

# 2025 Activity Report

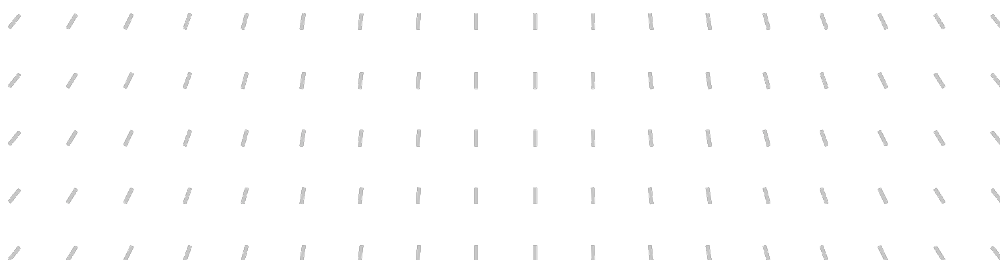
RESEARCH CENTRE: Inria Centre at Université Côte d'Azur

  
Project-Team

## SPLITS

Secure Programming Languages & Tools for Security





## **Project-Team SPLITS**

*Creation of the Project-Team: 2023 July 01*

Each year, Inria research teams publish an Activity Report presenting their work and results over the reporting period. These reports follow a common structure, with some optional sections depending on the specific team. They typically begin by outlining the overall objectives and research programme, including the main research themes, goals, and methodological approaches. They also describe the application domains targeted by the team, highlighting the scientific or societal contexts in which their work is situated. The reports then present the highlights of the year, covering major scientific achievements, software developments, or teaching contributions. When relevant, they include sections on software, platforms, and open data, detailing the tools developed and how they are shared. A substantial part is dedicated to new results, where scientific contributions are described in detail, often with subsections specifying participants and associated keywords. Finally, the Activity Report addresses funding, contracts, partnerships, and collaborations at various levels, from industrial agreements to international cooperations. It also covers dissemination and teaching activities, such as participation in scientific events, outreach, and supervision. The document concludes with a presentation of scientific production, including major publications and those produced during the year.

## Keywords

### Computer sciences and digital sciences

- A1.1.8. – Security of architectures
- A1.3.1. – Web
- A2.1.1. – Semantics of programming languages
- A2.1.4. – Functional programming
- A2.1.6. – Concurrent programming
- A2.1.7. – Distributed programming
- A2.1.10. – Domain-specific languages
- A2.1.11. – Proof languages
- A2.2.1. – Static analysis
- A2.2.8. – Code generation
- A2.2.9. – Security by compilation
- A4.5.1. – Static analysis
- A4.5.3. – Program proof

### Other research topics and application domains

- B6. – IT and telecom

## Contents

<b>Project-Team SPLITS</b>	<b>1</b>
<b>1 Team members, visitors, external collaborators</b>	<b>5</b>
<b>2 Overall objectives</b>	<b>5</b>
<b>3 Research program</b>	<b>6</b>
<b>4 Application domains</b>	<b>6</b>
4.1 Sessions . . . . .	6
4.2 Optimizing Compilers and Web Security . . . . .	6
4.3 Compilers for High Assurance Cryptography . . . . .	7
4.4 Software and Hardware for the new Spectre Era . . . . .	7
<b>5 Highlights of the year</b>	<b>7</b>
<b>6 Latest software developments, platforms, open data</b>	<b>8</b>
6.1 Latest software developments . . . . .	8
6.1.1 Easycrypt . . . . .	8
6.1.2 Jasmin . . . . .	8
6.1.3 Bigloo . . . . .	9
6.1.4 Hiphop.js . . . . .	9
6.1.5 Hop . . . . .	10
<b>7 New results</b>	<b>10</b>
7.1 Implementation of Dynamic Languages . . . . .	10
7.1.1 Compiling to WebAssembly . . . . .	10
7.1.2 Float Self Tagging . . . . .	10
7.1.3 High level languages down to micro-architecture . . . . .	11
7.2 Reactive Programming . . . . .	11
7.3 Systems for the New Spectre Era . . . . .	12
7.3.1 Software . . . . .	12
7.3.2 Preservation of speculative constant time . . . . .	12
7.3.3 Protection against Spectre RSB . . . . .	12
7.3.4 Using Intel’s Extended HW/SW Contract for Secure Compilation of Crypto Code . . . . .	13
7.3.5 Hardware . . . . .	13
7.4 Formal method for cryptography . . . . .	14
7.4.1 A Quantitative Probabilistic Relational Hoare Logic . . . . .	14
7.4.2 Proofs automation . . . . .	14
7.5 ITree semantics of Jasmin . . . . .	14
7.6 Session types . . . . .	15
7.7 Constructive characterizations of the must-preorder for asynchrony . . . . .	15
<b>8 Partnerships and cooperations</b>	<b>16</b>
8.1 International initiatives . . . . .	16
8.1.1 Inria associate team not involved in an IIL or an international program . . . . .	16
8.1.2 Participation in other International Programs . . . . .	16
8.2 International research visitors . . . . .	17
8.2.1 Visits of international scientists . . . . .	17
8.2.2 Visits to international teams . . . . .	17
8.3 National initiatives . . . . .	17
8.3.1 Action Exploratoire: AoT.js – Optimizing Compilation from Higher-Order Programming to Computer Architecture . . . . .	17
8.3.2 PEPR . . . . .	17

8.3.3	HOPR	18
8.3.4	Twinsec	18
8.4	Regional initiatives	18
8.4.1	Actions de Développement Technologique	18
<b>9</b>	<b>Dissemination</b>	<b>18</b>
9.1	Promoting scientific activities	18
9.1.1	Scientific events: organization	18
9.1.2	Scientific events: selection	19
9.1.3	Journal	19
9.1.4	Invited talks	19
9.1.5	Scientific expertise	19
9.1.6	Research administration	19
9.2	Teaching - Supervision - Juries - Educational and pedagogical outreach	20
9.2.1	Teaching	20
9.2.2	Supervision	20
9.2.3	Juries	20
<b>10</b>	<b>Scientific production</b>	<b>20</b>
10.1	Publications of the year	20
10.2	Cited publications	22

# 1 Team members, visitors, external collaborators

## Research Scientists

- Tamara Rezk [Team leader, INRIA, Senior Researcher, HDR]
- Ilaria Castellani [INRIA, Researcher]
- Benjamin Grégoire [INRIA, Researcher, HDR]
- Manuel Serrano [INRIA, Senior Researcher, HDR]

## PhD Students

- Davide Davoli [UNIV COTE D'AZUR, until Sep 2025]
- Guilhem Mizrahi [PQSHIELD, CIFRE, from Oct 2025]
- Lucas Tabary-Maujean [ENS PARIS-SACLAY, from Sep 2025]

