RESEARCH CENTRE

Sophia Antipolis - Méditerranée

2021 ACTIVITY REPORT

Project-Team STAMP

Safety Techniques based on Formalized Mathematical Proofs

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Proofs and Verification

Contents

Project-Team STAMP

Creation of the Project-Team: 2019 November 01

Keywords

Computer sciences and digital sciences

A2.1.11. – Proof languages

A2.4.3. – Proofs

A4.5. – Formal methods for security

A5.10.3. – Planning

A7.2. – Logic in Computer Science

A7.2.3. – Interactive Theorem Proving

A7.2.4. – Mechanized Formalization of Mathematics

A8.3. – Geometry, Topology

A8.4. – Computer Algebra

A8.10. – Computer arithmetic

Other research topics and application domains

B6.1. – Software industry

B9.5.1. – Computer science

B9.5.2. – Mathematics

1 Team members, visitors, external collaborators

Research Scientists

- Yves Bertot [Team leader, Inria, Senior Researcher, HDR]
- Cyril Cohen [Inria, Researcher]
- Benjamin Grégoire [Inria, Researcher]
- Laurence Rideau [Inria, Researcher]
- Enrico Tassi [Inria, Researcher]
- Laurent Théry [Inria, Researcher]

Post-Doctoral Fellows

- Pierre Boutry [Inria, until Aug 2021]
- Christian Doczkal [Univ Côte d'Azur, until Sep 2021]
- Jean Christophe Lechenet [Inria]

PhD Student

- Swarn Priya [Inria]

Technical Staff

- Pierre Boutry [Inria, Engineer, from Sep 2021]
- Maxime Dénès [Inria, Engineer]

Interns and Apprentices

- Thomas Portet [Univ Côte d'Azur, from Mar 2021 until Aug 2021]

Administrative Assistant

- Nathalie Bellesso [Inria]

External Collaborators

- Basavesh Ammanaghatta Shivakumar [Max Planck Society, from Oct 2021]
- Gilles Barthe [Institut Max-Planck, HDR]
- Christian Doczkal [Max Planck Society, from Oct 2021]
- Loïc Pottier [Ministère de l'Education Nationale, until Sep 2021, HDR]

2 Overall objectives

Computers and programs running on these computers are powerful tools for many domains of human activities. In some of these domains, program errors can have enormous consequences. It will become crucial for all stakeholders that the best techniques are used when designing these programs.

We advocate using higher-order logic proof assistants as tools to obtain better quality programs and designs. These tools make it possible to build designs where all decisive arguments are explicit, ambiguity is alleviated, and logical steps can be verified precisely. In practice, we are intensive users of the Coq system and we participate actively to the development of this tool, in collaboration with other teams at Inria, and we also take an active part in advocating its usage by academic and industrial users around the world.

Many domains of modern computer science and engineering make a heavy use of mathematics. If we wish to use proof assistants to avoid errors in designs, we need to develop corpora of formally verified mathematics that are adapted to these domains. Developing libraries of formally verified mathematics is the main motivation for our research. In these libraries, we wish to capture not only the knowledge that is usually recorded in definitions and theorems, but also the practical knowledge that is recorded in mathematical practice, idioms, and work habits. Thus, we are interested in logical facts, algorithms, and notation habits. Also, the very process of developing an ambitious library is a matter of organisation, with design decisions that need to be evaluated and improved. Refactoring of libraries is also an important topic. Among all higher-order logic based proof assistants, we contend that those based on Type theory are the best suited for this work on libraries, thanks to their strong capabilities for abstraction and modular re-use.

The interface between mathematics, computer science and engineering is large. To focus our activities, we will concentrate on applications of proof assistants to two main domains: cryptography and robotics. We also develop specific tools for proofs in cryptography, mainly around a proof tool named EasyCrypt.

3 Research program

3.1 Theoretical background

The proof assistants that we consider provide both a programming language, where users can describe algorithms performing tasks in their domain of interest, and a logical language to reason about the programs, thus making it possible to ensure that the algorithms do solve the problems for which they were designed. Trustability is gained because algorithms and logical statements provide multiple views of the same topic, thus making it possible to detect errors coming from a mismatch between expected and established properties. The verification process is itself a logical process, where the computer can bring rigor in aligning expectations and guarantees.

The foundations of proof assistants rest on the very foundations of mathematics. As a consequence, all aspects of reasoning must be made completely explicit in the process of formally verifying an algorithm. All aspects of the formal verification of an algorithm are expressed in a discourse whose consistency is verified by the computer, so that unclear or intuitive arguments need to be replaced by precise logical inferences.

One of the foundational features on which we rely extensively is *Type Theory*. In this approach a very simple programming language is equiped with a powerful discipline to check the consistency of usage: types represent sets of data with similar behavior, functions represent algorithms mapping types to other types, and the consistency can be verified by a simple computer program, a *type-checker*. Although they can be verified by a simple program, types can express arbitrary complex objects or properties, so that the verification work lives in an interesting realm, where verifying proofs is decidable, but finding the proofs is undecidable.

This process for producing new algorithms and theorems is a novelty in the development of mathematical knowledge or algorithms, and new working methods must be devised for it to become a productive approach to high quality software development. Questions that arise are numerous. How do we avoid requiring human assistance to work on mundane aspects of proofs? How do we take advantage of all the progress made in automatic theorem proving? How do we organize the maintenance of ambitious corpora of formally verified knowledge in the long term?

To acquire hands-on expertise, we concentrate our activity on three aspects. The first one is foundational: we develop and maintain a library of mathematical facts that covers many aspects of algebra. In the past, we applied this library to proofs in group theory, but it is increasingly used for many different areas of mathematics and by other teams around the world, from combinatorics to elliptic cryptography, for instance. The second aspect is applicative: we develop a specific tool for proofs in cryptography, where we need to reason on the probability that opponents manage to access information we wish to protect. For this activity, we develop a specific proof system, relying on a wider set of automatic tools, with the objective of finding the tools that are well adapted to this domain and to attract users that are initially specialists in cryptography but not in formal verification. The third domain is robotics, as we believe that the current trend towards more and more autonomous robots and vehicles will raise questions of safety and trustability where formal verification can bring significant added value.

