RESEARCH CENTRE

Inria Centre
at Université Côte d'Azur

2023 ACTIVITY REPORT

Project-Team SPLITS

Secure Programming Languages & Tools for Security

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Security and Confidentiality

Inria

Contents

Project-Team SPLITS

Creation of the Project-Team: 2023 July 01

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
- A2.4.1. – Analysis
- A2.4.3. – Proofs

Other research topics and application domains

- B6. – IT and telecom

1 Team members, visitors, external collaborators

Research Scientists

- Tamara Rezk [Team leader, INRIA, Senior Researcher, from Jul 2023, HDR]
- Ilaria Castellani [INRIA, Researcher, from Jul 2023]
- Benjamin Grégoire [INRIA, Researcher, from Jul 2023]
- Manuel Serrano [INRIA, Senior Researcher, from Jul 2023, HDR]

PhD Students

- Guillaume Combette [CEA, from Jun 2023, CEA]
- Davide Davoli [UNIV COTE D'AZUR , from Jul 2023]
- Aurore Poirier [Inria, from Jun 2023, Pacap]
- Swarn Priya [Inria, from Jun 2023, STAMP]
- Ignacio Tiraboschi [Inria, from Jun 2023, Antique]

Technical Staff

- Jean-Christophe Léchenet [INRIA, Engineer, from Oct 2023]

Administrative Assistant

- Christine Foggia [INRIA, from Mar 2023]

Visiting Scientist

- Robert Findler [Northwestern University, from Jun 2023]

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 into 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 π-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 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 (for now the code is targeted to the x86 architecture). The compiler is formally verified in the Coq 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 closed 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 Jasmin compilers 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 [1] 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

- Swarn Priya (Phd student of Benjamin Grégoire) was awarded with the Young Talents "Pour les femmes et la science" L'Oréal-UNESCO 2023 prize. She defended her PhD on November 21st, 2023. (Swarn Priya was officially affiliated to STAMP.)

- Tamara Rezk received the 2023 Distinguished Service Award for outstanding contributions as PC chair of CSF'23. The award was granted by IEEE Computer Security (Technical Community on Security and Privacy).

[1] Example CVEs for all major hardware industries: CVE-2020-0551, CVE-2021-26401, CVE-2022-23960

6 New software, platforms, open data

6.1 New software

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.

Release Contributions: This version introduces a new logic (ehoare) allowing to bound the expectation of a function in a probabilistic program.

News of the Year: The major release (2023.09) has been published. This release includes a new logic for bounding the expectation of a function in a probabilistic program.

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

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 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: 2023.06.0 is a major release of Jasmin. It contains a few noteworthy changes: - local functions now use call and ret instructions, - experimental support for the ARMv7 (i.e., Cortex-M4) architecture, - a few aspects of the safety checker can be finely controlled through annotations or command-line flags, - shift and rotation operators have a simpler semantics.

As usual, it also brings in various fixes and improvements, such as bit rotation operators and automatic slicing of the input program.

News of the Year: On June 2023, a major release (2023.06.0) has been published.

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

Publications: hal-04106448, hal-04218417, hal-03844366, hal-03430789, hal-03352062, hal-02404581, hal-02974993, hal-01649140

Contact: Jean-Christophe Léchenet

Participants: 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/indes/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-dev.inria.fr/hiphop

Contact: 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 Hop (also sometimes refereed to as Hop.js in the rest of this text) and our study on efficient JavaScript implementations as well as our development of analyses for distributed language sessions and security. These contributions concern improvements of the hopc compiler. They have been integrated in the new version that is about to be released and only described in unpublished internal reports. We also pursued our collaboration with the University of Montréal on the compilation of the Python programming language.

7.1.1 Debunking the Performance of Asynchronous JavaScript

In this study, we compared the performance of server-side JavaScript applications programmed using synchronous, asynchronous, and promise-based APIs. Our findings show that, in general, synchronous programs execute significantly faster and consume fewer resources compared to their asynchronous counterparts. The only situation where we observed better performance for the asynchronous model was in the case of a highly loaded web server. For that application, under that exceptional context, we noticed that the asynchronous approach enables servers to degrade more gracefully than the synchronous one. This suggests that an hybrid architecture that could combine the best of both worlds, synchronous operations under normal situations and asynchronous operations under super-heavy pressure, would be optimal. The Hop.js execution environment that supports both modes is an ideal playground for conducting this sort of experiment.