## Technical Staff

- Jean-Christophe Léchenet [INRIA, Engineer]
- Paolo Torrini [INRIA, Engineer]

## Interns and Apprentices

- Aghilas Boussaa [ENS PARIS, Intern, from Jun 2025 until Jul 2025]
- Lucile Magnier [ENS PARIS-SACLAY, Intern, from Jun 2025 until Jul 2025]
- Francisca Quintas Monteiro De Barros [INRIA, Intern, from Mar 2025 until Jul 2025]
- Lucas Tabary-Maujean [ENS PARIS-SACLAY, Intern, from Mar 2025 until Aug 2025]

## Administrative Assistant

- Christine Foggia [INRIA]

## Visiting Scientist

- Muhammad Awais [TELECOM PARIS, from Mar 2025 until May 2025]

## External Collaborators

- Gilles Barthe [INSTITUT MAX-PLANCK, from Sep 2025, HDR]
- Marc Feeley [UNIV MONTREAL]

# 2 Overall objectives

SPLITS, Secure Programming Languages & Tools for Security, is a research team at Inria (Inria équipe-projet), focusing on defensive system security and compilers. It was created on July 1st, 2023.

SPLITS' overall goal is to develop safeguard mechanisms for confidentiality and integrity of computer systems by providing mathematical proofs that align with currently valid threat models at various levels of abstraction. To accomplish this goal, our research plan is organized around six axes, addressing different levels of abstraction:

1. Web application security
2. Program analyses for verification and vulnerability detection
3. Optimizing compilers
4. Compilers for high assurance cryptography
5. Transient execution attacks and defenses
6. Session Types and provable security

### 3 Research program

One of the most important concepts in cybersecurity is, arguably, that of a threat model. A threat model determines what is the power of an attacker: essentially, it defines what the attacker can and cannot do. For example, a particular threat model may assume that factorization cannot be solved in polynomial time, or that a number given by a random generator can never be guessed, or that memory boundaries imposed by software can never be crossed by the attacker, or that certain hardware microarchitectural features such as the cache memory *can* be observed. Threat models are important, in particular, to reason about what one is defending against by focusing on certain attacker's capabilities and abstracting away from any attacks outside the considered threat model. Threat models represent the most vulnerable point in security research and engineering. This is because threat models may be contradicted by novel attacks, and assumptions of what an attacker cannot do might be violated in practice. A remarkable example of a wrong threat model came to light in January 2018, when Spectre vulnerabilities were revealed. Up to then, application security researchers and engineers had assumed that the abstraction provided by hardware, that is the hardware architecture, was mostly unbreakable by the attacker (with few exceptions such as, for example, leaks due to the cache memory). Our research plan consists in elucidating the synergies among different threat models in the new Spectre era, by facing major scientific challenges such as the design of secure and efficient software and hardware that take the new synergies into account.

## 4 Application domains

### 4.1 Sessions

A session refers to an interaction, devoted to a particular topic, among a number of participants. When a web server communicates with an object, session information may include tokens that authenticate the server, which could be used by an attacker to access an actuator and use the device to change the physical world, e.g. the token could be used to operate a smart vacuum cleaner and obtain a house map. An attacker modifying the normal flow of a web session, e.g. manipulating session cookies or tokens, violates session integrity.

Session types were introduced in the mid-nineties, as a tool for specifying and analyzing web services in a variant of the  $\pi$ -calculus. Since then, session types have been extended in various ways and integrated into several programming languages. In 2007, session types were further enriched, as the result of a dialogue between academia and the World Wide Web Consortium (W3C), with the goal of formalizing complex web sessions between a client and a server in web applications. More recently, session types have also been equipped with security information in order to achieve access control and secure information flow between parties, but never to help to deal with one of the most important problems regarding sessions: violations to session integrity. In the short term, we plan to investigate defenses against these violations using session types.

### 4.2 Optimizing Compilers and Web Security

Running a program in a popular programming language today, such as JavaScript or Python, brings us back to the performance of a running C program on a computer from about 20 years ago. Energy consumption is subject to a similar law.

Improving the performance of modern languages, either by developing new, better adapted, hardware architectures, or by developing new techniques for implementation and execution is therefore a subject of major research.

Hop.js, an ahead-of-time JavaScript compiler, calls upon a large part of the optimization techniques developed for functional languages (Scheme and ML languages) since the beginning of the 90s. However, these methods alone are not sufficient to obtain performance comparable to those of the best JIT compilers of the moment. To approach it, new optimizations based on so-called opportunistic optimizations have been invented. They are essentially based on the idea of transposing the techniques of long-used hardware speculations. These new software techniques are still in their infancy and constitute the central element of the program research that will be developed in SPLITS. In order to do that for the JavaScript language we will investigate the approach of opportunistic JavaScript typing as well as TypeScript, a typed version of JavaScript.

As noted earlier, an antagonism has recently been highlighted between processor performance and execution security, mainly due to speculation phenomena (e.g. Spectre mentioned above) whose importance is now well understood and accepted. To be able to bring a satisfactory answer to this difficult problem, that is to say an answer that would combine performance and safety, dual skills are required: on the one hand knowledge and in-depth understanding of the concepts and techniques used in the design of architectures and on the other hand a mastery of formal mathematical tools which allow us to reason about the behavior of processors and to establish proofs of correctness and security.

### 4.3 Compilers for High Assurance Cryptography

The Jasmin compiler was designed to achieve predictability and efficiency of the output binary code. The compiler is formally verified in the Rock proof assistant. The Jasmin compiler generates binary code with provable security guarantees to defend against a speculative attacker that can measure access to the cache memory: indeed, generated Jasmin code is not vulnerable -by construction- to Spectre-PHT nor Spectre-STL. Without using a threat model for speculative execution, Jasmin generates binary code with a performance close to the fastest cryptographic code (e.g. the OpenSSL implementation in Jasmin enjoys a performance competitive with OpenSSL, even slightly beating it). We plan to extend the Jasmin compiler to generalize the guarantees it provides for the speculative and quantum thread models.

### 4.4 Software and Hardware for the new Spectre Era

At the abstraction level provided by the hardware architecture, programs are assumed to execute sequentially in the order in which the program control flow is provided. However, at the level of the hardware implementation, program execution is more complex and involves for example execution of programs out-of-order and speculatively. This complexity at the microarchitectural level was supposed to be transparent for the developer, who should only reason about programs using the abstractions provided by the hardware architecture. Yet, Spectre attacks, quickly followed by many other attacks, demonstrated how an attacker could make use of speculative execution to exfiltrate secrets that were otherwise highly protected at the architectural level. The consequences of these speculative attacks can be devastating. For example, Branch Target Injection (a.k.a. BTI or Spectre v2) allows the attacker to ignore the architectural privilege boundaries, i.e. the attacker can control the execution of a more privileged program, for instance a program belonging to the operating system kernel. Since their disclosure, speculative attacks have strongly impacted both academia and industry<sup>1</sup> and have opened a new era for security for which *almost all* previous threat models need to be revisited. Our broad goal is to provide software and hardware for the speculative threat model.