4 Application domains

4.1 Mathematical Components

The Mathematical Components library is the main by-product of an effort started almost two decades ago to provide a formally verified proof for a major theorem in group theory. Because this major theorem had a proof published in books of several hundreds of pages, with elements coming from character theory, other coming from algebra, and some coming from real analysis, it was an exercice in building a large library, with results in many domains, and in establishing clear guidelines for further increase and data search.

This library has proved to be a useful repository of mathematical facts for a wide area of applications, so that it has a growing community of users in many contries (Denmark, France, Germany, Japan, Singapore, Spain, Sweden, UK, USA) and for a wide variety of topics (transcendental number theory, elliptic curve cryptography, articulated robot kinematics, recently block chain foundations).

Interesting questions on this library range around the importance of decidability and proof irrelevance, the way to structure knowledge to automatically inherit theorems from one topic to another, the way to generate infrastructure to make this automation efficient and predictable. In particular, we want to concentrate on adding a new mathematical topic to this library: real analysis and then complex analysis (Mathematical Components Analysis).

On the front of automation, we are convinced that a higher level language is required to describe similarities between theories, to generate theorems that are immediate consequences of structures, etc, and for this reason, we invest in the development of a new language on top of the proof assistant (ELPI).

4.2 Proofs in cryptography

When we work on cryptography, we are interested in the formal verification of proofs showing that some cryptographic primitives provide good guarantees against unwanted access to information. Over the years we have developed a technique for this kind of reasoning that relies on a programing logic (close to Hoare logic) with probabilistic aspects and the capability to establish relations between several implementations of a problem. The resulting programming logic is called *probabilistic relational Hoare logic*. We also study questions of *side-channel* attacks, where we wish to guarantee that opponents cannot gain access to protected knowledge, even if they observe specific features of execution, like execution time (to which the answer lies in *constant-time* execution) or partial access to memory bits (to which the answer lies in *masking*).

For this domain of application, we choose to work with a specific proof tool (EasyCrypt), which combines powerful first-order reasoning and uses of automatic tools, with a specific support for probabilistic relational Hoare Logic. The development of this EasyCrypt proof tool is one of the objectives of our team.

When it comes to formal proofs of resistance to side-channel attack, we contend that it is necessary to verify formally that the compiler used in the production of actually running code respects the resistance properties that were established in formally verified proofs. One of our objectives is to describe such a compiler (Jasmin) and show its strength on a variety of applications.

4.3 Proofs for robotics

Robots are man-made artifacts where numerous design decisions can be argued based on logical or mathematical principles. For this reason, we wish to use this domain of application as a focus for our investigations. The questions for which we are close to providing answers involve precision issues in numeric computation, obstacle avoidance and motion planning (including questions of graph theory), articulated limb cinematics and dynamics, and balance and active control.

From the mathematical perspective, these topics require that we improve our library to cover real algebraic geometry, computational geometry, real analysis, graph theory, and refinement relations between abstract algorithms and executable programs.

In the long run, we hope to exhibit robots where pieces of software and part of the design have been subject to formal verification.

5 Highlights of the year

This year we consider our formal proof of the theorem on the unsolvability of quintic polynomials by radicals to be a highlight, as it illustrates the wealth of existing results in the Mathematical Components library [11].

6 New software and platforms

Here we describe new software and platforms.

6.1 New software

6.1.1 Coq

Name: The Coq Proof Assistant

Keywords: Proof, Certification, Formalisation

Scientific Description: Coq is an interactive proof assistant based on the Calculus of (Co-)Inductive Constructions, extended with universe polymorphism. This type theory features inductive and co-inductive families, an impredicative sort and a hierarchy of predicative universes, making it a very expressive logic. The calculus allows to formalize both general mathematics and computer programs, ranging from theories of finite structures to abstract algebra and categories to programming language metatheory and compiler verification. Coq is organised as a (relatively small) kernel including efficient conversion tests on which are built a set of higher-level layers: a powerful proof engine and unification algorithm, various tactics/decision procedures, a transactional document model and, at the very top an integrated development environment (IDE).

Functional Description: Coq provides both a dependently-typed functional programming language and a logical formalism, which, altogether, support the formalisation of mathematical theories and the specification and certification of properties of programs. Coq also provides a large and extensible set of automatic or semi-automatic proof methods. Coq's programs are extractible to OCaml, Haskell, Scheme, ...

Release Contributions: Coq version 8.14 integrates many usability improvements, as well as an important change in the core language. The main changes include:

- The internal representation of match has changed to a more space-efficient and cleaner structure, allowing the fix of a completeness issue with cumulative inductive types in the type-checker. The internal representation is now closer to the user-level view of match, where the argument context of branches and the inductive binders "in" and "as" do not carry type annotations.

- A new "coqnative" binary performs separate native compilation of libraries, starting from a .vo file. It is supported by coq_makefile.

- Improvements to typeclasses and canonical structure resolution, allowing more terms to be considered as classes or keys.

- More control over notation declarations and support for primitive types in string and number notations.

- Removal of deprecated tactics, notably omega, which has been replaced by a greatly improved lia, along with many bug fixes.

- New Ltac2 APIs for interaction with Ltac1, manipulation of inductive types and printing.

Many changes and additions to the standard library in the numbers, vectors and lists libraries. A new signed primitive integers library Sint63 is available in addition to the unsigned Uint63 library.

News of the Year: Coq version 8.14 integrates many usability improvements, as well as an important change in the core language. See the changelog at https://coq.inria.fr/refman/changes.html#version-8-14 for an overview of the new features and changes, along with the full list of contributors.

URL: http://coq.inria.fr/

Contact: Matthieu Sozeau

Participants: Yves Bertot, Frederic Besson, Tej Chajed, Cyril Cohen, Pierre Corbineau, Pierre Courtieu, Maxime Denes, Jim Fehrle, Julien Forest, Emilio Jesús Gallego Arias, Gaetan Gilbert, Georges Gonthier, Benjamin Grégoire, Jason Gross, Hugo Herbelin, Vincent Laporte, Olivier Laurent, Assia Mahboubi, Kenji Maillard, Érik Martin-Dorel, Guillaume Melquiond, Pierre-Marie Pedrot, Clément Pit-Claudel, Kazuhiko Sakaguchi, Vincent Semeria, Michael Soegtrop, Arnaud Spiwack, Matthieu Sozeau, Enrico Tassi, Laurent Théry, Anton Trunov, Li-Yao Xia, Theo Zimmermann

Partners: CNRS, Université Paris-Sud, ENS Lyon, Université Paris-Diderot

6.1.2 Math-Components

Name: Mathematical Components library

Keyword: Proof assistant

Functional Description: The Mathematical Components library is a set of Coq libraries that cover the prerequiste for the mechanization of the proof of the Odd Order Theorem.