We further demonstrated that the overhead incurred by asynchronous executions is a combination of complex control flow, which makes it challenging for compilers to apply simple reduction optimizations, additional memory allocations, and high thread synchronization costs. The asynchronous model also requires intensive collaborations between JavaScript and the implementation language, with numerous exchanges between the two worlds. This places demands on the JavaScript automatic memory management system to track values passed to the implementation language.

This experiment also unveiled that a highly optimized and tuned JavaScript runtime can be defeated by a more general runtime system that enables smoother integration between the core language and foreign libraries. This suggests that such systems may be suitable when intensive collaborations between the core language and foreign libraries are expected. This is an incentive for pursuing the development of the generic runtime system hopc uses.

7.1.2 Optimizing JavaScript Variable-Arity Functions in the Large

JavaScript allows functions to be called with a different number of arguments than the specified number of parameters. The pseudo-variable `arguments` packs all the arguments into a heap-allocated structure and enables the receiver to introspect the received values. The design and semantics of `arguments` make it difficult to implement efficiently but it is used by millions or even maybe billions of applications. Therefore, improving it may have a significant global impact on computing resources.

We conducted a study on how modern JavaScript code utilizes the `arguments` property. By analyzing NPM, the largest JavaScript repository, we demonstrated that despite the introduction of alternative constructs to support variable-arity functions, the historical `arguments` property is still widely used, which is unfortunate so difficult it is to implement efficiently. To address this issue, we presented an

optimization that significantly accelerates its performance. The optimization can either completely eliminate the construction of the `arguments` object or transform the usual heap allocation, which a generic implementation employs, into a faster stack allocation. We validated the effectiveness of this optimization through a set of micro-benchmarks. When applied to the Hop.js compiler, the optimization resulted in performance gains of one to two orders of magnitude compared to the generic code. The analysis and optimization can also be adopted by JIT compilers, as they rely on fast local analyses instead of lengthy analysis of the entire program.

While the optimization may not have a spectacular impact on individual programs, its global effect could be significant. Our study revealed that approximately half of the 3 million JavaScript packages available on NPM directly or indirectly depend on one or several packages that use `arguments`. As a result, our optimization has the potential to accelerate around 1.5 million packages, which likely play a role in millions or billions of real applications. Considering the collective impact of optimizing all these executions, the potential benefits could be tremendous. We reckon that this compelling motivation justified our undertaking of this study.

7.1.3 An Executable Semantics for Faster Development of Optimizing Python Compilers

Python is a popular programming language whose performance is known to be uncompetitive in comparison to static languages such as C. Although significant efforts have already accelerated implementations of the language, more efficient ones are still required. The development of such optimized implementations is nevertheless hampered by its complex semantics and the lack of an official formal semantics. We addressed this issue by developing an approach to define an executable semantics targeting the development of optimizing compilers. This executable semantics is written in a format that highlights type checks, primitive values boxing and unboxing, and function calls which are all known sources of overhead. We also developed semPy, a partial evaluator of our executable semantics that can be used to remove redundant operations when evaluating arithmetic operators.

To validate our approach, we integrated these behaviors to Zipi, an AoT optimizing Python compiler prototype. Zipi compiles behaviors and dispatches operations to their corresponding behaviors at run time. This increases execution speed, offering performance that rivals PyPy. Although this speedup is limited to arithmetic-heavy programs, behaviors could be extended to other operations or serve alongside other optimization techniques. On some tasks, Zipi displays performance competitive with that of state of art Python implementations.

We hope semPy and the behavior optimization can contribute to the ongoing optimization efforts of Python implementations. It appears to us that they would be well suited for CPython, as they specifically address the known overhead of this implementation.

This work has been presented at the 16th ACM SIGPLAN International Conference on Software Language Engineering (SLE'23) conference [8].

7.2 Combining Symbolic Execution and Abstract Interpretation

Participants: Ignacio Tiraboschi, Tamara Rezk.