## 5 Highlights of the year

Benjamin Grégoire received a Distinguished Paper Award at the ACM Conference on Computer and Communications Security (CCS 2025) [5].

---

<sup>1</sup>Example CVEs for all major hardware industries: [CVE-2020-0551](#), [CVE-2021-26401](#), [CVE-2022-23960](#)

## 6 Latest software developments, platforms, open data

### 6.1 Latest software developments

#### 6.1.1 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.

**URL:** <https://github.com/EasyCrypt/easycrypt>

**Publications:** [hal-03352062](#), [hal-03469015](#)

**Contact:** Gilles Barthe

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

#### 6.1.2 Jasmin

**Name:** Jasmin compiler and analyser

**Keywords:** Cryptography, Static analysis, Compilers

**Scientific Description:** Jasmin is a workbench for high-assurance and high-speed cryptography. Jasmin implementations aim at being efficient, safe, correct, and secure.

Jasmin is both a language and a compiler from this language to assembly. The compiler is written and formally verified for correctness in the Rocq Prover. 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 comes with a set of tools to reason on Jasmin programs (a safety checker, a type-checker for Constant Time, a type-checker for Speculative Constant Time and an extraction to EasyCrypt to prove properties about the extracted Jasmin program, e.g. functional correctness).

**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 Rocq Prover). This ensures 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).

**Release Contributions:** Two major versions and four minor ones were published during the year 2025.

New features have been implemented in the programming language and its compiler, notably support for the RISC-V architecture, new data-types to simplify safety proofs, and more flexibility for “sub-arrays”, allowing to write more efficient programs.

The documentation of the software has been overhauled, vastly enriched and reorganized to simplify its maintenance and ensure it stays up-to-date.

Moreover there is a sustained work to fix issues when they are identified and bring improvements to the various tools: the compiler, safety analyzer, constant-time security analyzer, and extraction to EasyCrypt. These various components are also better tested.

**News of the Year:** Two major versions and four minor ones were published during the year 2025.

New features have been implemented in the programming language and its compiler, notably support for the RISC-V architecture, new data-types to simplify safety proofs, and more flexibility for “sub-arrays”, allowing to write more efficient programs.

**URL:** <https://github.com/jasmin-lang/jasmin>

**Publications:** [hal-05466117](#), [hal-05249675](#), [hal-04632106](#), [hal-04595591](#), [hal-04691165](#), [hal-04106448](#), [hal-04218417](#), [hal-03844366](#), [hal-03430789](#), [hal-03352062](#), [hal-02974993](#), [hal-02404581](#), [hal-01649140](#)

**Contact:** Jean-Christophe Léchenet

**Participants:** Alexandre Bourbeillon, Gaëtan Cassiers, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, Vincent Laporte, Jean-Christophe Léchenet, Swarn Priya, Santiago Arranz Olmos

**Partners:** The IMDEA Software Institute, Ecole Polytechnique, Universidade do Minho, Universidade do Porto, Max Planck Institute for Security and Privacy

### 6.1.3 Bigloo

**Keyword:** Compilers

**Functional Description:** Bigloo is a Scheme implementation devoted to one goal: enabling Scheme based programming style where C(++) is usually required. Bigloo attempts to make Scheme practical by offering features usually presented by traditional programming languages but not offered by Scheme and functional programming. Bigloo compiles Scheme modules. It delivers small and fast stand alone binary executables. Bigloo enables full connections between Scheme and C programs and between Scheme and Java programs.

**Release Contributions:** modification of the object system (language design and implementation), new APIs (alsa, flac, mpg123, avahi, csv parsing), new library functions (UDP support), new regular expressions support, new garbage collector (Boehm’s collection 7.3alpha1).

**URL:** <http://www-sop.inria.fr/teams/index/fp/Bigloo/>

**Contact:** Manuel Serrano

**Participant:** Manuel Serrano

### 6.1.4 Hiphop.js

**Name:** Hiphop.js

**Keywords:** Web 2.0, Synchronous Language, Programming language

**Functional Description:** HipHop.js is an Hop.js DLS for orchestrating web applications. HipHop.js helps programming and maintaining Web applications where the orchestration of asynchronous tasks is complex.

**URL:** <http://hop.inria.fr/hiphop>

**Contact:** Manuel Serrano

**Participant:** Manuel Serrano

### 6.1.5 Hop

**Keyword:** Programming language

**Scientific Description:** The Hop programming environment consists in a web broker that intuitively combines in a single architecture a web server and a web proxy. The broker embeds a Hop interpreter for executing server-side code and a Hop client-side compiler for generating the code that will get executed by the client.

An important effort is devoted to providing Hop with a realistic and efficient implementation. The Hop implementation is validated against web applications that are used on a daily-basis. In particular, we have developed Hop applications for authoring and projecting slides, editing calendars, reading RSS streams, or managing blogs.

**Functional Description:** Multitier web programming language and runtime environment.

**URL:** <http://hop.inria.fr>

**Contact:** Manuel Serrano

**Participant:** Manuel Serrano

## 7 New results

### 7.1 Implementation of Dynamic Languages

**Participants:** Manuel Serrano.

We have pursued the development of compilers of dynamic languages with a focus on the implementation of the Scheme programming language that we use as an intermediate language for compiling dynamic languages such as JavaScript or Python. This year, we have mostly focus on compiling Scheme to a new target and on the fast implementation of floating point numbers. These studies have been conducted with Robby Findler from Northwestern University in Chicago, Marc Feeley and Olivier Melançon from the University of Montréal and with Erven Rohou and Aurore Poirier from Inria Pacap in Rennes.

#### 7.1.1 Compiling to WebAssembly

WebAssembly (Wasm) has been extended to support features such as garbage collection, references, exceptions, and tail calls that facilitate compilation of managed languages. We capture a snapshot of the performance of languages that use these new capabilities from two perspectives. First, we present a language-by-language performance comparison of six managed language implementations on Wasm to the performance of their native implementations. Second, we focus on the implementation of the Bigloo Scheme compiler and explore the impact of different choices for compiling specific aspects of the language. Our findings suggest that Wasm has become a promising compilation target for most managed languages, but that its performance still falls short of that achieved by native code. Our results also show that the quality of the Wasm implementations vary, with the best ones being, on average, about 1.4× slower than the native backend and the worst ones seeing average slowdowns of more than 8×, with some tests even failing to execute correctly.