Release Contributions: This release is compatible with Coq 8.11, 8.12, 8.13 and 8.14.

The main additions are:

the theory of diagonalization of matrices, the pairwise predicate and its theory, bounded sequences and their theory, several lemmas in various parts of the library (order.v, finset.v, etc).

URL: https://math-comp.github.io/

Contact: Assia Mahboubi

Participants: Alexey Solovyev, Andrea Asperti, Assia Mahboubi, Cyril Cohen, Enrico Tassi, François Garillot, Georges Gonthier, Ioana Pasca, Jeremy Avigad, Laurence Rideau, Laurent Théry, Russell O'Connor, Sidi Ould Biha, Stéphane Le Roux, Yves Bertot

6.1.3 Easycrypt

Keywords: Proof assistant, Cryptography

Functional Description: EasyCrypt is a toolset for reasoning about relational properties of probabilistic computations with adversarial code. Its main application is the construction and verification of game-based cryptographic proofs. EasyCrypt can also be used for reasoning about differential privacy.

News of the Year: This year, Benjamin Gregoire, Adrien Koutsos and Pierre-Yves Strub extended the EasyCrypt proof assistant to reason about complexity, by adding a Hoare logic to prove computational complexity (execution time and oracle calls) of adversarial computations. This Hoare logic is built on top of EasyCrypt module system used to model adversaries, which has been extended to support complexity restrictions.

URL: https://www.easycrypt.info/trac/

Publications: hal-03352062, hal-03469015

Contact: Gilles Barthe

Participants: Benjamin Grégoire, Gilles Barthe, Pierre-Yves Strub, Adrien Koutsos

6.1.4 ELPI

Name: Embeddable Lambda Prolog Interpreter

Keywords: Constraint Programming, Programming language, Higher-order logic

Scientific Description: The programming language has the following features

- Native support for variable binding and substitution, via an Higher Order Abstract Syntax (HOAS) embedding of the object language. The programmer needs not to care about De Bruijn indexes.

- Native support for hypothetical context. When moving under a binder one can attach to the bound variable extra information that is collected when the variable gets out of scope. For example when writing a type-checker the programmer needs not to care about managing the typing context.

- Native support for higher-order unification variables, again via HOAS. Unification variables of the meta-language (lambdaProlog) can be reused to represent the unification variables of the object language. The programmer does not need to care about the unification-variable assignment map and cannot assign to a unification variable a term containing variables out of scope, or build a circular assignment.

- Native support for syntactic constraints and their meta-level handling rules. The generative semantics of Prolog can be disabled by turning a goal into a syntactic constraint (suspended goal). A syntactic constraint is resumed as soon as relevant variables get assigned. Syntactic constraints can be manipulated by constraint handling rules (CHR).

- Native support for backtracking, to ease implementation of search.

- The constraint store is extensible. The host application can declare non-syntactic constraints and uses custom constraint solvers to check their consistency.

- Clauses are graftable. The user is free to extend an existing program by inserting/removing clauses, both at runtime (using implication) and at "compilation" time by accumulating files.

Most of these features come with lambdaProlog. Constraints and propagation rules are novel in ELPI.

Functional Description: ELPI implements a variant of lambdaProlog enriched with Constraint Handling Rules, a programming language well suited to manipulate syntax trees with binders and unification variables.

ELPI is a research project aimed at providing a programming platform for the so called elaborator component of an interactive theorem prover.

ELPI is designed to be embedded into larger applications written in OCaml as an extension language. It comes with an API to drive the interpreter and with an FFI for defining built-in predicates and data types, as well as quotations and similar goodies that come in handy to adapt the language to the host application.

Release Contributions: - Bind OCaml APIs to control the garbage collector

- Fix bugs in predicate spilling, eta conversion and a few standard library predicates

- Improve implementation of findall, which is now able to handle holes and names in the list of solutions

URL: https://github.com/lpcic/elpi/

Publications: hal-01176856, hal-01410567, hal-01897468

Contact: Enrico Tassi

Participants: Enrico Tassi, Claudio Sacerdoti Coen

6.1.5 coq-elpi

Keywords: Metaprogramming, Extension

Scientific Description: Coq-elpi provides a Coq plugin that embeds ELPI. It also provides a way to embed Coq's terms into lambdaProlog using the Higher-Order Abstract Syntax approach (HOAS) and a way to read terms back. In addition to that it exports to ELPI a set of Coq's primitives, e.g. printing a message, accessing the environment of theorems and data types, defining a new constant and so on. For convenience it also provides a quotation and anti-quotation for Coq's syntax in lambdaProlog. E.g. {{nat}} is expanded to the type name of natural numbers, or {{A -> B}} to the representation of a product by unfolding the -> notation. Finally it provides a way to define new vernacular commands and new tactics.

Functional Description: Coq plugin embedding ELPI

Release Contributions: Minor relase for extra API for global reference data types

Publications: hal-01897468, hal-01637063

Contact: Enrico Tassi

Participant: Enrico Tassi

6.1.6 Jasmin

Name: Jasmin compiler and analyser

Keywords: Cryptography, Static analysis, Compilers

Functional Description: The Jasmin programming language smoothly combines high-level and low-level constructs, so as to support "assembly in the head" programming. Programmers can control many low-level details that are performance-critical: instruction selection and scheduling, what registers to spill and when, etc. The language also features high-level abstractions (variables, functions, arrays, loops, etc.) to structure the source code and make it more amenable to formal verification. The Jasmin compiler produces predictable assembly and ensures that the use of high-level abstractions incurs no run-time penalty.

The semantics is formally defined to allow rigorous reasoning about program behaviors. The compiler is formally verified for correctness (the proof is machine-checked by the Coq proof assistant). This justifies that many properties can be proved on a source program and still apply to the corresponding assembly program: safety, termination, functional correctness...

