

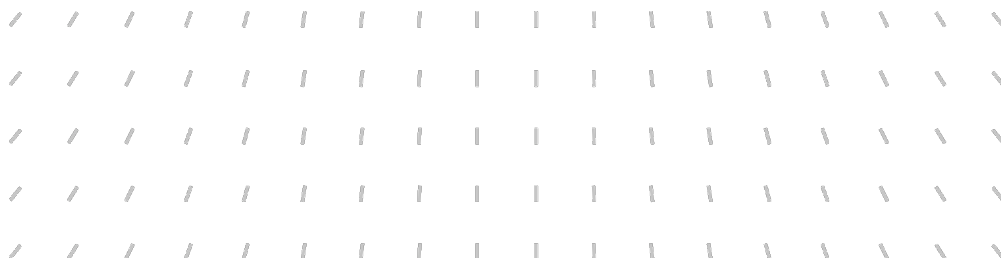
2025 Activity Report

RESEARCH CENTRE: Inria Paris Centre


Team

PROSECCO

Programming securely with cryptography



Team PROSECCO

Creation of the Team: 2025 January 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. – Architectures
 - A1.1.8. – Security of architectures
- A1.2. – Networks
 - A1.2.8. – Network security
- A1.3. – Distributed Systems
- A2. – Software sciences
 - A2.1. – Programming Languages
 - A2.1.1. – Semantics of programming languages
 - A2.1.4. – Functional programming
 - A2.1.7. – Distributed programming
 - A2.1.11. – Proof languages
 - A2.2. – Compilation
 - A2.2.1. – Static analysis
 - A2.2.5. – Run-time systems
 - A2.5. – Software engineering
- A4. – Security and privacy
 - A4.3. – Cryptography
 - A4.3.3. – Cryptographic protocols
 - A4.5. – Formal method for verification, reliability, certification
 - A4.5.2. – Model-checking
 - A4.5.3. – Program proof
 - A4.6. – Authentication
 - A4.8. – Privacy-enhancing technologies

Other research topics and application domains

- B6. – IT and telecom
 - B6.1. – Software industry
 - B6.1.1. – Software engineering
 - B6.3. – Network functions
 - B6.3.1. – Web
 - B6.3.2. – Network protocols
 - B6.4. – Internet of things
- B9. – Society and Knowledge
 - B9.6.2. – Juridical science
 - B9.10. – Privacy

Contents

Team PROSECCO	1
1 Team members, visitors, external collaborators	5
2 Overall objectives	6
2.1 Programming securely with cryptography	6
3 Research program	7
3.1 Symbolic verification of cryptographic applications	7
3.2 Computational verification of cryptographic applications	8
3.3 F*: A Higher-Order Effectful Language for Program Verification	9
3.4 Analysis of Rust Programs	9
3.5 Provably secure web applications	9
3.6 Design and Verification of next-generation protocols: identity, blockchains, and messaging	10
3.7 Formalizing Law	10
4 Application domains	10
4.1 High-Assurance Cryptographic Libraries	10
4.2 Design and Analysis of Protocol Standards	11
4.3 Web application security	11
4.4 Formalizing Law	11
5 Social and environmental responsibility	11
5.1 Footprint of research activities	11
6 Highlights of the year	11
6.1 Awards	11
7 Latest software developments, platforms, open data	12
7.1 Latest software developments	12
7.1.1 F*	12
7.1.2 Steel	12
7.1.3 StarMalloc	12
7.1.4 HACL*	13
7.1.5 DY*	13
7.1.6 Aeneas	14
7.1.7 Charon	14
7.1.8 Catala	15
7.1.9 ProVerif	15
7.1.10 CryptoVerif	15
7.1.11 Squirrel	16
8 New results	17
8.1 Verification of security protocols in the computational model	17
8.2 Verification of cryptographic protocol implementations in the symbolic model: the DY* framework	18
8.3 High-Assurance High-Performance Cryptography	18
8.4 Formal Verification of Rust Programs	19
8.5 Formalizing and Implementing Law	19
9 Bilateral contracts and grants with industry	19

10 Partnerships and cooperations	20
10.1 National initiatives	20
10.1.1 PEPR	20
10.1.2 ANR	21
11 Dissemination	21
11.1 Promoting scientific activities	21
11.1.1 Scientific events: selection	21
11.1.2 Journal	22
11.1.3 Invited talks	22
11.1.4 Research administration	22
11.2 Teaching - Supervision - Juries - Educational and pedagogical outreach	22
11.2.1 Teaching	22
11.2.2 Supervision	22
11.2.3 Juries	23
12 Scientific production	23
12.1 Major publications	23
12.2 Publications of the year	24
12.3 Cited publications	24