This work has been presented during the MPLR'25 conference. See [10] for more information.

#### 7.1.2 Float Self Tagging

Dynamic and polymorphic languages attach information, such as types, to run time objects, and therefore adapt the memory layout of values to include space for this information. This makes it difficult to efficiently implement IEEE754 floating-point numbers as this format does not leave an easily accessible space to store type information. The three main floating-point number encodings in use today, tagged pointers,

NaN-boxing, and NuN-boxing, have drawbacks. Tagged pointers entail a heap allocation of all float objects, and NaN/NuN-boxing puts additional runtime costs on type checks and the handling of other objects.

This paper introduces self-tagging, a new approach to object tagging that uses an invertible bitwise transformation to map floating-point numbers to tagged values that contain the correct type information at the correct position in their bit pattern, superimposing both their value and type information in a single machine word. Such a transformation can only map a subset of all floats to correctly typed tagged values, hence self-tagging takes advantage of the non-uniform distribution of floating point numbers used in practice to avoid heap allocation of the most frequently encountered floats.

Variants of self-tagging were implemented in two distinct Scheme compilers and evaluated on four microarchitectures to assess their performance and compare them to tagged pointers, NaN-boxing, and NuN-boxing. Experiments demonstrate that, in practice, the approach eliminates heap allocation of nearly all floating-point numbers and provides good execution speed of float-intensive benchmarks in Scheme with a negligible performance impact on other benchmarks, making it an attractive alternative to tagged pointers, alongside NaN-boxing and NuN-boxing.

This work has been presented during the OOPSLA'25 conference. See [9] for more information.

### 7.1.3 High level languages down to micro-architecture

Just-in-Time (JIT) compilers are able to specialize the code they generate according to a continuous profiling of the running programs. This gives them an advantage when compared to Ahead-of-Time (AoT) compilers that must choose the code to generate once for all. Is it possible to improve the performance of AoT compilers by adding Dynamic Binary Modification (DBM) to the executions?

We added to the Hopc AoT JavaScript compiler a new optimization based on DBM to the inline cache (IC), a classical optimization that dynamic languages use to implement object property accesses efficiently. Reducing the number of memory accesses, as done by the new optimization, does not reduce execution time on contemporary architectures.

The DBM optimization we have implemented is fully operational on x86\_64 architectures. We have conducted several experiments to evaluate its impact on performance and to study the reasons of the lack of acceleration.

The (negative) result we presented sheds new light on the best strategy to be used to implement dynamic languages. It tells that the old days, where removing instructions or removing memory reads always yielded to speed up, are over. Nowadays, implementing sophisticated compiler optimizations is only worth the effort if the processor is not able by itself to accelerate the code. This result applies to AoT compilers as well as JIT compilers.

This work has been presented during the PROGRAMMING'25 conference [4].

## 7.2 Reactive Programming

**Participants:** Manuel Serrano.

In 2025, after more than 18 months of extensive work, we completed the re-writing of the HipHop compilers. This effort required a full re-engineering of the entire source code. The latest version, now distributed, supports three compilation schemas:

- Circuit generation with loop duplication: This is the simplest method to implement. It has the benefit of being very close to the rules of the semantics, but it suffers from exponential growth for certain pathological (though fortunately somewhat artificial) programs.
- Circuit generation with reincarnation indexes: This method produces smaller and more efficient circuits, but its conceptual complexity is high. Although we have empirically validated it through extensive testing, there is no formal proof of its correctness.
- Sequential code for non-cyclic circuits: The HipHop compiler can also generate sequential code for non-cyclic circuits. Circuits that can be compiled this way are about 10 times faster than those that need to be executed through the simulation of an electric circuit.

To validate these three compilers, we developed a new testing platform based on principles from the Haskell QuickCheck tool. The platform generates and compares the execution of randomly generated programs across different compilers. This work was completed by the end of 2025. Both the new compilers and the testing platform will be described in a paper to be written in 2026.

### 7.3 Systems for the New Spectre Era

#### 7.3.1 Software

**Participants:** Davide Davoli, Tamara Rezk.

Operating systems are stratified software ecosystems that manage the interaction between the user application and the hardware resources. The most important component is the *kernel* that has complete control over the hardware and every application that is running, making it a popular target for attacks. In the current state-of-the-art of security, often referred to as the *Spectre era*, speculative execution and side-channels are well known to be effective vectors for compromising kernel security.

We show that kernel safety cannot be restored for attackers capable of using side-channels and speculative execution, and introduce enforcement mechanisms that can guarantee speculative kernel safety for safe system calls in the Spectre era. We implement three suitable mechanisms and we evaluate their performance overhead on the Linux kernel.

This work has been done with Martin Avanzini from the OLAS team, who co-supervised Davide Davoli together with Tamara Rezk, and published in the ACM Transactions on Privacy and Security journal [3].

#### 7.3.2 Preservation of speculative constant time

**Participants:** Benjamin Grégoire.

Compilers often weaken or even discard software-based countermeasures commonly used to protect programs against side-channel attacks; worse, they may also introduce vulnerabilities that attackers can exploit. The solution to this problem is to develop compilers that preserve such countermeasures. Prior work establishes that (a mildly modified version of) the CompCert and Jasmin formally verified compilers preserve constant-time, an information flow policy that ensures that programs are protected against timing side-channel attacks. However, nothing is known about preservation of speculative constant-time, a strengthening of the constant-time policy that ensures that programs are protected against Spectre-v1 attacks. We first show that preservation of speculative constant-time fails in practice by providing examples of secure programs whose compilation is not speculative constant-time using GCC (GCC -O0 and GCC -O1) and Jasmin. Then, we have defined a proof-of-concept compiler that distills some of the critical passes of the Jasmin compiler and use the Rock proof assistant to prove that it preserves speculative constant-time. Finally, we patch the Jasmin speculative constant-time type checker and demonstrate that all cryptographic implementations written in Jasmin can be fixed with minimal impact. This work [1] has been published in ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2025).

#### 7.3.3 Protection against Spectre RSB

**Participants:** Benjamin Grégoire.

Recent work shows that language support suffices to protect cryptographic code with minimal overhead against one class of Spectre attacks, Spectre-PHT, but leaves an open question of whether this result can be extended to also cover other classes of Spectre attacks. In this work, we answer this question in the affirmative: We design, validate, implement, and verify an approach to protect cryptographic implementations against all

known classes of Spectre attacks—the main challenge in this endeavor is attacks exploiting the return stack buffer, which are known as Spectre-RSB. Our approach combines a new value-dependent information-flow type system that enforces speculative constant-time in an idealized model of transient execution and a compiler transformation that realizes this idealized model on the generated low-level code. Using the Rocq proof assistant, we prove that the type system is sound with respect to the idealized semantics and that the compiler transformation preserves speculative constant-time. We implement our approach in the Jasmin framework for high-assurance cryptography and demonstrate that the overhead incurred by full Spectre protections is below 2% for most cryptographic primitives and reaches only about 5–7% for the more complex post-quantum key-encapsulation mechanism Kyber. This work [6] has been published in the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2025).