Jasmin programs can be automatically checked for safety and termination (using a trusted static analyzer). The Jasmin workbench leverages the EasyCrypt toolset for formal verification. Jasmin programs can be extracted to corresponding EasyCrypt programs to prove functional correctness, cryptographic security, or security against side-channel attacks (constant-time).

News of the Year: Year 2021 has brought several improvements to the Jasmin programming language, enabling the implementation of more complex programs: local functions (preserved during compilation), sub-arrays, etc. The release of a new major version is scheduled for early 2022.

Preparatory work to support several target architectures have also been carried out.

The correctness theorem of the compiler has been made more precise. It now allows to reason at source level about some non-functional properties of the program produced by the compiler. In particular, there is now a formal proof (in Coq) that the compiler always preserves the "constant-time" security property.

URL: https://github.com/jasmin-lang/jasmin

Publications: hal-03430789, hal-02974993, hal-01649140

Contact: Benjamin Grégoire

Participants: Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, Vincent Laporte, Jean-Christophe Lechenet, Swarn Priya

6.1.7 Math-comp-analysis

Name: Mathematical Components Analysis

Keyword: Proof assistant

Functional Description: This library adds definitions and theorems to the Math-components library for real numbers and their mathematical structures.

Release Contributions: Compatible with MathComp 1.12.0 and 1.13.0, Coq 8.11, 8.12, 8.13, and 8.14.

News of the Year: In 2021 were added parts of the theory of convergence for sequences and series, of trigonometric functions, and of measurable spaces and measures.

URL: https://github.com/math-comp/analysis

Publication: hal-01719918

Contact: Cyril Cohen

Participants: Cyril Cohen, Georges Gonthier, Marie Kerjean, Assia Mahboubi, Damien Rouhling, Laurence Rideau, Pierre-Yves Strub, Reynald Affeldt, Laurent Théry, Yves Bertot

Partners: Ecole Polytechnique, AIST Tsukuba

6.1.8 Hierarchy Builder

Keywords: Coq, Metaprogramming

Scientific Description: It is nowadays customary to organize libraries of machine checked proofs around hierarchies of algebraic structures. One influential example is the Mathematical Components library on top of which the long and intricate proof of the Odd Order Theorem could be fully formalized. Still, building algebraic hierarchies in a proof assistant such as Coq requires a lot of manual labor and often a deep expertise in the internals of the prover. Moreover, according to our experience, making a hierarchy evolve without causing breakage in client code is equally tricky: even a simple refactoring such as splitting a structure into two simpler ones is hard to get right. Hierarchy Builder is a high level language to build hierarchies of algebraic structures and to make these hierarchies evolve without breaking user code. The key concepts are the ones of factory, builder and abbreviation that let the hierarchy developer describe an actual interface for their library. Behind that interface the developer can provide appropriate code to ensure retro compatibility. We implement the Hierarchy Builder language in the hierarchy-builder addon for the Coq system using the Elpi extension language.

Functional Description: Hierarchy Builder is a high level language for Coq to build hierarchies of algebraic structures and to make these hierarchies evolve without breaking user code. The key concepts are the ones of factory, builder and abbreviation that let the hierarchy developer describe an actual interface for their library. Behind that interface the developer can provide appropriate code to ensure retro compatibility.

Release Contributions: Support for structure parameters and coercions in mixin/factory statements. Adding compatibility with Coq 8.15

URL: https://github.com/math-comp/hierarchy-builder

Publication: hal-02478907

Contact: Enrico Tassi

Participants: Enrico Tassi, Cyril Cohen

Partner: University of Tsukuba

6.1.9 Abel - Ruffini

Name: A proof of Abel-Ruffini theorem.

Keywords: Number theory, Formalisation, Proof assistant

Functional Description: A proof of Galois Theorem (equivalence between being solvable by radicals and having a solvable Galois group) and Abel - Ruffini Theorem (unsolvability of quintic equations) in the Coq proof-assistant and using the Mathematical Components library.

Release Contributions: This is a full proof in Coq using the Mathematical Components library of Galois and Abel-Ruffini theorems about the unsolvability of the quintic. Compared to version 1.1, the proof that a solvable extension is solvable by radicals now uses Hilbert's Theorem 90 instead of matrix diagonalization. It is compatible with mathcomp version 1.12 and 1.13 and Coq from 8.10 to 8.14.

URL: https://github.com/math-comp/Abel

Contact: Cyril Cohen

Partner: Ecole Polytechnique

6.1.10 Semantics

Keywords: Semantic, Programming language, Coq

Functional Description: A didactical Coq development to introduce various semantics styles. Shows how to derive an interpreter, a compiler, a verifier, or a program analyser from formal descriptions, and how to prove their consistency.

This is a library for the Coq system, where the description of a toy programming language is presented. The value of this library is that it can be re-used in classrooms to teach programming language semantics or the Coq system. The topics covered include introductory notions to domain theory, pre and post-conditions, abstract interpretation, compilation, and the proofs of consistency between all these point of views on the same programming language. Standalone tools for the object programming language can be derived from this development.

Release Contributions: This version now contains an example of small compiler and a partial correctness proof (completeness).

URL: https://github.com/coq-community/semantics

Contact: Yves Bertot

Participants: Christine Paulin, Yves Bertot

7 New results

7.1 Post-quantum cryptography

Participants: Manuel Barbosa *(University of Porto, INESC TEC, Portugal)*, Gilles Barthe *(MPI-SP, Germany, IMDEA, Spain)*, Xiong Fan *(Algorand, Boston, USA)*, Benjamin Grégoire, Shi-Han Hung *(University of Texas, USA)*, Jonathan Katz *(University of Maryland, USA)*, Pierre-Yves Strub *(École Polytechnique)*, Xiaodi Wu *(University of Maryland, USA)*, Li Zhou *(MPI-SP, Germany)*.

EasyCrypt is a formal verification tool used extensively for formalizing concrete security proofs of cryptographic constructions. However, the EasyCrypt formal logics consider only classical attackers, which means that post-quantum security proofs cannot be formalized and machine-checked with this tool. In [7] we prove that a natural extension of the EasyCrypt core logics permits capturing a wide class of post-quantum cryptography proofs, settling a question raised by (Unruh, POPL 2019). Leveraging our positive result, we implemented EasyPQC, an extension of EasyCrypt for post-quantum security proofs, and used EasyPQC to verify postquantum security of three classical constructions: PRF-based MAC, Full Domain Hash, and GPV08 identity-based encryption.