1 Team members, visitors, external collaborators

Research Scientists

- Bruno Blanchet [Team leader, INRIA, Senior Researcher, HDR]
- Aymeric Fromherz [INRIA, ISFP]
- Adrien Koutsos [INRIA, Researcher]

PhD Students

- Marco Bertoni [INRIA, from Dec 2025]
- Alain Delaet-Tixeuil [INRIA, until Sep 2025]
- Antonin Reitz [INRIA]
- Justine Sauvage [INRIA]
- Theo Vignon [ENS PARIS-SACLAY]
- Theophile Wallez [INRIA, until Jun 2025]

Technical Staff

- Guillaume Boisseau [INRIA, Engineer]
- Vincent Botbol [INRIA, Engineer]
- Yoann Prak [INRIA, Engineer]

Interns and Apprentices

- Samuel Avril [ENS DE LYON, Intern, from May 2025 until Jul 2025]
- Florian Duzes [INRIA, Intern, from Apr 2025 until Aug 2025]
- Vivien Gachet [ENS DE LYON, Intern, from Mar 2025 until Aug 2025]
- Fernando Leal Sanchez [INRIA, Intern, from Feb 2025 until Jul 2025]
- Emmanuel Mera [ENS PARIS-SACLAY, Intern, from Mar 2025 until Aug 2025]

Administrative Assistants

- Christelle Guiziou [INRIA]
- Abigail Palma [INRIA]

External Collaborators

- Caroline Fontaine [CNRS]
- Lucas Franceschino [CRYPEN, until Mar 2025]
- Estelle Hary [POLE EMPLOI, until Sep 2025]

2 Overall objectives

2.1 Programming securely with cryptography

In recent years, an increasing amount of sensitive data is being generated, manipulated, and accessed online, from bank accounts to health records. Both national security and individual privacy have come to rely on the security of web-based software applications. But even a single design flaw or implementation bug in an application may be exploited by a malicious criminal to steal, modify, or forge the private records of innocent users. Such *attacks* are becoming increasingly common and now affect millions of users every year.

The risks of deploying insecure software are too great to tolerate anything less than mathematical proof, but applications have become too large for security experts to examine by hand, and automated verification tools do not scale. Today, there is not a single widely-used web application for which we can give a proof of security, even against a small class of attacks. In fact, design and implementation flaws are still found in widely-distributed and thoroughly-vetted security libraries designed and implemented by experts.

Software security is in crisis. A focused research effort is needed if security programming and analysis techniques are to keep up with the rapid development and deployment of security-critical distributed applications based on new cryptographic protocols and secure hardware devices. The goal of our team PROSECCO is to draw upon our expertise in cryptographic protocols and program verification to make decisive contributions in this direction.

Our vision is that, over its lifetime, PROSECCO will contribute to making the use of formal techniques when programming with cryptography as natural as the use of a software debugger. To this end, our long-term goals are to design and implement programming language abstractions, cryptographic models, verification tools, and verified security libraries that developers can use to deploy provably secure distributed applications. Our target applications include cryptographic libraries, network protocol implementations, web applications, and cloud-based web services. In particular, we aim to verify full software applications, including both the cryptographic core and the high-level application code. Furthermore, we aim to verify implementations, not just models. Finally, we aim to account for computational cryptography, not just its symbolic abstraction.

We identify four key focus areas for our research in the short- to medium term.

New programming languages for verified software Building realistic verified applications requires new programming languages that enable the systematic development of efficient software hand-in-hand with their proofs of correctness. We design and implement the programming language F*, in collaboration with Microsoft Research. F* (pronounced F star) is a general-purpose functional programming language with a state-of-the-art type-and-effect system aimed at program verification. Its type system includes polymorphism, dependent types, monadic effects, refinement types, and a weakest precondition calculus. Together, these features allow expressing precise and compact specifications for programs, including functional correctness and security properties. The F* type-checker aims to prove that programs meet their specifications using a combination of SMT solving and interactive proofs. Programs written in F* can be translated to efficient OCaml, F#, or C for execution. The main ongoing use cases of F* in our group are HACLS*, a verified cryptographic library, and DY*, a framework for verifying protocol implementations. Nevertheless, we also consider non-cryptographic security software, for which we also use F* and its extensions, for instance security-enhanced memory allocators, who are often the last line of defense against memory vulnerabilities in critical C and C++ software [26].

We also design two frameworks for the analysis of Rust programs (hacspecc and Aeneas), by translation to various theorem provers including F*.