Symbolic execution is a program analysis technique commonly utilized to determine whether programs violate properties and, in case violations are found, to generate inputs that can trigger them. Used in the context of security properties such as noninterference, symbolic execution is precise when looking for counterexample pairs of traces when insecure information flows are found, however it is sound only for a subset of executions. Thus, it does not allow to prove the correctness of programs with executions beyond the given bound. By contrast, abstract interpretation-based static analysis guarantees soundness but generally lacks the ability to provide counterexample pairs of traces. In this paper, we propose to weave both to obtain the best of two worlds. We demonstrate this with a series of static analyses, including a static analysis called RedSoundRSE aimed at verifying noninterference. RedSoundRSE provides both semantically sound results and the ability to derive counterexample pairs of traces up to a bound. It relies on a combination of symbolic execution and abstract domains inspired by the well known notion

of reduced product. We formalize RedSoundRSE and prove its soundness as well as its relative precision up to a bound. We also provide a prototype implementation of RedSoundRSE and evaluate it on a sample of challenging examples.

This work has been presented at VMCAI'23 [9].

7.3 Hardware for the New Spectre Era

Participants: Tamara Rezk.

We propose ProSpeCT, 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. This guarantee is achieved by tracking secrets in the processor pipeline and ensuring that they do not influence the microarchitectural state during speculative execution. Our formalization covers a broad class of speculation mechanisms, generalizing prior work. As a result, our security proof covers all known Spectre attacks, including load value injection (LVI) attacks.

In addition to the formal model, we provide a prototype hardware implementation of ProSpeCT on a RISC-V processor and show evidence of its low impact on hardware cost, performance, and required software changes. In particular, the experimental evaluation confirms our expectation that for a compliant constant-time binary, enabling ProSpeCT incurs no performance overhead.

This work has been presented at USENIX Security 2023 [7].

7.4 Secret Erasure

7.4.1 Binary Code Analyzer

Participants: Tamara Rezk.

We tackle the problem of designing efficient binary-level verification for a subset of information flow properties encompassing constant-time and secret-erasure. These properties are crucial for cryptographic implementations, but are generally not preserved by compilers. Our proposal builds on relational symbolic execution enhanced with new optimizations dedicated to information flow and binary-level analysis, yielding a dramatic improvement over prior work based on symbolic execution. We implement a prototype, Binsec/Rel, for bug-finding and bounded-verification of constant-time and secret-erasure, and perform extensive experiments on a set of 338 cryptographic implementations, demonstrating the benefits of our approach. Using Binsec/Rel, we also automate two prior manual studies on preservation of constant-time and secret-erasure by compilers for a total of 4148 and 1156 binaries respectively. Interestingly, our analysis highlights incorrect usages of volatile data pointer for secret erasure and shows that scrubbing mechanisms based on volatile function pointers can introduce additional register spilling which might break secret-erasure. We also discovered that gcc -O0 and backend passes of clang introduce violations of constant-time in implementations that were previously deemed secure by a state-of-the-art constant-time verification tool operating at LLVM level, showing the importance of reasoning at binary-level.

This work has been published as a journal in ACM Transactions on Privacy and Security [2].

7.4.2 More security guarantees for the Jasmin compiler

Participants: Benjamin Grégoire, Jean-Christophe Léchenet.

We revisit the problem of erasing sensitive data from memory and registers during return from a cryptographic routine. While the problem and related attacker model is fairly easy to phrase, it turns out to be surprisingly hard to guarantee security in this model when implementing cryptography in common languages such as C/C++ or Rust. We revisit the issues surrounding zeroization and then present a principled solution in the sense that it guarantees that sensitive data is erased and it clearly defines when this happens. We implement our solution as extension to the formally verified Jasmin compiler and extend the correctness proof of the compiler to cover zeroization. We show that the approach seamlessly integrates with state-of-the-art protections against microarchitectural attacks by integrating zeroization into Libjade, a cryptographic library written in Jasmin with systematic protections against timing and Spectre-v1 attacks. Benchmarks show that in many cases the overhead of zeroization is barely measurable and that it stays below 2% except for highly optimized symmetric crypto routines on short inputs. This paper is accepted at CHES 2024.

7.5 Proof of post-quantum cryptography

Participants: Benjamin Grégoire.