7.2 Adding complexity notion to EasyCrypt

Participants: Manuel Barbosa *(University of Porto, INESC TEC, Portugal)*, Gilles Barthe *(MPI-SP, Germany, IMDEA, Spain)*, Benjamin Grégoire, Adrien Koutsos, Pierre-Yves Strub *(École Polytechnique)*.

We extended EasyCrypt to be able to reason about the computational complexity of adversaries. The key technical tool is a Hoare logic for reasoning about computational complexity (execution time and oracle calls) of adversarial computations. Our Hoare logic is built on top of the module system used by EasyCrypt for modeling adversaries. We proved the soundness of our logic w.r.t. the semantics of EasyCrypt programs. We showed how our approach can express precise relationships between the probability of adversarial success and their execution time. As a main benefit of our approach, we revisited security proofs of some well-known cryptographic constructions and we present a new formalization of Universal Composability (UC). The work has been published in [8].

7.3 Resistance to timing attack and Spectre

Participants: Gilles Barthe *(MPI-SP, Germany, IMDEA, Spain)*, Sunjay Cauligi *(UC San Diego, USA)*, Benjamin Grégoire, Adrien Koutsos, Kevin Liao *(MPI-SP, Germany, MIT, Boston, USA)*, Vincent Laporte, Tiago Oliveira *(University of Porto, INESC TEC, Portugal)*, Swarn Priya, Tamara Rezk, Peter Schwabe *(MPI-SP, Germany)*.

High-assurance cryptography leverages methods from program verification and cryptography engineering to deliver efficient cryptographic software with machine-checked proofs of memory safety, functional correctness, provable security, and absence of timing leaks. Traditionally, these guarantees are established under a sequential execution semantics. However, this semantics is not aligned with the behavior of modern processors that make use of speculative execution to improve performance. This mismatch, combined with the high-profile Spectre-style attacks that exploit speculative execution, naturally casts doubts on the robustness of high-assurance cryptography guarantees. In [9] we dispel these doubts by showing that the benefits of high-assurance cryptography extend to speculative execution, costing only a modest performance overhead. We build atop the Jasmin verification framework an end-to-end

approach for proving properties of cryptographic software under speculative execution, and validate our approach experimentally with efficient, functionally correct assembly implementations of ChaCha20 and Poly1305.

Many security properties of interest are captured by instrumented semantics that model the functional behavior and the leakage of programs. For several important properties, including cryptographic constant-time (CCT), leakage models are sufficiently abstract that one can define instrumented semantics for high-level and low-level programs. One important goal is then to relate leakage of source programs and leakage of their compilation—this can be used, e.g. to prove preservation of CCT. To simplify this task, we put forward the idea of structured leakage. In contrast to the usual modeling of leakage as a sequence of observations, structured leakage is tightly coupled with the operational semantics of programs. This coupling greatly simplifies the definition of leakage transformers that map the leakage of source programs to leakage of their compilation and yields more precise statements about the preservation of security properties. We illustrate our methods on the Jasmin compiler and prove preservation results for two policies of interest: CCT and cost. This work is presented in [10].

7.4 Jasmin development

Participants: Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Santiago Arranz Olmos *(MPI-SP, Germany)*.

We continued the work on Jasmin [3], allowing to add new features to the Jasmin compiler: global arrays, non-inlined function calls, pointers to arrays, better error messages. We also added various small transformations to the compiler. We started generalizing the compiler such that it will be able to generate code for different architectures (in particular Arm, but we also plan to make it work for RISC-V). All the new features are now fully proved and we plan to release a version of the compiler and its complete proof in the near future.

7.5 CryptoVerif to EasyCrypt

Participants: Bruno Blanchet, Pierre Boutry, Christian Doczkal, Benjamin Grégoire, Pierre-Yves Strub *(École Polytechnique)*.

The verification of cryptographic schemes —from the protocol level down to the correct implementation of the cryptographic primitives— is a challenging task. This has led to the development of a number of verification tools for cryptographic properties. These tools differ significantly as it comes to the properties they can express and reason about as well as the level of automation they provide. Thus, even though it is already challenging to choose the best suited tool amongst the plethora of existing ones for a given protocol, it is also impossible to prove a protocol relying on different verifiers even when different parts of the protocol could be handled by different tools.

We developed a translation from CryptoVerif to EasyCrypt that allows cryptographic assumptions that cannot be proved in CryptoVerif to be translated to EasyCrypt and proved there. We used the translation to start the proof of different hypotheses assumed in CryptoVerif:

- The reduction of the N query "real/ideal" formulation of the IND-CCA2 game in CryptoVerif to the standard single-challenge formulation (done).
- The reduction from the N participant games (e.g. insider or outsider adversaries) for authenticated KEMs to 1 or 2 participant games (in progress).
- The reduction of the N query formulation of the Computational/Gap Diffie-Hellman (CDH/GDH) games in CryptoVerif to the standard, single-query formulation. The obtained bounds are better than what can be obtained by a direct hybrid argument (almost done).

7.6 Fast equality tests with coq-elpi

Participants: Benjamin Grégoire, Jean-Christophe Léchenet, Enrico Tassi.

We studied how to use coq-elpi to implement fast equality tests for inductive datatypes in Coq. This work needs to be completed with a benchmark to test the impact on efficiency for an "inversion" tactic.

7.7 Tabulating in Elpi

Participant: Enrico Tassi.

We studied how tabulating (in other words memoization) can improve the efficiency of the Elpi implementation, using a toy interpreter for first order logic as a case study. The toy interpreter shares the same basic design as Elpi, and some of the newly studied data structures have already been transferred to Elpi. However, this work needs to be complemented with a treatement of higher order logic. The ultimate goal is to make Elpi a good tool for type class resolution in the Coq system.

7.8 Universe polymorphism in coq-elpi

Participants: Enzo Crance, Enrico Tassi.

Universe polymorphism in Coq is an important feature to ensure its usability at higher orders. Tools that exploit the meta theoretic properties of type theory need to manage this aspect of logic in a powerful way. In particular, Enzo Crance developed a tool to exploit *univalent parametricity*, using the coq-elpi language. We worked together to make sure universe polymorphism was handled correctly, thus improving the applicability of that tool.

