity Report

2007

# Table of contents

# 1. Team

**Head of project-team**
Gilles Barthe [ Research Director INRIA, HdR ]

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

**Research scientist**
Benjamin Grégoire [ Research scientist INRIA ]
Tamara Rezk [ Research scientist INRIA, since September 2007 ]

**Administrative assistant**
Nathalie Bellesso
Claire Seneca [ replacement, since May 2007 ]

**PhD students**
Julien Charles [ MESR grant, Teaching Assistant UNSA ]
Clément Hurlin
César Kunz
Jorge-Luis Sacchini [ since November 2007 ]
Santiago Zanella [ Microsoft grant ]

**Visiting PhD students**
Salvador Cavadini [ visiting PhD student from University of San Luis, Argentina, 24 months ]
Gustavo Petri [ visiting PhD student from University of San Luis, Argentina, since February 2007, 18 months ]

**Post-doctoral Fellow**
Romain Janvier [ until September 2007 ]
Yu Zhang [ until August 2007 ]

**Student interns**
Henri-Charles Blondeel [ Supelec, France and KTH, Sweden, 6 months ]
Franco Brusatti [ University of Rosario, Argentina, 6 months ]
Andres Krapf [ University of Rosario, Argentina, 6 months ]
Federico Olmedo [ University of Rosario, Argentina, 6 months ]
Jorge-Luis Sacchini [ University of Rosario, Argentina, 6 months ]

**Technical Staff**
Sophie Hadjadj [ until November 2007 ]
Anne Pacalet
Colin Riba [ since November 2007 ]

**Visiting scientist**
Miguel Pagano [ University of Cordoba, Argentina, 1.5 months ]
Gimena Pujol Vivero [ University of Malaga, Spain, 1 month ]

# 2. Overall Objectives

## 2.1. Overall Objectives

The Everest project concentrates on increasing reliability and security of mobile and embedded software. This is achieved by developing and applying formal methods and language-based techniques, covering both platform and application level. The project's privileged application domain ranges from trusted personal devices, such as mobile phones and smart cards, to ubiquitous computing.

The project focuses on the following research areas:

- Program verification and Proof Carrying Code;
- Machine-checked semantics;
- Machine-checked provable cryptography;
- Foundations of proof assistants.

# 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 run-time 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, and aliasing 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 secure 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 exists for describing computational systems.
- *Logic is precise*. While types generally over-approximate program behaviour, logic can be used to provide precise statements about program behaviour. Special conditions can be assumed and exceptional behaviours 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 analyses to be combined*. Logic provides a common setting into which the declarative content of a typing judgement 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 logic-based program verification of Java, which has culminated in the realisation of program verification environments for single-threaded Java.

## 3.3. Machine-checked semantics and algorithms

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 modelling 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 smart cards, such as Global Platform.

We are also interested in developing formal, machine-checked security proofs for cryptographic algorithms, using tools from provable cryptography. In particular, in the past we have formalised the Generic Model and Random Oracle Model, and formally verified security bounds for the probability of an attacker breaking the discrete logarithm and related encryption and signing schemes. We now focus on formal proofs for the computational model, which has the advantage that it makes less hypotheses on the attacker. Proofs in the computational model are based on game-playing techniques.

## 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 run-time 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 run-time 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.

We give precise mathematical definitions of these requirements, using programming language semantics, and provide means to enforce these requirements using type systems or logic.

## 3.5. Proof Carrying Code

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 realise 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

Smart devices, including new generation smart cards 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 organisations 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 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 software base (a virtual 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 devices (so-called post issuance), and by allowing several commercially independent applications to run on a single device. This open character forms their commercial strength, but also creates a technical challenge: reliability, correctness and security become crucial issues, since malicious applications might potentially exploit bugs in the smart device platform, with detrimental effects on security and/or privacy.

### 4.2. Global computing

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 unauthorised 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. Jack: a tool for applet validation

**Participants:** Gilles Barthe, Julien Charles, Benjamin Grégoire, Marieke Huisman.

In 2003, the team took over the development of Jack (Java Applet Correctness Kit, a tool for applet validation), initiated within the research laboratory of the smart card manufacturer Gemplus (now Gemalto). Since then, we added many features to Jack, in particular support for interactive verification of proof obligations, and for reasoning at bytecode level. In 2007, Jack has been made publicly available under the Cecill C licence. For more information about Jack, see http://www-sop.inria.fr/everest/soft/Jack/jack.html.

## 6. New Results

### 6.1. Program verification and Proof Carrying Code

**Participants:** Gilles Barthe, Henri-Charles Blondeel, Salvador Cavadini, Julien Charles, Benjamin Grégoire, Marieke Huisman, Clément Hurlin, César Kunz, Anne Pacalet, Gustavo Petri, Tamara Rezk, Jorge-Luis Sacchini.