The area of post-quantum cryptography (PQC) focuses on classical cryptosystems that are provably secure against quantum adversaries. PQC is based on computational problems that are conjectured to be hard for quantum computers, e.g., the learning with errors problem [13]. Simply relying on such assumptions, however, is insufficient to ensure security against quantum attackers; one must also verify that a security reduction holds in the quantum setting.

A natural question to ask is therefore whether we need a fundamentally different approach to the design of formal verification tools to capture these results, which seem tantalizingly close to the classical setting. For example, Unruh [14] suggests that the EasyCrypt is not sound for quantum adversaries. Concretely, [14] claim that the CHSH protocol, which is secure in the classic setting but not in the quantum setting, can be proved secure in EasyCrypt. While this point is moot, because the EasyCrypt logics was not designed (or claimed) to be sound for the quantum setting, it does raises an important question that we addressed in the paper [12]:

1. can we adapt the EasyCrypt program logic and libraries in a way that guarantees their soundness for PQC proofs?

2. is the resulting framework expressive and practical to use?

While the primary advantage of the logic proposed in [12] is its usability, a notable drawback is its limited expressivity. Numerous valid proof techniques for post-quantum cryptography cannot be articulated within this logic. Consequently, we have initiated a collaborative effort to define a new logic for proving post-quantum cryptography, aiming for expressiveness comparable to [14] while maintaining a high level of usability as seen in [12].

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 analogue, for concurrency and distribution, of data types for sequential computation. Originally conceived as a static analysis technique for a variant of the π-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 the *absence of communication errors* (no type mismatch in exchanged data) and *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 satisfy additional 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.

Choreographies are global specifications for multiparty communication protocols, very close in spirit to multiparty session types. A classical question for choreographies is whether they are realizable by means of a distributed implementation. We have been addressing this question using as an intermediate model *branching pomsets*, a recently proposed model for concurrent communicating processes. We have also investigated the relation between branching pomsets and several classes of event structures, some of which have already been used to model multiparty session types [1].

7.6.1 Branching pomsets for choreographies

Participants: Ilaria Castellani.

Choreographic languages describe possible sequences of interactions among a set of agents. Typical models are based on languages or automata over sending and receiving actions. Pomsets provide a more compact alternative by using a partial order to explicitly represent causality and concurrency between these actions. However, pomsets offer no representation of choices, thus a set of pomsets is required to represent branching behavior. For example, if an agent Alice can send one of two possible messages to Bob three times, one would need a set of $2 \times 2 \times 2$ distinct pomsets to represent all possible branches of Alice's behavior. This paper proposes an extension of pomsets, named *branching pomsets*, with a branching structure that can represent Alice's behavior using $2+2+2$ ordered actions. We compare the expressiveness of branching pomsets with that of several forms of event structures from the literature. We encode choreographies as branching pomsets and show that the pomset semantics of the encoded choreographies are bisimilar to their operational semantics. Furthermore, we define well-formedness conditions on branching pomsets, inspired by multiparty session types, and we prove that the well-formedness of a branching pomset is a sufficient condition for the realizability of the represented communication protocol. Finally, we present a prototype tool that implements our theory of branching pomsets, focusing on its applications to choreographies. These results are presented in [3].

7.6.2 Session type encodings

Participants: Ilaria Castellani.

To celebrate the 30th edition of EXPRESS (Expressiveness in Concurrency) and the 20th edition of SOS (Structural Operational Semantics), we presented a retrospective view on how session types can be expressed in a type theory for the standard π-calculus by means of a suitable encoding [4]. This encoding allows one to reuse results about the π-calculus in the context of session-based communications, thus deepening the understanding of sessions and reducing redundancies in their theoretical foundations. We have also reviewed the practical implications of these results (e.g., refined forms of deadlock analysis, type inference).

8 Partnerships and cooperations

8.1 International initiatives

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

HipHopSec

Title: Secure Reactive IoT Programming

Duration: 2020 -> 2024

Coordinator: Robby Findler (robby@eecs.northwestern.edu)

Partners: Northwestern University (Chicago) (USA)

Inria contact: Manuel Serrano

Summary: Nowadays most applications are distributed, that is, they run on several computers: a mobile device for the graphical user interface, a gateway for storing data in a local area; a remote server of a large cloud platform for resource demanding computing; an object connected to Internet in the IoT (Internet of Things); etc. For many different reasons, this makes programming much more difficult than it was when only a single computer was involved:

- Applications are composed of extensive lists of diverse components, each coming with their own specification and imposing their own constraints on application development.
- Due to the distributed nature of the applications, developers have to implement appropriate communication protocols, which is difficult to do correctly and securely.
- Communicating applications need to resort to parallelism to handle requests from their clients with acceptable latency. No matter whether it is multi-threading (as in Java) or asynchronous programming (as in JavaScript/Node.js), this style of programming is notoriously difficult and error-prone.