7.9 Hierarchy Builder

Participants: Cyril Cohen, Enrico Tassi, Kazuhiko Sakaguchi *(University of Tsukuba, Japan)*.

Building algebraic hierarchies in a proof assistant such as Coq requires a lot of manual labor and often a deep expertise. To reduce the cost, we developed Hierarchy Builder (HB), a high level language to build hierarchies of algebraic structures and to make these hierarchies evolve without breaking user code. This relies on coq-elpi. We extended HB to support parameterized structures and hierarchies of functions and morphisms as well and implemented a detection of Non Forgetful Inheritance as described in the paper [1]. This was necessary to port the Mathematical Components library.

A fair amount of work was invested to make the tool usable in the long run: speed of execution, quality of error messages, diagnostic commands (HB.about), etc.

7.10 Mathematical Components on Hierarchy Builder

Participants: Reynald Affeldt *(AIST, Tokyo, Japan)*, Xavier Allamigeon, Yves Bertot, Quentin Canu, Cyril Cohen, Pierre Roux *(ONERA)*, Kazuhiko Sakaguchi *(University of Tsukuba, Japan)*, Enrico Tassi, Laurent Théry, Anton Trunov *(zillica, St. Petersburg, Russia)*.

The key to keep the Mathcomp library growing in a rational way is that it revolves around a hierarchy of interfaces which organizes operations and properties. Interfaces come with theories which apply, automatically, to all the objects which are registered as validating the interface. We replaced the hand-crafted hierarchy by one generated by Hierarchy Builder. The work is almost finished but still needs some feature implementations in Hierarchy Builder.

This work was presented at the 2021 Coq workshop [13].

7.11 Unsolvability of the Quintic Formalized in Dependent Type Theory

Participants: Sophie Bernard, Cyril Cohen, Assia Mahboubi, Pierre-Yves Strub *(École Polytechnique)*.

We provide a Coq formalization that there does not exist a general method for solving by radicals polynomial equations of degree greater than 4. This development includes a proof of Galois' Theorem of the equivalence between solvable extensions and extensions solvable by radicals. The unsolvability of the general quintic follows from applying this theorem to a well chosen polynomial with unsolvable Galois group.

This work was published and presented at ITP 2021 [11].

7.12 Lebesgue measure for Mathematical Components

Participants: Reynald Affeldt *(AIST, Tokyo, Japan)*, Cyril Cohen.

We provide a construction of the Lebesgue measure, which is a necessary and difficult step in the formalization of Lebesgue integration and its variants. The originality of our approach is the use of the mathematical structure of algebras of sets as a ground for Carathéodory's extension theorem. This is how the construction is often taught in undergraduate classes and we believe that this approach improves the modularity of the formalization because it favors abstract lemmas. It takes advantage of the Hierarchy Builder. This is part of MathComp analysis [2].

This work was presented at the 2021 Coq workshop.

7.13 Real functions in MathComp analysis

Participants: Reynald Affeldt *(AIST, Tokyo, Japan)*, Yves Bertot, Laurent Théry.

Following the HOL-light formalization, we have extended the MathComp Analysis [2] library with the definition and basic properties of the usual real functions (ln, sin, cos, tan, asin, acos, atan) and some higher-order property (sufficient conditions for the inverse function of a function to be continuous).

7.14 Coq formalization of robotics

Participants: Reynald Affeldt *(AIST, Tokyo, Japan)*, Laurent Théry.

We completed some part of the coq-robot library concerning the use of quaternions to represent rigid transformations and the definition of octonions. The code is visible on an open source repository on github.

7.15 A Coq Nix Toolbox

Participants: Cyril Cohen, Théo Zimmermann.

We created and maintained the coq-nix-toolbox, a tool chain that provides support for using the nix package manager in conjunction with a Coq development. In particular, it support easy dependency management, caching and generation of github action CI jobs.

This work was presented at 2021 Coq workshop.

7.16 Formal study of Double-word arithmetic algorithms

Participants: Laurence Rideau, Jean-Michel Muller *(CNRS, ENS de Lyon)*.

The article describing the work on the formalisation of "basic building blocks of double-word arithmetics" has been accepted and published in *Transactions on Mathematical Software* [5].

Our collaboration continues on the formalisation of algorithms for Euclidian norms. These algorithms for Euclidian norms use the square root of double-word numbers. We have formalised the correctness of two algorithms for square root, including error bound validation. Both algorithms have been studied and formalized in the general case, including overflow and underflow.

The article describing this work has been submitted to *Transactions on Mathematical Software* [16].

7.17 Document management for the Coq system

Participants: Enrico Tassi, Maxime Dénès.

We have been redesigning the communication protocol between Coq and its user-interface software to make it compliant with the LSP protocol used in Visual Studio Code. We are now exploring the use of event-based programming for this integrated development environment.

7.18 Vertical cell decomposition for motion planning algorithms

Participants: Yves Bertot, Thomas Portet *(Université Côte d'Azur)*.

We developed a formal description for a known algorithm to decompose a plane region into cells that are guaranteed to be free of obstacles. The goal of this development is to make it possible to prove that some trajectories are collision free.

7.19 Formalized theorems in graph theory

Participant: Christian Doczkal.

We submitted a paper on the proof of Wagner's theorem at "Interactive Theorem Proving" (ITP'2021), which was accepted and presented in June [15].

We formalized two proofs of the Weak Perfect Graph theorem that are both significantly simpler than a previously published proof (at CPP2020). This is part of the latest release of the GraphTheory library, in May.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

Participants: Benjamin Grégoire, Swarn Priya, Yves Bertot.

The STAMP team participates with the Grace team (Inria Saclay) in the JASMIN contract funded in the framework of the Inria-Nomadic Labs collaboration for research related to the Tezos blockchain. This contract funds the PhD thesis of Swarn Priya.

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Associate Teams in the framework of an Inria International Lab or in the framework of an Inria International Program

FLAVOR

Participants: Yves Bertot, Cyril Cohen, Laurent Théry.

Title: Formal Library of Analysis for the Verification of Robots

Duration: 2020 ->

Coordinator: Reynald Affeldt (reynald.affeldt@aist.go.jp)

Partners:

- National Institute of Advanced Industrial Science and Technology

Inria contact: Yves Bertot

Summary: The main objectives of this joint research project are as follows:

Formal verification of motion planning As a background for this topic, we will study questions concerning the formal verification of computational geometry algorithms: cell decomposition, Voronoï diagrams, Bezier curves.