1. In a Proof-Carrying Code scenario where the functional correctness of the downloaded code is essential, certificate generation is a challenging task. Certificate translation is an approach that extends the set of properties considered in a PCC setting by transforming certificates of source programs into certificates of compiled code. Previously, we have considered certificate transformers individually for each standard compiler optimisation, for an intermediate RTL language and a particular verification environment. The lack of a generic framework was a clear limitation of our earlier work; it was quite difficult to extend our results or even to assess the scalability of certificate translation.

We address these limitations in an algebraic setting, abstracting away from the specifics of programming languages, program transformations, and verification methods. The main ingredient of the certificate transformation is the relation between the verification environment and the static analysis that justifies a program transformation, two elements that tightly constrain the framework. Hence, we gain a substantial leverage by modelling both elements in the same abstract interpretation framework, not only because we avoid being attached to a single setting, but also because a unified formalisation of the verification and the analysis permits a more precise study of their relations.

2. We have done theoretical and practical experiments in extending certificate translation to aspect-oriented programs; one primary motivation for this work is to apply the verification infrastructure developed in the Mobius project to component verification.

   Basically, we developed a compositional (Hoare-like) logic for a simplified aspect oriented language. It is simplified in the sense that “pointcut descriptors" that specify when an advice must be triggered are reduced to statically decidable conditions. This is unavoidable if we plan to statically verify the code. In addition, given a compiler that “flattens" a simple AOP program to RTL code, we described a process to translate certificates between the corresponding verification environments.

3. We have defined an information flow type system for non-interference for a sequential JVM-like language that includes classes, objects, arrays, exceptions and method calls, and we have proven that it guarantees non-interference [15], [21]. For increased confidence, we have formalised the proof in the proof assistant Coq, showing soundness of the type system *w.r.t.* the Bicolano semantics [4] (see below); an additional benefit of the formalisation is that from our proof we have extracted a certified lightweight bytecode verifier for information flow.

   We have further extended our information flow type system in two directions:

   **Language coverage** In the past couple of years, definitions for secure information flow (more specifically, for non-interference) have emerged in order to account for concurrent aspects of programs. Recently, A. Sabelfeld and A. Russo from Chalmers have proposed an input-output definition for high-level concurrent imperative language that easily extends non-interference type systems. We have collaborated with them (in the context of the Mobius project) to extend this work for Java bytecode and to prove a type-preservation result from secure Java high-level programs to bytecode. We have published our first results in Esorics 2007 [5]. We expect to be able to formalise the soundness proof of this work, using an extension of the Bicolano semantics for concurrent Java.

   **Support for more expressive policies** In practice, non-interference is a quite restrictive confidentiality property to specify secret information leakage. It is technically challenging to express more flexible policies that characterise allowance of disclosure of secret (partial) information without disclosing more than what is specified. Policies that permit partial or total disclosure of given secret information are called declassification policies and have been widely studied in the information flow literature. In recent work we have envisioned a verification technique that extends type systems for non-interference as well as their soundness proofs in a modular way, to cope with declassification policies. This technique uses a (semantics-preserving) program transformation that transforms specific declassifiers language constructs into public variables that can later be checked for leaks with standard type systems.

   The importance of this rewriting of declassification policies into non-interference policies is twofold: on one side, verification of declassification can be handled with non-interference tools, on the other hand, soundness proofs for declassification can be parametrised by soundness proofs for non-interference and a proof of a semantics-preserving transformation. We plan to adapt our type system and proof for non-interference formalised in Coq in order to account for declassification, and thus to obtain the first certified extracted analyser for bytecode declassification policies.