Recently, we extended our work on programming languages to a domain-specific language for implementing law, for instance tax computation, which is also critical as it impacts every citizen. We indeed noticed that much of the infrastructure and methodologies we developed for cryptographic security software can be transferred to other domains in need of high-assurance software. The combination of software engineering and formal methods that we employ at the Prosecco team may thus have a more general field of application beyond cryptographic software.

Symbolic verification of cryptographic applications We aim to develop our own security verification tools for models and implementations of cryptographic protocols and security APIs using symbolic cryptography. Our starting point is the tools we have previously developed: the specialized cryptographic prover ProVerif and

the F* verification system via the DY* framework. These tools are already used to verify industrial-strength cryptographic protocol implementations and commercial cryptographic hardware. We plan to extend and combine these approaches to capture more sophisticated attacks on applications consisting of protocols, software, and hardware, as well as to prove symbolic security properties for such composite systems.

Computational verification of cryptographic applications We aim to develop our own cryptographic application verification tools that use the computational model of cryptography. The tools include the computational provers CryptoVerif and Squirrel, and the F* verification system. Working together, we plan to extend these tools to analyze, for the first time, cryptographic protocols, security APIs, and their implementations under fully precise cryptographic assumptions. We also plan to pursue links between tools, in order to use each tool where it is the strongest.

Building provably secure web applications We aim to develop analysis tools and verified libraries to help programmers build provably secure web applications. The tools will include static and dynamic verification tools for client- and server-side JavaScript web applications, their verified deployment within HTML5 websites and browser extensions, as well as type-preserving compilers from high-level applications written in F* to JavaScript. In addition, we plan to model new security APIs in browsers and smartphones and develop the first formal semantics for various HTML5 web standards. We plan to combine these tools and models to analyze the security of multi-party web applications, consisting of clients on browsers and smartphones, and servers in the cloud.

3 Research program

3.1 Symbolic verification of cryptographic applications

Despite decades of experience, designing and implementing cryptographic applications remains dangerously error-prone, even for experts. This is partly because cryptographic security is an inherently hard problem, and partly because automated verification tools require carefully-crafted inputs and are not widely applicable. To take just the example of TLS, a widely-deployed and well-studied cryptographic protocol designed, implemented, and verified by security experts, the lack of a formal proof about all its details has regularly led to the discovery of major attacks (including several in PROSECCO) on both the protocol and its implementations, after many years of unsuspecting use.

As a result, the automated verification for cryptographic applications is an active area of research, with a wide variety of tools being employed for verifying different kinds of applications.

In previous work, we have developed the following approaches:

- ProVerif: a symbolic prover for cryptographic protocol models
- F*: a new language that enables the verification of cryptographic applications

Verifying cryptographic protocols with ProVerif Given a model of a cryptographic protocol, the problem is to verify that an active attacker, possibly with access to some cryptographic keys but unable to guess other secrets, cannot thwart security goals such as authentication and secrecy [46]; it has motivated a serious research effort on the formal analysis of cryptographic protocols, starting with [39] and eventually leading to effective verification tools, such as our tool ProVerif.

To use ProVerif, one encodes a protocol model in a formal language, called the applied pi-calculus, and ProVerif abstracts it to a set of generalized Horn clauses. This abstraction is a small approximation: it just ignores the number of repetitions of each action, so ProVerif is still very precise, more precise than, say, tree automata-based techniques. The price to pay for this precision is that ProVerif does not always terminate; however, it terminates in most cases in practice, and it always terminates on the interesting class of *tagged protocols* [35]. ProVerif can handle a wide variety of cryptographic primitives, defined by rewrite rules or by some equations, and prove a wide variety of security properties: secrecy [32, 20], correspondences (including authentication) [33], and observational equivalences [34]. Observational equivalence means that an adversary cannot distinguish two processes (protocols); equivalences can be used to formalize a wide range of properties, but they are particularly difficult to prove. Even if the class of equivalences that ProVerif

can prove is limited to equivalences between processes that differ only by the terms they contain, these equivalences are useful in practice and ProVerif has long been the only tool that proves equivalences for an unbounded number of sessions. (Maude-NPA in 2014 and Tamarin in 2015 adopted ProVerif’s approach to proving equivalences.)

Using ProVerif, it is now possible to verify large parts of industrial-strength protocols, such as TLS [3], Signal [44], JFK [21], and Web Services Security [31], against powerful adversaries that can run an unlimited number of protocol sessions, for strong security properties expressed as correspondence queries or equivalence assertions. ProVerif is used by many teams at the international level, and has been used in more than 140 research papers (references).