Formal verification of control algorithms As a background for this topic, we will study questions concerning real analysis, integration, and ordinary differential equations. We will look at closed-loop algorithms, reasoning about incertainties (using intervals or probabilities).

Deriving embedded software from formal descriptions As a background for this topic, we will study how to derive code amenable for execution on known hardware for educational robotics (Rasberry, Arduino). This should extend work on algorithm refinement and extraction.

9.1.2 Visits of international scientists

Santiago Arranz Olmos

Status intern (master)

Institution of origin: Universdad Nacional de Córdoba

Country: Argentina

Dates: November, 15-19

Context of the visit: Work on Jasmin and adaptation to ARM

Mobility program/type of mobility: research stay

Li Zhou

Status Post-Doc

Institution of origin: Max Planck Institute

Country: Germany

Dates: December, 8-15

Context of the visit: Work on EasyCrypt and Post-quantum Cryptography

Mobility program/type of mobility: research stay

9.2 National initiatives

9.2.1 ANR

- TECAP "Analyse de protocoles, Unir les outils existants", starting on October 1st, 2017, for 60 months, with a grant of 89 kEuros. Other partners are Inria teams PESTO (Inria Nancy grand-est), Ecole Polytechnique, ENS Cachan, IRISA Rennes, and CNRS. The corresponding researcher for this contract is Benjamin Grégoire.

- SafeTLS "La sécurisation de l'Internet du futur avec TLS 1.3" started on October 1st, 2016, for 60 months, with a grant of 147kEuros. Other partners are Université de Rennes 1, and secrétariat Général de la Défense et de la Sécurité Nationale. The corresponding researcher for this contract is Benjamin Grégoire.

- Scrypt "Compilation sécurisée de primitives cryptographiques" started on February 1st, 2019, for 48 months, with a grant of 100 kEuros. Other partners are Inria team Celtique (Inria Rennes Bretagne Atlantique), Ecole polytechnique, and AMOSSYS SAS. The corresponding researcher for this contract is Benjamin Grégoire.

- NuSCAP "Numerical Safety for Computer-Aided Proofs", started on February 1st, 2021 for 48 months, with a grant covering traveling costs. Other partners are CNRS-LIP, Sorbonne University LIP6, and CNRS-LAAS. The corresponding researcher for this contract is Laurence Rideau.

9.2.2 FUI

The acronym *FUI* stands for "fonds unique interministériel" and is aimed at research and development projects in pre-industrial phase. The STAMP team is part of one such project.

- VERISICC (formal verification for masking techniques for security against side-channel attacks). This contract concerns 5 partners: CRYPTOEXPERTS a company from the Paris region (Île de France), ANSSI (Agence Nationale de Sécurité des Systèmes d'Information), Oberthur Technologies, University of Luxembourg, and STAMP. A sixth company (Ninjalabs) acts as a sub-contractant. The financial grant for STAMP is 391 kEuros, including 111kEuros that are reserved for the sub-contractant. This project started in October 2018 for a duration of 4 years. The corresponding researcher for this contract is Benjamin Grégoire.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Member of the organizing committees

Participants: Cyril Cohen, Christian Doczkal, Enrico Tassi.

Cyril Cohen and Enrico Tassi organized the international coding sprint on "porting Mathematical Components to Hierarchy Builder" in April.
Christian Doczkal was a co-organizer with Jean-Marie Madiot of the 2021 Coq-Workshop.

10.1.2 Scientific events: selection

Chair of conference program committees

Participant: Enrico Tassi.

Enrico Tassi was chair for "Logical Frameworks and Meta-Languages: Theory and Practice" (Pittsburgh, USA), in July.

Member of the conference program committees

Participants: Yves Bertot, Enrico Tassi.

Yves Bertot was member of the program committee for "Certified Programs and Proofs (CPP'2022)". Enrico Tassi was a member of the program committee for "Formal Integrated Development Environment (F-IDE'21)" and "Coq for Programming Languages (CoqPL'22)".

Reviewer

Participants: Yves Bertot, Pierre Boutry, Benjamin Grégoire, Enrico Tassi.

Pierre Boutry was a reviewer for "Certified Programs and Proofs (CPP'2022)". Benjamin Grégoire was a reviewer for "Computer Security Foundations (CSF'21)". Enrico Tassi was a reviewer for "Interactive Theorem Proving (ITP'21)". Yves Bertot was a reviewer for "Formal Structures for Computation and Deduction (FSCD'21)".

10.1.3 Journal

Reviewer - reviewing activities

Participants: Yves Bertot, Cyril Cohen, Laurent Théry.

Yves Bertot, Cyril Cohen, and Laurent Théry reviewed articles for "Journal of Automated Reasoning" (JAR). Laurent Théry reviewed an article for "Logical Methods in Computer Science" (LMCS). Yves Bertot reviewed an article for "Journal of Symbolic Logic" (JSL).

10.1.4 Invited talks

Participant: Cyril Cohen.

Cyril Cohen did a talk for AFADL 2021 about the proof of Abel-Ruffini theorem.
Cyril Cohen did a presentation and an introductory course of 4 days of the Mathematical Components Library at University of Paris (IRIF laboratory) in december 2021.

10.1.5 Leadership within the scientific community

Participant: Yves Bertot.

Yves Bertot is member of the steering committee for "Interactive Theorem Proving (ITP)", for "Coq Workshop", and for "Coq for Programming Languages".
Yves Bertot is chair of the general assembly for the Coq consortium.

10.1.6 Scientific expertise

Participant: Yves Bertot.

Yves Bertot was an expert for the evaluation of projects for Foundation UNIT and for "Agence Nationale de la Recherche (ANR)".

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Master : Yves Bertot, “Proofs and reliable programming using Coq”, 21hours ETD, Nov-Dec 2021, Master Informatique et Interactions, Université Côte d’Azur, France.
- Continuing education : Yves Bertot and Pierre Boutry, "Coq : la preuve par le logiciel", Inria Academy, 28 hours, July, November, December 2020

10.2.2 Supervision

Participants: Yves Bertot, Cyril Cohen, Benjamin Grégoire, Assia Mahboubi.