4. In collaboration with H. Lehner at ETH Zürich, Switzerland, we have worked on the Mobius DirectVCGen, which generates verification conditions both for annotated Java source code and bytecode. It is developed to be used in the context of a proof-transforming compiler. The main characteristic of the tool is that the generated verification conditions for source code and bytecode are equivalent. In particular, a simple transformation allows us to re-use a source code level proof of a proof obligation for the corresponding bytecode level proof obligation. The source code verification condition generator is written in Java, and the bytecode verification condition generator is written in Coq. The verification conditions are stated in both cases in Coq's specification language, using the Bicolano formalisation as background theory. As the DirectVCGen is part of the Mobius PVE, it uses the specification parser of ESC/Java 2 as well as its Coq output.
5. We have continued to work on example proofs using JML's native construct. The main idea is to have a bytecode verifier written in Java and a formalisation of the bytecode verifier in Coq. Correctness of the bytecode verifier is proven at the Coq level, and then compliance of the Java bytecode verifier with the Coq bytecode verifier is proven using the native construct. We have begun the formalisation in Coq, but this is still work in progress.
6. We have studied the use of a Verification Condition Generator (VCgen) for Java Bytecode. The work focused in particular on reducing the large number of proof obligations that are generated when a VCgen is applied to Java Bytecode. This explosion in the number of generated proof obligations is caused by instructions that can throw run-time exceptions. Our solution is to use static analysis to prune unreachable exceptions from the control flow graph of the program. We focused on null-pointer analyses to eliminate possible null-pointer exceptions. Some of the work has been formalised in Coq. This work has been presented at TGC [8].
7. The Mobius specific task force, in which the Everest team is one of the active partners, has continued work on the BML Reference Manual. BML, short for Bytecode Modeling Language, is the bytecode cousin of JML [6]. The reference manual specifies the syntax and semantics of the language, the class file format that is used to store the annotations, and it defines the compilation from JML to BML. The reference manual serves as a basis for the tool development of the bytecode subcomponent of the Mobius tool set. A preliminary version of the reference manual is available via http://www-sop.inria.fr/everest/BML.
8. In earlier work, an algorithm to generate annotations for high-level security properties has been implemented in Jack [20]. In collaboration with A. Tamalet from the University of Nijmegen, Netherlands, we are working on a formalisation of this algorithm, in order to prove the approach correct formally. Given a property described by a finite state machine (FSM) and a Java application, we have defined how the FSM can be captured by appropriate JML set annotations. For a basic program format, we have defined a general semantics that can be instantiated in different ways, to specify a run-time annotation checking or a monitoring semantics. Further, we have defined an algorithm that transforms a program monitored by an automaton into an annotated program. We prove that the generated annotations correctly capture the monitoring, *i.e.*, if the monitored program reaches an error-state, then one of the generated assertions will be violated. The formal correctness proof is being developed in a theorem prover. As a last case, we still have to find suitable restrictions for try-catch-finally statements, under which the correctness result holds. A next step is to propagate the annotations, and to show that eventually correctness of the annotations can be proven statically.
9. In collaboration with C. Haack from the University of Nijmegen, Netherlands, we have developed an annotation system to specify thread permissions (based upon Boyland's fractional annotation system [17]). Basically, the annotation system allows to specify when a thread can rely upon a variable to be "stable", either because it has exclusive (write) access to the variable, or because all other threads only can read it, which provides a sufficient condition to allow thread-modular reasoning about this variable. We have shown how the annotation system can be used to specify many common multithreaded programming patterns [10] and how they can exploited for further verification of multithreaded programs [11]. Together with C. Haack, an automatic verification method for a subset of the annotation system is under development.

10. Program slicing helps humans to better understand programs, but it can also simplify formal program verification. C programs are especially difficult to check because of their low level operations. Together with the CEA, we are developing a tool that can slice C programs without excluding those low level aspect of the language. The slicing precision relies on the PDG (Program Dependences Graph) that is a key element of the computation. Interprocedural analysis has to be done carefully to avoid losing too much information. We are also able to enhance the precision by having different slices for each source function. The tool can give a program slice relevant to a given property. This property can then be checked on the (smaller) sliced program. At the moment, the tool is usable on realistic medium size programs.
11. In the literature, different definitions of *confidentiality* for multi-threaded applications have been proposed. We have been studying these definitions, and reformulated them for a single, common program model. This allows for an easier comparison of the different definitions. In additions, the definitions that do not depend on probabilities, can be model checked for concrete programs. To experiment with this, the program model and the different security properties have been formalised in the Concurrency Workbench.
12. In collaboration with D. Gurov and I. Aktug from KTH, Sweden, we have continued our work on adapting our compositional verification techniques [2] to richer program models. To do this, we have generalised our basic program model, for sequential programs without exceptions, to a generic program model. The basic program model is a simple instantiation of this generic model, other instantiations are a program model with exceptions, and a program model with multi-threaded control flow. For the generic model, we show under which conditions our compositional verification technique still applies. The concrete instantiations that we have developed do indeed satisfy these conditions.

    In a different line of work, we have continued the work on characterising behavioural properties by sets of structural properties. We have extended the translation from modal logic behavioural formulae to formulae with greatest fixed points [9]. The translation uses a tableau construction, encoding a symbolic execution of the behavioural formulae. During the symbolic execution, a symbolic call stack is maintained. This call stack is used to decide whether a formula has been sufficiently explored, and thus to make the tableau construction terminate. The translation has been implemented in Ocaml.