The former Indes now pursued in Splits team, Northwestern, and Collège de France teams are studying programming languages and have each created complementary solutions that address the aforementioned problems. Combined together, they could lead to a robust and secure execution environment for the web and IoT programming. Indes will bring its expertise in secure web programming, Collège de France its expertise in synchronous reactive programming, Northwestern its expertise in secure execution environments and run-time validation of security properties of program executions. Finally Northwestern will contribute with its expertise in medical descriptions, which will be the main application domain of the secure execution environment the participants aim to develop.

The main objective of the collaboration is the development of a robust and secure integrated programming environment for reactive applications suitable for web and IoT applications. The programming of medical prescriptions will be our favored application domain. We will base our work on three pillars: Hop.js, the contract system designed for the Racket language, and HipHop.js, a domain specific language for reactive programming within Hop.js.

- HipHop.js has currently minimal integration with Hop.js and a rudimentary programming environment. We will continue the development of HipHop.js with the goal of turning it into a usable and reliable platform.
- The formal semantics of HipHop.js is based on rewriting logics, automata theory and Boolean equations. Thus, HipHop.js programs can be verified using existing techniques based on the satisfiability of logic formulas. Such techniques have been widely used for synchronous reactive programs, but never before in the more dynamic world of web or medical applications.

- Supporting medical prescriptions as programs requires not only a language with special syntactic abstractions to match the notations of the medical domain, but also a fundamentally new way to think about prescription vs. computer programs. For example, medical personnel often modifies prescriptions in the middle of a treatment. In linguistic terms this requires that the programming language in use supports the ability to pause a program while it is running, modify its code, and restart it from the point of the pause but with the modified version of the code, this in a guaranteed consistent way. We hope to build such a programming language, with a semantics inspired by synchronous-reactive programming in the style of HipHop.js but tailored to the medical domain.
- Contracts state precise properties of the interfaces of components and validate them at run time. Over the last fifteen years, Racket developers, including those dealing with the language itself, have used contracts extensively to validate properties that range from simple type-like constraints to partial functional correctness and even security. Our goal is to design and implement a contract system for Hop/HipHop.js that is as expressive as that of Racket. Hop/HipHop.js is based on Javascript, a different linguistic setting than that of Racket; however, existing work on Javascript proxies and macros has resulted in encouraging preliminary results on contracts for higher-order functions and objects in Javascript. We aim at lifting and extending these results to Hop/HipHop.js. Given an expressive contract system for Hop/HipHop.js, we will investigate: (i) how to state and enforce security policies for Hop/HipHop.js applications with contracts; and (ii) how different compilation and implementation techniques can alleviate existing performance issues of applications, a current weakness that impedes the widespread adoption of contracts.
- Improving the quality of the code requires support from testing. S. You (working with C. Dimoulas and R. Findler) is working on improving automated testing techniques. So far he has discovered a new theoretical result showing how to use concolic testing for higher-order functions. This result may have applications for testing in JavaScript and we are hopeful that we can leverage it to Hop.js.

8.1.2 Visits of international scientists

Other international visits to the team

Robby Findler

Status Full professor

Institution of origin: Northwestern University

Country: USA

Dates: 2023/11/20-2023/12/01

Context of the visit: The HipHop programming language (funded by the HipHopSec inria associate team).