### 7.3.4 Using Intel’s Extended HW/SW Contract for Secure Compilation of Crypto Code

**Participants:** Benjamin Grégoire.

It is a widely accepted standard practice to implement cryptographic software so that secret inputs do not influence the cycle count. Software following this paradigm is often referred to as “constant-time” software and typically involves following three rules: 1) never branch on a secret-dependent condition, 2) never access memory at a secret-dependent location, and 3) avoid variable-time arithmetic operations on secret data. The third rule requires knowledge about such variable-time arithmetic instructions, or vice versa, which operations are safe to use on secret inputs. For a long time, this knowledge was based on either documentation or microbenchmarks, but critically, there were never any guarantees for future microarchitectures. This changed with the introduction of the data-operand-independent-timing (DOIT) mode on Intel CPUs and, to some extent, the data-independent-timing (DIT) mode on Arm CPUs. Both Intel and Arm document a subset of their respective instruction sets that are intended to leak no information about their inputs through timing, even on future microarchitectures if the CPU is set to run in a dedicated DOIT (or DIT) mode. We have presented a principled solution that leverages DOIT to enable cryptographic software that is future-proof constant-time, in the sense that it ensures that only instructions from the DOIT subset are used to operate on secret data, even during speculative execution after a mispredicted branch or function return location. For this solution, we build on top of existing security type systems in the Jasmin framework for high-assurance cryptography. We then use our solution to evaluate the extent to which existing cryptographic software built to be “constant-time” is already secure in this stricter paradigm implied by DOIT and what is the performance impact to move from constant-time to future-proof constant-time. This work [2] has been published in the journal Transactions on Cryptographic Hardware and Embedded Systems (TCHES 2025).

### 7.3.5 Hardware

**Participants:** Davide Davoli, Lucile Magnier, Tamara Rezk, Benjamin Grégoire.

In the past, we proposed ProSpeCT [14], a generic formal processor model providing provably secure speculation for the constant-time policy. For constant-time programs under a non-speculative semantics, ProSpeCT guarantees that speculative and out-of-order execution cause no microarchitectural leaks. In addition to the formal model, we provided a prototype hardware implementation of ProSpeCT on a RISC-V processor. In order to lower the impact on hardware cost and the cost regarding the required software changes, we are studying possible alternative solutions to ProSpeCT, extending the RISC-V architecture and associated compilers with specialized instructions and evaluating the performance on different extended RISC-V processors. Lucile Magnier from ENS Saclay was a L3 intern during part of this work (which is still in progress). This work is done in collaborations with researchers from KU Leuven.

## 7.4 Formal method for cryptography

### 7.4.1 A Quantitative Probabilistic Relational Hoare Logic

**Participants:** Benjamin Grégoire, Davide Davoli.

We introduce eRHL, a program logic for reasoning about relational expectation properties of pairs of probabilistic programs. eRHL is quantitative, i.e., its pre- and post-conditions take values in the extended non-negative reals. Thanks to its quantitative assertions, eRHL overcomes randomness alignment restrictions from prior logics, including PRHL, a popular relational program logic used to reason about security of cryptographic constructions, and apRHL, a variant of PRHL for differential privacy. As a result, eRHL is the first relational probabilistic program logic to be supported by non-trivial soundness and completeness results for all almost surely terminating programs. We show that eRHL is sound and complete with respect to program equivalence, statistical distance, and differential privacy. We also show that every PRHL judgment is valid if and only if it is provable in eRHL. We showcase the practical benefits of eRHL with examples that are beyond reach of PRHL and apRHL. This work [7] has been published in ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2025).

### 7.4.2 Proofs automation

**Participants:** Benjamin Grégoire.

Jasmin is a programming language for high-speed and high-assurance cryptography. Correctness proofs of Jasmin programs are typically carried out deductively in EasyCrypt. This allows generality, modularity and composable reasoning, but does not scale well for low-level architecture-specific routines. CryptoLine offers a semi-automatic approach to formally verify algebraically-rich low-level cryptographic routines. CryptoLine proofs are self-contained: they are not integrated into higher-level formal verification developments. We show how to soundly use CryptoLine to discharge subgoals in functional correctness proofs for complex Jasmin programs. We extend Jasmin with annotations and provide an automatic translation into a CryptoLine model, where most complex transformations are certified. We also formalize and implement the automatic extraction of the semantics of a CryptoLine proof to EasyCrypt. Our motivating use-case is the X-Wing hybrid KEM, for which we present the first formally verified implementation. This paper [5] has been published to the ACM Conference on Computer and Communications Security (CCS 2025), and has obtained a Distinguished Paper Award.

## 7.5 ITree semantics of Jasmin

**Participants:** Benjamin Grégoire, Paolo Torrini.

We have replaced the old semantics of the Jasmin compiler front-end with a new one based on Interaction Trees (ITrees), allowing us to take non-terminating programs into account. Our new semantics relies on two types of ITree events, along lines already known in the literature: recursive calls and errors. This suffices to introduce recursive functions unrestrictedly in the Jasmin semantics, and to deal with error propagation in a clean way. In order to keep our compiler correctness proofs as simple as possible, we want to conclude them trivially in case the source program goes wrong. To this purpose, we have defined a program equivalence relation which makes it possible to take advantage of undefined behavior, and in particular to match errors raised in the source program with arbitrary target code. An extended abstract on this work has been presented at Rocqshop'25, affiliated to ITP'25 [11]. On top of this, we have defined a Jasmin-specific relational Hoare logic which, while supporting extensive reuse, has helped us make the new proof more structured and concise. We have further used ITrees to define a probabilistic semantics of Jasmin programs which has been used to verify security properties. This work has been submitted for publication.

## 7.6 Session types

**Participants:** Ilaria Castellani.

Session types describe communication protocols involving two or more participants by specifying the sequence of exchanged messages and their functionality (sender, receiver and type of carried data). They may be viewed as the analog, for concurrency and distribution, of data types for sequential computation. Originally conceived as a static analysis technique for a variant of the  $\pi$ -calculus, session types have been progressively embedded into a range of functional, concurrent, and object-oriented programming languages.

The aim of session types is to ensure safety properties for sessions, such as *communication safety*, i.e. the absence of communication errors (no type mismatch in exchanged data). In the absence of session interleaving, session types additionally ensure the property of *deadlock-freedom* (no standstill until all participants are terminated). When describing multiparty protocols, session types often target also the liveness property of *progress* or *lock-freedom* (no participant waits forever).