## 6.2. Machine-checked semantics

**Participants:** Gilles Barthe, Benjamin Grégoire, Marieke Huisman, Andres Krapf, Anne Pacalet, Gustavo Petri.

1. In order to use the Bicolano formalisation more accurately as background theory, we extended the tool Bico. Given a collection of Java class files, the background theory describes the corresponding Bicolano datastructures. To generate this automatically, the prototype tool Bico has been developed by P. Czarnik at the University of Warsaw, Poland. However, the existing translation from bytecode to Bicolano formalisation did not manage class inter-dependencies, it did not provide automatic compilation of all generated modules, and the code was undocumented. We improved this situation, in particular to manage class interdependency, by adding generation of appropriate Makefiles, and a restructuring of the generated files into well-defined modules.

    In addition, we also added an annotation generation feature to Bico. Inside the Mobius DirectVCGen this will produce a file with the Coq annotations, which are bound to the source files, or given as separate JML specification files, if the Class file currently inspected is only available as a bytecode file.
2. We have formalised an extension of Bicolano for multi-threading (BicolanoMT). This formalises the additional instructions and native methods, related to multi-threading. In addition, this requires to divide the single-threaded instructions into different groups, depending on whether they update the call stack, *i.e.*, call or return from a method, in which case synchronisation might happen, or not.

3. We have completed the formalisation of the Java Memory Model (JMM) and proved the *data-race-freedom* (DRF) guarantee that race-free programs are sequentially consistent [12]. Further investigations have been made into how the DRF guarantee can be extended to "benign data races", *i.e.*, data races that always write the same value. This work has been done in collaboration with D. Aspinall and J. Sevcik from the University of Edinburgh, UK.

   We are currently investigating how we can make the link between BicolanoMT and the JMM formally, *i.e.*, can we map bytecode instructions into actions of the memory model, and vice versa (for sequentially consistent executions).

4. A hypervisor, also called Virtual Machine Monitor (VMM), is a virtualisation platform that allows multiple Operating Systems to run on a host computer at the same time. One of the main problems to tackle in this kind of systems is the memory management.

   Therefore, we formalised the Memory Management behaviour that is being controlled by a Virtual Machine Monitor. In particular, we formally specify and verify security properties related to isolation (with respect to memory, *i.e.*, data privacy).

## 6.3. Machine-checked provable cryptography

**Participants:** Gilles Barthe, Franco Brusatti, Benjamin Grégoire, Romain Janvier, Federico Olmedo, Tamara Rezk, Santiago Zanella, Yu Zhang.

Provable security, whose origins can be traced back to the pioneering work of Goldwasser and Micali [18], advocates a mathematical approach based on complexity theory in which the goals and requirements of cryptosystems are specified precisely, and where the security proof is carried out rigorously and makes all underlying assumptions explicit. In a typical provable security setting, one reasons about effective adversaries, modelled as arbitrary probabilistic polynomial-time Turing machines, and about their probability of thwarting a security objective, *e.g.*, secrecy. In a similar fashion, security assumptions about cryptographic primitives bound the probability of polynomial algorithms to solve difficult problems, *e.g.*, computing discrete logarithms. The security proof is performed by reduction by showing that the existence of an effective adversary with a certain advantage in breaking security implies the existence of an effective algorithm contradicting the security assumptions.

The game-playing technique is a general method to structure and unify cryptographic proofs. Its central idea is to view the interaction between an adversary and the cryptosystem as a game, and to study game transformations that preserve security, thus allowing to transform an initial game — that explicitly encodes a security property — into a game where it is easy to bound the advantage of the adversary. Code-based techniques (*CBT*) is an instance of the game-playing technique that has been used successfully to verify state-of-the-art cryptographic schemes, see *e.g.* [16]; its distinguishing feature is to take a code-centric view of games, security hypotheses and computational assumptions, that are expressed using (probabilistic, imperative, polynomial-time) programs. Under this view, game transformations become program transformations, and can be justified rigorously by semantic means. Although *CBT* proofs are easier to verify and are more easily amenable to machine-checking, they go far beyond established theories of program equivalence and exhibit a rich and broad set of reasoning principles that draws from program verification, algebraic reasoning, and probability and complexity theory. Thus, despite the beneficial effect of their underlying framework, *CBT* proofs remain inherently difficult to verify. In an inspiring paper, Halevi [19] argues that formal verification techniques are mandatory to improve trust in game-based proofs, going as far as describing an interface to a tool for computer-assisted code-based proofs. As a first step towards the completion of Halevi's programme, we developed *CertiCrypt*, a framework to construct machine-checked code-based proofs in the Coq proof assistant [22]. *CertiCrypt* achieves many important goals of Halevi's ideal tool. At the same time, it brings a formal semantics perspective on the design of the tool. The main characteristics of *CertiCrypt* are:

- *Direct and faithful encoding of code-based techniques*. In order to take advantage of the generality of the *CBT* approach and to be readily accessible to cryptographers, we have chosen a formalism that is commonly used by cryptographers to describe games. Concretely, the lowest layer of *CertiCrypt* is a deep embedding in Coq of an imperative programming language with random assignments, structured datatypes, and procedure calls. The language semantics takes non-standard features such as complexity of programs, variable usage and calling policies into account. These are of paramount importance in cryptographic proofs.