Mobility program/type of mobility: research stay

Marc Feeley

Status full professor

Institution of origin: University of Montréal

Country: Canada

Dates: 2023/06/15-/2023/07/15

Context of the visit: Compilation of dynamic languages

Mobility program/type of mobility: research stay

Tiago Oliveira

Status post-Doc

Institution of origin: Max Planck Institute for Security and Privacy (MPI-SP)

Country: Germany

Dates: 2023/09/25-2023/10/06

Context of the visit: Implementation of Dilithuim signature scheme in Jasmin

Mobility program/type of mobility: research stay

Li Zhou

Status researcher

Institution of origin: chinese academy of sciences

Country: China

Dates: 2023/10/05-2023/10/18

Context of the visit: A logic for post-quantum cryptography

Mobility program/type of mobility: research stay

Santiago Arranz Olmos

Status PhD

Institution of origin: MPI-SP

Country: Germany

Dates: 2023/11/20-2023/12/01

Context of the visit: Protection against Specter Attack in Jasmin

Mobility program/type of mobility: research stay

8.1.3 Visits to international teams

Research stays abroad Ilaria Castellani visited the team of Nobuko Yoshida at the University of Oxford for two weeks and Emilio Tuosto at the Gran Sasso Science Institute for one week.

8.2 National initiatives

8.2.1 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⊕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.

8.2.2 ANR CISC

Participants: Ilaria Castellani, Tamara Rezk, Manuel Serrano.

The CISC project (Certified IoT Secure Compilation) is funded by the ANR for 42 months, ending in September 2023. The goal of the CISC project is to provide strong security and privacy guarantees for IoT applications by means of a language to orchestrate IoT applications from the microcontroller to the cloud. Tamara Rezk coordinates this project, and Manuel Serrano and Ilaria Castellani participate in the project. The partners of this project are the INRIA teams Celtique and SPLiTS, and Collège de France.

8.2.3 PEPR

Participant: Benjamin Grégoire, Jean-Christophe Léchenet.

SVP PEPR Cybersecurity. We participate in a project concerned with the verification of security protocols. Partners in this project are CNRS IRISA Rennes (coordinator Stéphanie Delaune), INRIA, University of Paris-Saclay, University of Lorraine, University of Côte d'Azur, ENS Rennes. The funds allocated to our team in this collaboration are 333 KEuros. The corresponding researcher for this contract is Benjamin Grégoire.

9 Dissemination

9.1 Promoting scientific activities

9.1.1 Scientific events: organisation

We organized the Inaugural SPliTS workshop on September 29th, 2023, in Belles Rives, Antibes.

Chair of conference program committees

- Manuel Serrano was the Program Chair of the PROGRAMMING'23 Conference.
- Tamara Rezk was the Program Chair of CSF'23: Computer Security Foundations.
- Ilaria Castellani was Program co-Chair of PLACES'23: 14th Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software.

Member of the conference program committees

- Manuel Serrano participated in the program committees of:
  - DLS'23: Dynamic Language Symposium;
  - ECOOP'23: European Conference on Object-Oriented Programming;
- Tamara Rezk participated in the program committee of the ACM Conference on Computer and Communications Security (CCS'23).
- Benjamin Grégoire participated in the program committee of the IEEE Computer Security Foundations Symposium (CSF'23).
- Ilaria Castellani participated in the program committee of COORDINATION 2023: 25th International Conference on Coordination Models and Languages.
- Manuel Serrano is member of the PROGRAMMING Steering Committee

- Tamara Rezk is member of the Programming Language Mentoring Workshop (PLMW) Steering Committee
- Tamara Rezk is member of the CSF Steering Committee
- Tamara Rezk is member of the Foundations of Computer Security Workshop (FCS) Steering Committee
- Tamara Rezk is member of the Principles of Secure Compilation (PriSC) Steering Committee

9.1.2 Invited talks

- Tamara Rezk was invited as:
  - Speaker at the NICT/Inria/IMT Workshop, on March 1st, 2023.
  - Keynote speaker at the Annual Meeting of the WG "Formal Methods for Security" (GT MFS), on March 28th, 2023.
  - Keynote speaker at WASP4ALL 2023: Building the future Cyber Security landscape, on June 1st, 2023.
  - Keynote speaker at the Navigating the Cybersecurity Landscape, Chalmers, on October 11th, 2023. In the same event, she was invited to participate in a pannel on Cybersecurity.
  - Keynote speaker at the Programming Languages and Analysis for Security Workshop at CCS, Copenhagen, on November 26th, 2023.

9.1.3 Scientific expertise

Tamara Rezk was an expert in the selection committee for the Estonian Research Council 2023.