While binary sessions can be described by a single session type, multiparty sessions require two kinds of types: a *global type* that describes the whole session protocol, and *local types* that describe the individual contributions of the participants to the protocol. The key requirement to achieve safety properties such as deadlock-freedom is that the local types of the processes implementing the participants be obtained as projections from the same global type. To ensure progress, global types must also satisfy some well-formedness requirements.

What makes session types particularly attractive is that they offer several advantages at once: 1) static safety guarantees, 2) automatic check of protocol implementation correctness, based on local types, and 3) a strong connection with linear logics and with concurrency models such as communicating automata, graphical choreographies and message-sequence charts.

We have pursued our work on multiparty session types during this year, but this work did not lead to any publication in 2025.

## 7.7 Constructive characterizations of the must-preorder for asynchrony

**Participants:** Ilaria Castellani.

This work was carried out in collaboration with Giovanni Bernardi (IRIF, Université Paris Cité), Paul Laforgue (IRIF and Nomadic Labs) and Léo Stefanescu (MPI-SWS - Max Planck Institute for Software Systems, Kaiserslautern, now moved to University of Cambridge).

The must-preorder by De Nicola and Hennessy is a contextual refinement which states that a server  $q$  refines a server  $p$  if all clients satisfied by  $p$  are also satisfied by  $q$ . Due to the universal quantification over clients, this definition does not yield a practical proof method for the must-preorder, and alternative characterizations are necessary to reason over it. Finding these characterizations for asynchronous semantics, where outputs are non-blocking, has thus far proven to be a challenge, usually tackled via ad-hoc definitions. We show that the standard characterizations of the must-preorder, as given for synchronous semantics, carry over as they stand to asynchronous communication, provided that servers are enhanced to act as forwarders, i.e. they are given the ability to input any message as long as they store it back into the shared buffer. Our development is constructive, completely mechanized in Rocq, and independent of any calculus since it is carried out on Selinger output-buffered agents with feedback, a class of Labeled Transition Systems (LTSs) satisfying a number of axioms. This class of LTSs represents programs that communicate via a shared unordered buffer, as in the asynchronous Calculus of Communicating Systems (CCS) or the asynchronous  $\pi$ -calculus. We show that the standard coinductive characterization allows us to prove in Rocq that concrete programs are related by the must-preorder. Finally, our proofs demonstrate that Brouwer's bar induction principle is a useful technique to reason on liveness preserving program transformations.

This work was presented at the conference ESOP 2025 [12].

## 8 Partnerships and cooperations

### 8.1 International initiatives

#### 8.1.1 Inria associate team not involved in an IIL or an international program

**Participants:** Erven Rohou, Manuel Serrano, Marc Feeley.

**Title:** COLD

**Partner Institution(s):** Inria Rennes (leader), University of Montreal (CANADA)

Cold is an Inria/University of Montreal associated team studying compilation techniques for dynamic languages. It is co-funded by Inria and the University of Montreal. Erven Rohou is the leader of the team.

Dynamic programming languages offer flexibility and generally facilitate rapid software development. Programs written using dynamic languages are typically slower, consume more memory, and are less energy efficient. This is especially concerning, considering that dynamic languages such as Python and JavaScript are extensively used. JavaScript is the main language for implementing web applications, while Python is the most used language for software development today and in particular in the very active field of Machine Learning and Artificial Intelligence.

To solve this issue, our team researches optimizing compilation techniques for dynamic languages. Such techniques generate optimized code when translating a program from its source code to machine code. This provides better performance without having to sacrifice the flexibility of dynamic languages. Furthermore, since novel optimizing techniques can be integrated into existing compilers, they can improve current programs with no additional effort by the application programmers.

In the last decades, the discovery of tracing just-in-time compilation and advances in partial evaluation have significantly improved the performance of dynamic languages. However, work remains to be done to bridge the gap between dynamic and static languages, a performant yet less flexible family of languages. Bridging this gap will enable significant savings in execution time and energy efficiency globally.

#### 8.1.2 Participation in other International Programs

##### Retrofit (ANR/NSF)

**Participants:** Manuel Serrano.

**Title:** Bringing Esterel out of its Shell

**Partner Institution(s):** Northwestern University, USA

**Date/Duration:** May 1st, 2025 42 months

Synchronous reactive programming languages have proven successful for real-time, safety-critical applications, such as aviation and nuclear power. We consider it a mistake that these languages have not been extended to other domains that require responsiveness to interacting agents with strict attention to sequencing and time bounds. In our view, the Web, the IoT, and medical care are neglected domains where synchronous reactive languages have potential to bring order-of-magnitude improvements in reliability and responsiveness. In the Retrofit project, we aim to demonstrate the benefit of synchronous reactive programming for general programming. For that, Esterel will be our main source of inspiration as we create new synchronous reactive languages for these domains. To assess the benefits of synchronous reactive programming we will build and assess realistic dynamic prototypes in these new languages.

## 8.2 International research visitors

### 8.2.1 Visits of international scientists

- Gilles Barthe

**Affiliation:** MPI, Bochum, Germany

**Subject:** We worked on several topics related to the Formosa cryptography project, in particular on Jasmin and EasyCrypt, as well as on the implementation of the eRHL logic, in collaboration with Martin Avanzini (OLAS).

**Date/Duration:** May 19-30, August 25 to Sep 16, Oct 27 to Nov 5, Dec 8 to 19

- Marc Feeley

**Affiliation:** University of Montreal, Quebec, Canada

**Subject:** During the visit of M. Feeley, we studied new implementation techniques for dynamic languages

**Date/Duration:** July 2025

### 8.2.2 Visits to international teams

#### Research stays abroad

Manuel Serrano

**Visited institution:** University of Montreal (Prof. M. Feeley)

**Country:** Canada

**Dates:** September

**Context of the visit:** COLD (Inria PACAP associate team)

**Mobility program/type of mobility:** research stay

## 8.3 National initiatives

### 8.3.1 Action Exploratoire: AoT.js – Optimizing Compilation from Higher-Order Programming to Computer Architecture

**Participants:** Erven Rohou, Manuel Serrano.

This *action exploratoire* is bi-localized in Rennes and Sophia-Antipolis.

JavaScript programs are typically executed by a JIT compiler, able to handle efficiently the dynamic aspects of the language. However, JIT compilers are not always viable or sensible (*e.g.*, on constrained IoT systems, due to secured read-only memory ( $W\oplus X$ ), or because of the energy spent recompiling again and again). We propose to rely on ahead-of-time compilation, and achieve performance thanks to optimistic compilation, and detailed analysis of the behavior of the processor, thus requiring a wide range of expertise from high-level dynamic languages to microarchitecture.