- *Support for automated proofs*. We have developed certified tactics for the most common transformations: 1) semantics preserving transformations, including compiler optimisations such as dead code elimination, code motion, constant propagation and common subexpression elimination; 2) transformations based on indistinguishability, *i.e.*, a change that cannot be detected with a non-negligible probability; and 3) transformations based on failure events, where both games behave identically unless a certain *failure* occurs, and it is shown that this failure occurs with negligible probability. This corresponds to the so-called *fundamental lemma* of game-playing proofs.
- *Complete and independently verifiable proofs*. *CertiCrypt* benefits from being developed on top of the Coq proof assistant to go beyond Halevi's vision in two respects. First, it supports the construction of full proofs, whereas Halevi mostly focuses on their "mundane parts". Second, it permits independent verifiability of proofs by third parties, which is an important motivation behind game-based proofs. Regarding full proofs, *CertiCrypt* requires that all (complexity-theoretic, group-theoretic, probabilistic) side conditions to apply transformations are justified within Coq, and also enables ad hoc reasoning, *e.g.*, to conclude the proof in case of a sequence of games that ends in a non-trivial game. Regarding verifiability, *CertiCrypt* inherits from Coq its ability to provide certificates, or proof objects, that are automatically verifiable with a small trusted core, namely the type-checker of Coq.

## 6.4. Type theory and proof assistants

**Participants:** Gilles Barthe, Benjamin Grégoire, Colin Riba, Jorge-Luis Sacchini.

1. We participate in the development of the Coq proof assistant. In particular, we maintain and extend the conversion algorithm based on a compiler and a virtual machine (which has been recently added to Coq).
2. The library for arbitrary precision arithmetic in Coq, developed with L. Théry (Marelle), has been integrated in Coq.
3. We study of a new version of the Calculus of Inductive Constructions, where the typing rule for pattern matching is more permissive. This new typing rule allows to suppress cases which are trivially impossible due to dependent typing.

# 7. Contracts and Grants with Industry

## 7.1. Contracts

**CEA** The CEA (Commissariat à l'Energie Atomique) develops static analysis tools for C programs based on techniques such as precondition computation using Hoare logic and abstract interpretation. The collaboration (2006-2009) aims to enhance these tools by adding slicing capabilities in order to help managing bigger applications. Building smaller models is especially useful to study applications that can not be fully handled by other analyses. Anne Pacalet and Salvador Cavadini are involved in this collaboration.

**INRIA-Microsoft Research Joint Laboratory** (Secure Distributed Computations and their Proofs, 2006-2009). Other participants are the Programming Principles and Tools group at Microsoft Research Cambridge and the Moscova team (Rocquencourt). This project intends to design formal tools for programming distributed computations with effective security guarantees. Gilles Barthe, Benjamin Grégoire, Tamara Rezk, and Santiago Zanella are involved in the project, see http://www.msr-inria.inria.fr/.

## 7.2. National projects

**ParSec** (Parallelism and Security, 2007 - 2010). Other participants are Mimosa (INRIA Sophia-Antipolis), Moscova (INRIA Rocquencourt), PPS (Paris 7), Lande (INRIA Rennes), see http://moscova.inria.fr/~zappa/projects/parsec/consortium.html.

**Scalp** (Security of Cryptographic Algorithms with Probabilities, 2007 - 2011). Other participants are Verimag (Grenoble), Plume (ENS Lyon), LRI (Paris 11) and Cedric (CNAM), see http://scalp.gforge.inria.fr/.

## 7.3. European projects

**Mobius** Gilles Barthe is the scientific coordinator of the European Integrated Project Mobius (Mobility, Ubiquity and Security), launched under the FET Global Computing Proactive Initiative in the 6th Framework program (2005-2009). The project gathers 12 academic and 4 industrial partners. The goal of this project is to develop the technology for establishing trust and security for the next generation of global computers, using the Proof Carrying Code paradigm. The essential features of the Mobius security architecture will be innovative trust management, static enforcement mechanisms and support for system component downloading, see http://mobius.inria.fr.

**Thematic Networks** The team participates in the networks **Types** (type theory), see http://www.cs.chalmers.se/Cs/Research/Logic/Types/ and **Appsem 2** (Applied Semantics, concluded), see http://www.tcs.informatik.uni-muenchen.de/~mhofmann/appsem2/.

**AlFA LERNET** The team participates in the AlFA LERNET “LER-Language Engineering and Rigorous Software Development” project, a grant contract funded by the European Commission, started in March 2005, in which European and South American universities and research centers participate.