9.1.4 Research administration

- Tamara Rezk is part of the *bureau du CEP* at INRIA Sophia Antipolis. In 2023, she also coordinated the working group for the creation of the EP OLAS.
- Manuel Serrano was vice-head of the Inria Evaluation Committee until end of August 2023. As such he co-organized all the grants, promotion juries and the juries of the national recruiting campaigns. He also co-organized all the team evaluation seminars.
- Benjamin Grégoire is member of the CSD.
- Ilaria Castellani is a member of the INRIA Equal Opportunity Committee, and of the organising committees of the Sophia Antipolis Colloquium and of the Forum Numerica seminar series.

9.2 Teaching - Supervision - Juries

- Manuel Serrano, Efficient AOT JavaScript Compilation, PLISS'23 summer school (Bertinoro, Italy).
- Tamara Rezk gave a course on Web Security at University of Nice Sophia Antipolis.
- Tamara Rezk taught a course on Code Safety at the Master Cyber Ecole Polytechnique.

9.2.1 Supervision

- PhD defended: Jayanth Krishnamurthy, Debugging Techniques for the HipHop.js Reactive Programming Language, supervisor: Manuel Serrano.
- Phd defended: Benjamin Grégoire supervised the thesis of Swarn Priya (Université Côte d'Azur) until November 21, codirector Yves Bertot.

- PhD in progress: Aurore Poirier, Optimizing Compilation from Higher-Order Programming to Computer Architecture, supervisors: Erven Rohou & Manuel Serrano.
- PhD in progress: Olivier Melançon, Basic Block Versioning for Python, supervisors: Marc Feeley & Manuel Serrano.
- PhD in progress: Davide Davoli, Kernel Security in the Spectre Era, co-supervision Martin Avanzini and Tamara Rezk.
- PhD in progress: Ignacio Tiraboschi, Program Analyses for Security, co-supervision Xavier Rival and Tamara Rezk.
- PhD in progress: Guillaume Combette, Binary Analyses for Security, co-supervision Sébastien Bardin and Tamara Rezk.

9.2.2 Juries

- Benjamin Grégoire was a member of the jury Abdul Rahman Taleb (Sorbonne Université) in November.
- Tamara Rezk was opponent for the PhD defense of Andreas Lindner, KTH Royal Institute of Technology.
- Tamara Rezk was rapporteuse and a jury member for the HDR thesis of Clémentine Maurice, University of Lille.
- Tamara Rezk was rapporteuse and a jury member for the PhD thesis of Feras Al Kassar, Eurecom.
- Tamara Rezk was a member of the CSD committee for Jonathan Brossard, CNAM.
- Ilaria Castellani was a jury member for the PhD thesis of Loïc Germerie Guizouarn, Université Côte d'Azur.

10 Scientific production

10.1 Publications of the year

International journals

[1] I. Castellani, M. Dezani-Ciancaglini and P. Giannini. 'Event structure semantics for multiparty sessions'. In: *Journal of Logical and Algebraic Methods in Programming* 131 (Feb. 2023). DOI: 10.1016/j.jlamp.2022.100844. URL: https://inria.hal.science/hal-03940191.

[2] L.-A. Daniel, S. Bardin and T. Rezk. 'Binsec/Rel: Symbolic Binary Analyzer for Security with Applications to Constant-Time and Secret-Erasure'. In: *ACM Transactions on Privacy and Security* (2023). URL: https://inria.hal.science/hal-03833598.

[3] L. Edixhoven, S.-S. Jongmans, J. Proença and I. Castellani. 'Branching pomsets: Design, expressiveness and applications to choreographies'. In: *Journal of Logical and Algebraic Methods in Programming* 136 (20th Sept. 2023). DOI: 10.1016/j.jlamp.2023.100919. URL: https://inria.hal.science/hal-04360686.

Invited conferences

[4] I. Castellani, O. Dardha, L. Padovani and D. Sangiorgi. 'EXPRESSing Session Types'. In: *Electronic Proceedings in Theoretical Computer Science*. EXPRESS / SOS 2023 - Combined 30th International Workshop on Expressiveness in Concurrency and 20th Workshop on Structural Operational Semantics. Vol. EPTCS-387. Proceedings Combined 30th International Workshop on Expressiveness in Concurrency and 20th Workshop on Structural Operational Semantics. Anvers (Antwerpen), Belgium, 14th Sept. 2023, pp. 8–25. DOI: 10.4204/EPTCS.387.2. URL: https://inria.hal.science/hal-04349502.