### 8.3.2 PEPR

**Participant:** Benjamin Grégoire, Jean-Christophe L  chenet, Paolo Torrini.

**Title:** SVP PEPR Cybersecurity. ]

**Partner Institution(s):** CNRS IRISA Rennes (coordinator St  phanie Delaune), Inria, University of Paris-Saclay, University of Lorraine, University C  te d'Azur, ENS Rennes

**Date/Duration: 2022-2028**

We participate in a project concerned with the verification of security protocols. The funds allocated to our team in this collaboration are 333 KEuros. The corresponding researcher for this contract is Benjamin Grégoire.

**8.3.3 HOPR**

**Participant:** Benjamin Grégoire.

HOPR (Higher-Order Probabilistic and resource-aware Reasoning) aims at investigating logical and semantical frameworks to reason on computation including higher-order, probabilistic and resource-bounded aspects. It explores applications to cryptography and to differential privacy. This project started in January 2025, for a duration of 4 years.

**8.3.4 Twinsec**

**Participant:** Tamara Rezk.

Title: Digital Twin for Hardware Security

Partners: CEA, GRENoble INP-UGA, CNRS, Mines Saint Etienne, CentraleSupélec

Duration: From December 16, 2024 to February 15, 2028

Abstract: The TwinSec project, entitled “Digital Twin for Hardware Security,” is conducted under the CEA High-Risk Research Program, as defined in the grant agreement “ANR-24-RR11-0004” between the French National Research Agency (ANR) and the CEA. Within this project, SPLITS contributes to the work package focused on software–hardware contracts for binary analysis.

**8.4 Regional initiatives****8.4.1 Actions de Développement Technologique**

**Participant:** Benjamin Grégoire, Côme Le Breton, Jean-Christophe L chenet.

We carried out an *Action de D veloppement Technologique* initiative with SED engineers from the AMDT team. The goal was to improve an important component of the Jasmin framework, the safety checker. While the Jasmin compiler had been updated in 2022 to support multiple target architectures, the same move had not been performed yet for the safety checker, still available only on x86. This project made the safety checker also available on ARM and RISC-V.

**9 Dissemination****9.1 Promoting scientific activities****9.1.1 Scientific events: organization**

Tamara Rezk organized together with Martin Avanzini and Davide Sangiorgi from OLAS, a workshop entitled “TSuNAMI — Trusted and Secure systems: Novel Approaches in Methods and Inference” at Inria, June 10, 2025.

### 9.1.2 Scientific events: selection

#### Chair of conference program committees

- Tamara Rezk was Program Track Chair of ACM CCS 2025.
- Manuel Serrano was Associate Program Chair of ACM ICFP 2025.

#### Member of the conference program committees

- Tamara Rezk participated in the program committees of GTMF25, MadWeb25, Prisc25, and Usenix Security 2025.
- Benjamin Grégoire participated in the program committee of CSF25.
- Ilaria Castellani participated in the program committee of COORDINATION 2025: 27th International Conference on Coordination Models and Languages.
- Ilaria Castellani is a member of the Steering Committees of the conference COORDINATION and of the workshop EXPRESS/SOS.

**Reviewer** Benjamin Grégoire was a reviewer for ICLP 2025

### 9.1.3 Journal

#### Member of the editorial boards

- Manuel Serrano is member of the **Programming Steering Committee** and the ACM ICFP Steering Committee.
- Tamara Rezk is a member of the CSF and FCS steering committees.

### 9.1.4 Invited talks

- Tamara Rezk was:
  - Invited speaker at the **10th AMSEC Workshop on system security**, "On Kernel's Safety in the Spectre Era", Amsterdam, March 18, 2025.
  - Keynote speaker at the Chalmers ICT Security Workshop, "Micro-architectural attacks, a crucial intersection of software and hardware security", Gothenburg, March 20, 2025,
  - Speaker at the Stevens Institute of Technology, "On Kernel's Safety in the Spectre Era", New York, July 11, 2025.

### 9.1.5 Scientific expertise

- Tamara Rezk participated in the committee of iCore 2025 to rank security conferences.
- Tamara Rezk participated in the committee of WWTF 2025 to select project proposals for funding.
- Tamara Rezk was a member of the Committee Meeting for promotion to MdC level 1 at Eurecom in June 2025.
- Tamara Rezk was a member of the Expert Panel of Mathematics, Computer Science, and Informatics of the Estonian Research Council in May 2025.

### 9.1.6 Research administration

- Ilaria Castellani is a member of the INRIA Gender Equality and Equal Opportunities Committee.
- Tamara Rezk is a member of the Bureau de CP.
- Benjamin Grégoire is a member of the Comité de Suivi Doctoral.

## 9.2 Teaching - Supervision - Juries - Educational and pedagogical outreach

### 9.2.1 Teaching

Tamara Rezk taught courses in Université Côte d’Azur, Chalmers, and École Polytechnique.

### 9.2.2 Supervision

- PhD defended in November 2025 [13]: Davide Davoli, co-supervised by Martin Avanzini and Tamara Rezk.
- PhD discontinued: Aurore Poirier, co-supervised by Erven Rohou and Manuel Serrano.
- PhD in progress: Olivier Melancon, co-supervised by Marc Feeley and Manuel Serrano.
- PhD in progress: Lucas Tabari-Maujean, co-supervised by Thomas Roches and Benjamin Grégoire.
- PhD in progress: Guilhem Mizrahi, co-supervised by Pierre-Yves Strub, Adrian Thillard, and Benjamin Grégoire.
- PhD in progress: Mehdi Golpayegani (officially in the OLAS team), co-supervised by Martin Avanzini (OLAS) and Benjamin Grégoire.
- Master in progress: Maria Emilia Caldara (University of Córdoba, Argentina) is working on a master dissertation under the supervision of Tamara Rezk.

### 9.2.3 Juries

- Tamara Rezk was a member of the PhD jury of Afiqah AZAHARI (supervisor: Davide Balzarotti), Eurecom, December 2025.
- Tamara Rezk was a member of the PhD jury of Enrico Barberis (supervisor: Herbert Bos), VU Amsterdam, March 2025.
- Tamara Rezk was a member of the HDR jury of Ronan Lashermes, May 2025.
- Benjamin Grégoire was a reviewer for the PhD of Pierrick PHILIPPE (supervisors: Pierre-Alain Fouque and Mohamed Sabt).
- Ilaria Castellani was a reviewer for the PhD thesis of Andrea Esposito (supervisor: Marco Bernardo), Università di Urbino Carlo Bo, Italy, May 2025.
- Manuel Serrano was a reviewer for the PhD thesis of Cameron Moy (supervisor: Matthias Felleisen), University of Northeastern, USA, December 2025.