## 7.4. International projects and collaborations

**STIC AmSud, Reseco project:** Gilles Barthe is coordinator of the Reseco project within STIC AmSud, a regional program of scientific cooperation between France, Argentina, Brazil, Chile, Peru and Uruguay. The objective of the project is to investigate reliability and security in a computational model where both the platform and applications are dynamic, so that incoming software, built from off-the-shelf components, may be destined to form part of the platform or to execute as a standard application. The partners of the Reseco project are Oasis (INRIA-Sophia Antipolis), Chile University and Diego Portales University in Chile, Republic University in Montevideo, Uruguay and National University of Cordoba (FaMAF) in Argentina.

**KTH, Stockholm, Sweden:** we collaborate with D. Gurov and I. Aktug on the development of compositional verification techniques for control-flow security properties for (multi-threaded) applets.

# 8. Dissemination

## 8.1. Conference and workshop attendance, travel

- Gilles Barthe and Benjamin Grégoire attended the inauguration of the INRIA/Microsoft joint research lab in Paris, January 2007.
- Gilles Barthe and Benjamin Grégoire attended the colloquium in honour of Gilles Kahn in Paris, January 2007.
- Julien Charles and Marieke Huisman participated in the final meeting (with the reviewers) of the ACI Geccoo in Paris, January 2007.
- Gilles Barthe, Benjamin Grégoire, Marieke Huisman and Santiago Zanella presented their work at the Dagstuhl seminar on Mobility, Ubiquity and Security, Germany, February 2007.

- Gilles Barthe visited the ProSec Security Group of Goteborg University, Sweden, March 2007.
- Gilles Barthe and Romain Janvier presented their work at the Artist meeting on Security Activity, Trento, Italy, February 2007.
- Gilles Barthe attended the final review of the EU project Inspired, Brussels, Belgium, March 2007.
- Clément Hurlin visited the Security of Systems group at the University of Nijmegen, Netherlands, March 2007.
- Anne Pacalet participated in several work meetings with CEA, Paris, March, June 2007
- Gilles Barthe and Marieke Huisman presented their work [6], [4] at ETAPS 2007, Braga, Portugal, March 2007. Marieke Huisman was co-chair of the Bytecode workshop, Gilles Barthe organised a Mobius tutorial.
- Gilles Barthe participated in a meeting on the European projects Mobius and Serenity, Malaga, Spain, May 2007.
- Gilles Barthe and Andres Krapf visited the Virtual Logix company in Paris, May 2007.
- Yu Zhang visited the *Laboratoire Spécification et Vérification* at ENS Cachan, Paris, May 2007.
- Gilles Barthe, Julien Charles, Benjamin Grégoire, Marieke Huisman, Clément Hurlin, César Kunz, Anne Pacalet, Gustavo Petri, Jorge-Luis Sacchini and Santiago Zanella participated in the annual meeting of the Mobius project, Dublin, June 2007.
- Gilles Barthe visited the universities of Buenos Aires and Cordobá in Argentina, and Santiago de Chili, Chili, June 2007.
- Marieke Huisman visited and presented her work at the Department of Theoretical Computer Science of the Royal Institute of Technology (KTH) in Stockholm, Sweden, June 2007.
- Marieke Huisman and Clément Hurlin participated in a meeting of the ParSec project, Paris, June 2007.
- Yu Zhang attended RDP, Paris, France, June 2007.
- Clément Hurlin presented his work [13] at the Isabelle workshop, Bremen, Germany, June 2007.
- Santiago Zanella attended the 2nd Workshop on Formal and Computational Cryptography, and the 20th Annual IEEE Computer Security Foundation Symposium, Venice, Italy, July 2007. During the latter, he gave a short talk presenting ongoing work.
- Julien Charles and Clément Hurlin attended FTfJP and ECOOP 2007, Berlin, Germany, July 2007.
- Benjamin Grégoire lectured at the TYPES summer school, Bertinoro, Italy, August 2007. Clément Hurlin and Jorge-Luis Sacchini attended the school.
- Gilles Barthe lectured at the Marktoberdorf summer school, Marktoberdorf, Germany, August 2007.
- Gilles Barthe and Santiago Zanella visited the Research Center for InforSecurity at the AIST, Tokyo, Japan, August 2007.
- Marieke Huisman, Clément Hurlin and Gustavo Petri presented their work [11], [12] at the VAMP workshop at Concur 2007, Lisbon, September 2007. Marieke Huisman was one of the co-organisers of this workshop.
- Gilles Barthe, Benjamin Grégoire, and Santiago Zanella attended the kick off meeting of the SCALP project, Paris, September 2007.
- Gustavo Petri and Santiago Zanella attended the LASER Summer School on Software Engineering, Elba, Italy, September 2007.
- Gilles Barthe and Santiago Zanella visited the INRIA/Microsoft joint research lab in Paris, September 2007.
- Gilles Barthe was an invited speaker at the Nordic Workshop on Programming Theory, Oslo, Norway, October 2007.