Benjamin Grégoire and Yves Bertot are supervising the thesis of Swarn Priya.
Cyril Cohen and Assia Mahboubi are supervising the thesis of Chris Hughes (based in Nantes).

10.2.3 Juries

Participant: Yves Bertot.

Yves Bertot was a member of the jury for Diane Gallois-Wong (University of Paris-Saclay) and Yannick Forster (University of Saarbrücken, Germany).

10.3 Popularization

10.3.1 Internal or external Inria responsibilities

Participant: Laurence Rideau.

Laurence Rideau is a member of the editorial board for Interstice.

Participant: Yves Bertot.

Yves Bertot participated to Salon "Open Source Experience (OSXP)" where he presented the capabilities of the Coq system, in November.

11 Scientific production

11.1 Major publications

[1] R. Affeldt, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling and K. Sakaguchi. 'Competing inheritance paths in dependent type theory: a case study in functional analysis'. In: IJCAR 2020 - International Joint Conference on Automated Reasoning. Paris, France, 29th June 2020, pp. 1–19. URL: https://hal.inria.fr/hal-02463336.

[2] R. Affeldt, C. Cohen and D. Rouhling. 'Formalization Techniques for Asymptotic Reasoning in Classical Analysis'. In: *Journal of Formalized Reasoning* (Oct. 2018). URL: https://hal.inria.fr/hal-01719918.

[3] J. B. Almeida, M. Barbosa, G. Barthe, A. Blot, B. Grégoire, V. Laporte, T. Oliveira, H. Pacheco, B. Schmidt and P.-Y. Strub. 'Jasmin: High-Assurance and High-Speed Cryptography'. In: *CCS 2017 - Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas, United States, Oct. 2017, pp. 1–17. URL: https://hal.archives-ouvertes.fr/hal-01649140.

11.2 Publications of the year

International journals

[4] G. Barthe, M. Gourjon, B. Grégoire, M. Orlt, C. Paglialonga and L. Porth. 'Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (23rd Feb. 2021), pp. 189–228. DOI: 10.46586/tches.v2021.i2.189-228. URL: https://hal.inria.fr/hal-03528937.

[5] J.-M. Muller and L. Rideau. 'Formalization of double-word arithmetic, and comments on "Tight and rigorous error bounds for basic building blocks of double-word arithmetic"'. In: *ACM Transactions on Mathematical Software* (2021). URL: https://hal.archives-ouvertes.fr/hal-02972245.

[6] F. Steinberg, L. Théry and H. Thies. 'Computable analysis and notions of continuity in Coq'. In: *Logical Methods in Computer Science* (12th May 2021). URL: https://hal.inria.fr/hal-03324295.

International peer-reviewed conferences

[7] M. Barbosa, G. Barthe, X. Fan, B. Grégoire, S.-H. Hung, J. Katz, P.-Y. Strub, X. Wu and L. Zhou. 'EasyPQC: Verifying Post-Quantum Cryptography'. In: ACM CCS 2021 - ACM SIGSAC Conference on Computer and Communications Security. CCS '21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event, South Korea: ACM, 13th Nov. 2021, pp. 2564–2586. DOI: 10.1145/3460120.3484567. URL: https://hal.inria.fr/hal-03529301.

[8] M. Barbosa, G. Barthe, B. Grégoire, A. Koutsos and P.-Y. Strub. 'Mechanized Proofs of Adversarial Complexity and Application to Universal Composability'. In: CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event, South Korea: ACM, 15th Nov. 2021, pp. 2541–2563. DOI: 10.1145/3460120.3484548. URL: https://hal.archives-ouvertes.fr/hal-03469015.

[9] G. Barthe, S. Cauligi, B. Grégoire, A. Koutsos, K. Liao, T. Oliveira, S. Priya, T. Rezk and P. Schwabe. 'High-Assurance Cryptography in the Spectre Era'. In: IEEE Symposium of Security and Privacy (S&P'21). Virtual, France, 24th May 2021. URL: https://hal.inria.fr/hal-03352062.

[10] G. Barthe, B. Grégoire, V. Laporte and S. Priya. 'Structured Leakage and Applications to Cryptographic Constant-Time and Cost'. In: CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security. CCS '21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event, South Korea: ACM, 15th Nov. 2021, pp. 462–476. DOI: 10.1145/3460120.3484761. URL: https://hal.archives-ouvertes.fr/hal-03430789.

[11] S. Bernard, C. Cohen, A. Mahboubi and P.-Y. Strub. 'Unsolvability of the Quintic Formalized in Dependent Type Theory'. In: ITP 2021 - 12th International Conference on Interactive Theorem Proving. Rome / Virtual, France, 29th June 2021. URL: https://hal.inria.fr/hal-03136002.

[12] L. Théry. 'Proof Pearl : Playing with the Tower of Hanoi Formally'. In: ITP 2021 - 12th International Conference on Interactive Theorem Proving. Rome / Virtual, Italy, 29th June 2021. URL: https://hal.inria.fr/hal-03324274.

Conferences without proceedings

[13] R. Affeldt, X. Allamigeon, Y. Bertot, Q. Canu, C. Cohen, P. Roux, K. Sakaguchi, E. Tassi, L. Théry and A. Trunov. 'Porting the Mathematical Components library to Hierarchy Builder'. In: the COQ Workshop 2021. virtuel- Rome, Italy, 2nd July 2021. URL: https://hal.archives-ouvertes.fr/hal-03463762.

Reports & preprints

[14] C. Cohen. *Formalization of a sign determination algorithm in real algebraic geometry*. 29th June 2021. URL: https://hal.inria.fr/hal-03274013.

[15] C. Doczkal. *A Variant of Wagner's Theorem Based on Combinatorial Hypermaps*. 15th Feb. 2021. URL: https://hal.archives-ouvertes.fr/hal-03142192.

[16] V. Lefèvre, N. Louvet, J.-M. Muller, J. Picot and L. Rideau. *Accurate calculation of Euclidean Norms using Double-word arithmetic*. 16th Dec. 2021. URL: https://hal.archives-ouvertes.fr/hal-03482567.

Other scientific publications

[17] C. Cohen and T. Zimmermann. *A Nix toolbox for reproducible Coq environments, Continuous Integration and artifact reuse*. Virtual, France, 2nd July 2021. URL: https://hal.inria.fr/hal-03366644.