## 10 Scientific production

### 10.1 Publications of the year

#### International journals

- [1] S. Arranz Olmos, G. Barthe, L. Blatter, B. Grégoire and V. Laporte. ‘Preservation of Speculative Constant-time by Compilation’. In: *Proceedings of the ACM on Programming Languages* 9.POPL (9th Jan. 2025), pp. 1293–1325. DOI: [10.1145/3704880](https://doi.org/10.1145/3704880). URL: <https://hal.univ-lorraine.fr/hal-04663857> (cit. on p. 12).
- [2] S. Arranz-Olmos, G. Barthe, B. Grégoire, J. Jancar, V. Laporte, T. Oliveira and P. Schwabe. ‘Let’s DOIT: Using Intel’s Extended HW/SW Contract for Secure Compilation of Crypto Code’. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.3 (5th June 2025), pp. 644–667. DOI: [10.46586/tches.v2025.i3.644-667](https://doi.org/10.46586/tches.v2025.i3.644-667). URL: <https://hal.univ-lorraine.fr/hal-05249675> (cit. on p. 13).

- [3] D. Davoli, M. Avanzini and T. Rezk. ‘Comprehensive Kernel Safety in the Spectre Era: Mitigations and Performance Evaluation’. In: *ACM Transactions on Privacy and Security* (12th June 2025). DOI: [10.1145/3743678](https://doi.org/10.1145/3743678). URL: <https://hal.science/hal-05173527> (cit. on p. 12).
- [4] A. Poirier, E. Rohou and M. Serrano. ‘An Attempt to Catch Up with JIT Compilers: The False Lead of Optimizing Inline Caches’. In: *The Art, Science, and Engineering of Programming* 10.6 (15th Feb. 2025). DOI: [10.22152/programming-journal.org/2025/10/6](https://doi.org/10.22152/programming-journal.org/2025/10/6). URL: <https://hal.science/hal-04904428> (cit. on p. 11).

#### International peer-reviewed conferences

- [5] J. B. Almeida, M. Barbosa, G. Barthe, L. Blatter, G. Delerue, J. D. Duarte, B. Grégoire, T. Oliveira, M. Quaresma, P.-Y. Strub, M.-H. Tsai, B.-Y. Wang and B.-Y. Yang. ‘Jazzline: Composable CryptoLine Functional Correctness Proofs for Jasmin Programs’. In: *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*. CCS 2025 - 2025 ACM SIGSAC Conference on Computer and Communications Security. Taipei, Taiwan, 19th Nov. 2025, pp. 1409–1423. DOI: [10.1145/3719027.3744814](https://doi.org/10.1145/3719027.3744814). URL: <https://inria.hal.science/hal-05466117> (cit. on pp. 7, 14).
- [6] S. Arranz Olmos, G. Barthe, C. Chuengsatiansup, B. Grégoire, V. Laporte, T. Oliveira, P. Schwabe, Y. Yarom and Z. Zhang. ‘Protecting cryptographic code against Spectre-RSB (and, in fact, all known Spectre variants)’. In: *ASPLOS ’25: 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Vol. 2. Rotterdam, Netherlands: ACM, 30th Mar. 2025, pp. 933–948. DOI: [10.1145/3676641.3716015](https://doi.org/10.1145/3676641.3716015). URL: <https://inria.hal.science/hal-04632106> (cit. on p. 13).
- [7] M. Avanzini, G. Barthe, D. Davoli and B. Grégoire. ‘A Quantitative Probabilistic Relational Hoare Logic’. In: *ACM Digital Library*. POPL 2025 - 52nd ACM SIGPLAN Symposium on Principles of Programming Languages. Vol. 9. Proceedings of the ACM on Programming Languages POLP 2025. Denver (Colorado), United States, 20th Jan. 2025. DOI: [10.1145/3704876](https://doi.org/10.1145/3704876). URL: <https://inria.hal.science/hal-04834149> (cit. on p. 14).
- [8] J.-C. Léchenet. ‘Mechanized Dominator Tree Certification’. In: *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP ’26)*. CPP 2026 - 15th ACM SIGPLAN International Conference on Certified Programs and Proofs. Rennes, France: ACM, Jan. 2026. DOI: [10.1145/3779031.3779107](https://doi.org/10.1145/3779031.3779107). URL: <https://inria.hal.science/hal-05413838>.
- [9] O. Melançon, M. Serrano and M. Feeley. ‘Float Self-Tagging’. In: *OOPSLA 2025*. Vol. 9. OOPSLA2. Singapour, Singapore, 9th Oct. 2025, pp. 1620–1646. DOI: [10.1145/3763108](https://doi.org/10.1145/3763108). URL: <https://inria.hal.science/hal-05440858> (cit. on p. 11).
- [10] M. Serrano and R. B. Findler. ‘A Snapshot of the Performance of Wasm Backends for Managed Languages’. In: *MPLR ’25: 22nd ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*. Singapore Singapore, Singapore: ACM, 12th Oct. 2025, pp. 93–106. DOI: [10.1145/3759426.3760983](https://doi.org/10.1145/3759426.3760983). URL: <https://inria.hal.science/hal-05440850> (cit. on p. 10).
- [11] P. Torrini and B. Gregoire. ‘Interaction Trees and Verified Compilation (Extended Abstract)’. In: <https://coq-workshop.gitlab.io/2025>. Rocqshop’25. Reykjavik, Iceland, 27th Sept. 2025, p. 3. URL: <https://inria.hal.science/hal-05482317> (cit. on p. 14).

#### Conferences without proceedings

- [12] G. Bernardi, I. Castellani, P. Laforgue and L. Stefanescu. ‘Constructive characterisations of the must-preorder for asynchrony’. In: *ESOP*. Vol. 15694. Lecture Notes in Computer Science. Hamilton, Ontario, Canada, Canada: Springer Nature Switzerland, 1st May 2025, pp. 88–116. DOI: [10.4230/LIPIcs](https://doi.org/10.4230/LIPIcs). URL: <https://hal.science/hal-04642776> (cit. on p. 15).

**Doctoral dissertations and habilitation theses**

- [13] D. Davoli. ‘Provable security for kernels and cryptography in the presence of speculative execution’. Université Côte d’Azur, 6th Oct. 2025. URL: <https://theses.hal.science/tel-05386724> (cit. on p. 20).

**10.2 Cited publications**

- [14] L.-A. Daniel, M. Bognar, J. Noorman, S. Bardin, T. Rezk and F. Piessens. ‘ProSpeCT: Provably Secure Speculation for the Constant-Time Policy’. In: USENIX Security Symposium. Anaheim, France, 9th Aug. 2023. URL: <https://inria.hal.science/hal-04389436> (cit. on p. 13).