- Santiago Zanella presented his work at the 2nd Informal ACM SIGPLAN Workshop on Mechanizing Metatheory, Freiburg, Germany, October 2007.
- Julien Charles visited the Software Component Technology Group at ETH Zürich, October 2007.
- Gilles Barthe presented his work at FMCO 2007, Amsterdam, Netherlands, October 2007.
- Gilles Barthe and Tamara Rezk attended a meeting of the ReSeCo project in Montevideo, Uruguay, November 2007.
- Tamara Rezk visited Famaf, University of Córdoba, Argentina, December 2007.
- Gilles Barthe participated in a meeting of the ParSec project, Paris, December 2007.

## 8.2. Leadership within scientific community

- Gilles Barthe is scientific coordinator of:
  - the FET Integrated Project Mobius;
  - the Stic-Amsud project *Reseco*.
- Benjamin Grégoire and Marieke Huisman are task leaders for tasks in the Mobius project.
- Gilles Barthe is a member of the editorial board of the Journal of Automated Reasoning.
- Gilles Barthe was a member of the program committees for TGC 2007 (co-chair), FMSE 2007, FM 2008.
- Marieke Huisman was a member of the program committees for Bytecode 2007, FTfJP 2007, VAMP 2007 (co-chair), SAVCBS 07.
- Tamara Rezk was a member of the program committee for the verification track of ACM SAC 2008.
- Gilles Barthe is member of the Steering Committee of the Symposium on Trustworthy Global Computing.
- Marieke Huisman organised, together with Christian Haack, Joe Kiniry and Erik Poll, the VAMP (Verification of Multithreaded Java-like Programs) workshop at Concur, Lisbon, September 2007.
- Gilles Barthe, with help from the other members of the team, organised TGC 2007 and the clustered FP6 FET reviews of the European projects on Global Computing, Sophia Antipolis, November 2007.
- Tamara Rezk organised, together with Ricardo Corin, the TGC satellite workshop on the Interplay of Programming Languages and Cryptography, Sophia Antipolis, November 2007.

## 8.3. Visiting scientists

We had several visiting scientists, many of whom gave a talk in our seminar. Besides the long-term visits of Miguel Pagano and Gimena Pujol Vivero, Alejandro Hevia visited for three weeks, Bruno Pontes Soares Rocha (U. Brésil, Brasil), Bart Jacobs (U. Leuven, Belgium), David Aspinall and Jaroslav Sevcik (U. Edinburgh, UK), Christian Haack (U. Nijmegen, Netherlands) and Jan Olaf Blech (U. Kaiserslautern, Germany) visited for a week.

## 8.4. Supervision of Ph.D. projects

- Anne Pacalet is the local supervisor of Salvador Cavadini.
- Gilles Barthe supervises the Ph.D. project of Santiago Zanella (started June 2006).
- Benjamin Grégoire supervises the Ph.D. project of Julien Charles (started September 2005).
- Gilles Barthe and Benjamin Grégoire supervise the Ph.D. project of César Kunz (started October 2005).
- Marieke Huisman supervises the Ph.D. project Clément Hurlin (started September 2006).
- Marieke Huisman is the local supervisor of Gustavo Petri.

## 8.5. Ph.D. committees

Gilles Barthe was a member of the licenciate jury of Daniel Hedin (Goteborg University, Sweden), and of the PhD jury's of Yann Regis-Gianas (Paris) and Colin Riba (Nancy).

## 8.6. Supervision of internships

- Marieke Huisman supervised Henri-Charles Blondeel.
- Anne Pacalet supervised Andres Krapf.
- Benjamin Grégoire supervised Jorge-Luis Sacchini.
- Gilles Barthe and Benjamin Grégoire co-supervised Franco Brusatti and Federico Olmedo.

## 8.7. Teaching

Julien Charles taught Unix environment (2nd year, exercise sessions, 18 hours) at Polytec'elec, and RMI (2nd year, course and exercise sessions, 52 hours) at IUT.

# 9. Bibliography

## Year Publications

### Doctoral dissertations and Habilitation theses

[1] M. PAVLOVA. *Specification and verification of Java bytecode*, Ph. D. Thesis, Université de Nice Sophia-Antipolis, 2007.

### Articles in refereed journals and book chapters

[2] D. GUROV, M. HUISMAN, C. SPRENGER. *Compositional Verification of Sequential Programs with Procedures*, in "Information and Computation", Conditionally accepted, 2007.