International peer-reviewed conferences

[5] J. Bacelar Almeida, M. Barbosa, G. Barthe, B. Grégoire, V. Laporte, J.-C. Léchenet, T. Oliveira, H. Pacheco, M. Quaresma, P. Schwabe, A. Séré and P.-Y. Strub. 'Formally verifying Kyber: Episode IV: Implementation correctness'. In: *ACR Transactions on Cryptographic Hardware and Embedded Systems*. CHES 2023 - Conference on Cryptographic Hardware and Embedded Systems. Vol. 2023. 3. Praha, Czech Republic, 9th June 2023, pp. 164–193. DOI: 10.46586/tches.v2023.i3.164-193. URL: https://inria.hal.science/hal-04218417.

[6] M. Barbosa, G. Barthe, C. Doczkal, J. Don, S. Fehr, B. Grégoire, Y.-H. Huang, A. Hülsing, Y. Lee and X. Wu. 'Fixing and Mechanizing the Security Proof of Fiat-Shamir with Aborts and Dilithium'. In: *Lecture Notes in Computer Science*. CRYPTO 2023 - 43rd International Cryptology Conference. Vol. LNCS-14085. Advances in Cryptology – CRYPTO 2023 : 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part V. Santa Barbara, United States, 20th Aug. 2023, pp. 358–389. DOI: 10.1007/978-3-031-38554-4_12. URL: https://hal.science/hal-04315311.

[7] 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.

[8] O. Melançon, M. Feeley and M. Serrano. 'An Executable Semantics for Faster Development of Optimizing Python Compilers'. In: SLE '23: 16th ACM SIGPLAN International Conference on Software Language Engineering. Cascais Portugal, Portugal: ACM, 22nd Oct. 2023, pp. 15–28. DOI: 10.1145/3623476.3623529. URL: https://inria.hal.science/hal-04391611.

[9] I. Tiraboschi, T. Rezk and X. Rival. 'Sound Symbolic Execution via Abstract Interpretation and its Application to Security'. In: *Lecture Notes in Computer Science*. VMCAI 2023 - 24th International Conference on Verification, Model Checking, and Abstract Interpretation. Vol. 13881. Verification, Model Checking, and Abstract Interpretation 24th International Conference, VMCAI 2023, Boston, MA, USA, January 16–17, 2023, Proceeding. Boston, MA, United States: Springer Nature Switzerland, 17th Jan. 2023, pp. 267–295. DOI: 10.1007/978-3-031-24950-1_13. URL: https://hal.science/hal-03942146.

Reports & preprints

[10] B. Blanchet, P. Boutry, C. Doczkal, B. Grégoire and P.-Y. Strub. *CV2EC: Getting the Best of Both Worlds*. 4th Dec. 2023. URL: https://inria.hal.science/hal-04321656.

[11] A. Canteaut, M. Serrano, C. Grandmont, G. Pallez, V. Perrier, X. Rival and E. Thomé. *Bilan de la mandature 2019-2023 de la Commission d'Évaluation Inria*. Inria, 31st Aug. 2023. URL: https://inria.hal.science/hal-04193082.

10.2 Cited publications

[12] M. Barbosa, G. Barthe, X. Fan, B. Grégoire, S. Hung, J. Katz, P. Strub, X. Wu and L. Zhou. 'EasyPQC: Verifying Post-Quantum Cryptography'. In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Y. Kim, J. Kim, G. Vigna and E. Shi. ACM, 2021, pp. 2564–2586. DOI: 10.1145/3460120.3484567. URL: https://doi.org/10.1145/3460120.3484567.

[13] O. Regev. 'On lattices, learning with errors, random linear codes, and cryptography'. In: *37th*. Ed. by H. N. Gabow and R. Fagin. Baltimore, MA, USA, May 2005, pp. 84–93. DOI: 10.1145/1060590.1060603.

[14] D. Unruh. 'Quantum relational Hoare logic'. In: *Proc. ACM Program. Lang.* 3.POPL (2019), 33:1–33:31. DOI: 10.1145/3290346. URL: https://doi.org/10.1145/3290346.