### Publications in Conferences and Workshops

[3] G. BARTHE, L. BURDY, J. CHARLES, B. GRÉGOIRE, M. HUISMAN, J.-L. LANET, M. PAVLOVA, A. REQUET. *JACK: a tool for validation of security and behaviour of Java applications*, in "FMCO: Proceedings of 5th International Symposium on Formal Methods for Components and Objects", Lecture Notes in Computer Science, To appear, Springer-Verlag, 2007.

[4] G. BARTHE, D. PICHARDIE, T. REZK. *A Certified Lightweight Non-Interference Java Bytecode Verifier*, in "European Symposium on Programming", R. D. NICCOLA (editor), Lecture Notes in Computer Science, vol. 4421, Springer-Verlag, 2007, p. 125 - 140.

[5] G. BARTHE, T. REZK, A. RUSSO, A. SABELFELD. *Security of Multithreaded Programs by Compilation*, in "European Symposium On Research In Computer Security", Lecture Notes in Computer Science, Springer-Verlag, 2007, p. 2–18, http://www.cs.chalmers.se/~andrei/esorics07.pdf.

[6] L. BURDY, M. HUISMAN, M. PAVLOVA. *Preliminary Design of BML: A Behavioral Interface Specification Language for Java bytecode*, in "Fundamental Approaches to Software Engineering (FASE 2007)", Lecture Notes in Computer Science, vol. 4422, Springer-Verlag, 2007, p. 215-229.

[7] C. FOURNET, T. REZK. *Cryptographically Sound Implementations for Typed Information-Flow Security*, in "Principles of Programming Languages 2008", To appear, 2007.

[8] B. GRÉGOIRE, J. SACCHINI. *Combining a Verification Condition Generator for a Bytecode Language with Static Analyses*, in "Trustworthy Global Computing", To appear, Lecture Notes in Computer Science, 2007.

[9] M. HUISMAN, D. GUROV. *Composing Modal Properties of Programs with Procedures*, in "Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA 2007)", 2007.

[10] M. HUISMAN, C. HURLIN. *Permission Specifications for Common Multithreaded Programming Patterns*, in "Henk Barendregt's 60th birthday collection", To appear, 2007.

[11] M. HUISMAN, C. HURLIN. *The Stability Problem for Verification of Concurrent Object-Oriented Programs*, in "Verification and Analysis of Multi-threaded Java-like Programs (VAMP)", 2007.

[12] M. HUISMAN, G. PETRI. *The Java Memory Model: a Formal Explanation*, in "Verification and Analysis of Multi-threaded Java-like Programs (VAMP)", 2007.

[13] C. HURLIN, A. CHAIEB, P. FONTAINE, S. MERZ, T. WEBER. *Practical Proof Reconstruction for First-Order Logic and Set-Theoretical Constructions*, in "Isabelle Workshop (ISABELLE-WS), Bremen, Germany", L. DIXON, M. JOHANSSON (editors), 2007, p. 2–13.

### Internal Reports

[14] G. BARTHE, T. REZK, A. RUSSO, A. SABELFELD. *Security of Multithreaded Programs by Compilation*, Technical report, Chalmers University of Technology, 2007.

## References in notes

[15] G. BARTHE, D. PICHARDIE, T. REZK. *Deriving an Information Flow Checker for the JVM*, Technical report, INRIA, 2006.

[16] M. BELLARE, P. ROGAWAY. *The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs*, in "Proceedings of the 25th International Cryptology Conference", LNCS, vol. 4004, 2006, p. 409-426.

[17] J. BOYLAND. *Checking Interference with Fractional Permissions*, in "Static Analysis Symposium", R. COUSOT (editor), Lecture Notes in Computer Science, vol. 2694, Springer–Verlag, 2003, p. 55–72.

[18] S. GOLDWASSER, S. MICALI. *Probabilistic Encryption.*, in "J. Comput. Syst. Sci.", vol. 28, n$^{o}$ 2, 1984, p. 270-299.

[19] S. HALEVI. *A plausible approach to computer-aided cryptographic proofs*, 2005, Cryptology ePrint Archive, Report 2005/181.

[20] M. PAVLOVA, G. BARTHE, L. BURDY, M. HUISMAN, J.-L. LANET. *Enforcing High-Level Security Properties For Applets*, in "Proceedings of CARDIS'04", P. PARADINAS, J.-J. QUISQUATER (editors), kluwer, 2004, ftp://ftp-sop.inria.fr/everest/publis/P+04cardis.pdf.

[21] T. REZK. *Verification of confidentiality policies for mobile code*, Ph. D. Thesis, Université de Nice Sophia-Antipolis, 2006.

[22] THE COQ DEVELOPMENT TEAM. *The Coq Proof Assistant Reference Manual V8.1*, 2006, http://coq.inria.fr.