Verifying cryptographic applications using F* Verifying the implementation of a protocol has traditionally been considered much harder than verifying its model. This is mainly because implementations have to consider real-world details of the protocol, such as message formats [50], that models typically ignore. So even if a protocol has been proved secure in theory, its implementation may be buggy and insecure. However, with recent advances in both program verification and symbolic protocol verification tools, it has become possible to verify fully functional protocol implementations in the symbolic model. One approach is to extract a symbolic protocol model from an implementation and then verify the model, say, using ProVerif. This approach has been quite successful, yielding a verified implementation of TLS in F# [30]. However, the generated models are typically quite large and whole-program symbolic verification does not scale very well.

An alternate approach is to develop a verification method directly for implementation code, using well-known program verification techniques. We design and implement the programming language F* [9], [22, 45], in collaboration with Microsoft Research. F* is an ML-like functional programming language aimed at program verification. Its type system includes polymorphism, dependent types, monadic effects, refinement types, and a weakest precondition calculus. Together, these features allow expressing precise and compact specifications for programs, including functional correctness and security properties. The F* type-checker aims to prove that programs meet their specifications using a combination of SMT solving and interactive proofs. Programs written in F* can be translated to efficient OCaml, F#, or C for execution [49]. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest [28] (a larger collaboration with Microsoft Research). This includes a verified implementation of TLS 1.2 and 1.3 [29] and of the underlying cryptographic primitives [10]. More recently, we have built a new symbolic protocol verification framework in F* called DY* [27] and used it to verify real-world cryptographic protocols like Signal, ACME and Noise.

3.2 Computational verification of cryptographic applications

Proofs done by cryptographers in the computational model are mostly manual. Our goal is to provide computer support to build or verify these proofs. In order to reach this goal, we have designed the automatic tool CryptoVerif, which generates proofs by sequences of games. We already applied it to important protocols such as TLS [3] and Signal [44] but more work is still needed in order to develop this approach, so that it is easier to apply to more protocols.

Another tool we develop, called the Squirrel Prover, uses a symbolic approach called the computationally complete symbolic adversary (CCSA) [24] to verify cryptographic protocols in the computational model. Squirrel is an interactive theorem prover, hence provides less automation than CryptoVerif, but allows the user to guide the proof more easily when complex arguments are needed; and it is better-suited for some protocols, notably for stateful protocols.

A third approach is to directly verify executable cryptographic code by typing. A recent work [40] shows how to use refinement typechecking to prove computational security for protocol implementations. In this method, henceforth referred to as computational F*, typechecking is used as the main step to justify a classic game-hopping proof of computational security. The correctness of this method is based on a probabilistic semantics of F# programs and crucially relies on uses of type abstraction and parametricity to establish strong security properties, such as indistinguishability.

In principle, the three approaches—game-based proofs in CryptoVerif, interactive proofs in Squirrel, and typechecking proofs in F*—are complementary. Understanding how to combine these approaches remains an open and active topic of research. For example, CryptoVerif can generate OCaml implementations from

CryptoVerif specifications that have been proved secure [36]. We are currently working on this approach to generate implementations in F*.

3.3 F*: A Higher-Order Effectful Language for Program Verification

F* [9], [22] is a verification system for effectful programs developed collaboratively by Inria and Microsoft Research. It puts together the automation of an SMT-backed deductive verification tool with the expressive power of a proof assistant based on dependent types. After verification, F* programs can be extracted to efficient OCaml, F#, or C code [49]. This enables verifying the functional correctness and security of realistic applications. F*'s type system includes dependent types, monadic effects, refinement types, and a weakest precondition calculus. Together, these features allow expressing precise and compact specifications for programs, including functional correctness and security properties. The F* type-checker aims to prove that programs meet their specifications using a combination of SMT solving and interactive proofs. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest. This includes verified implementations of TLS 1.2 and 1.3 [29] and of the underlying cryptographic primitives [10], [48, 47].

3.4 Analysis of Rust Programs

Rust is a modern programming language that provides both performance and memory safety and is well suited for critical system programming. We develop two frameworks for analyzing Rust programs.

We develop hacspec, a purely functional domain-specific language embedded in Rust for writing succinct executable specifications, in particular for cryptographic algorithms, which can be translated to proof back-ends like F* and Coq.

We also develop Aeneas [41], which leverages Rust's rich region-based type system to eliminate memory reasoning for a large class of Rust programs, as long as they do not rely on interior mutability or unsafe code. Doing so, Aeneas relieves the proof engineer of the burden of memory-based reasoning, allowing them to instead focus on functional properties of their code. Aeneas proposes a new Low-Level Borrow Calculus (LLBC) that captures a large subset of Rust programs, and a translation from LLBC to a pure lambda-calculus, which enables the verification of Rust programs through different theorem provers, such as Lean, Coq, or F*.

3.5 Provably secure web applications

Web applications are fast becoming the dominant programming platform for new software, probably because they offer a quick and easy way for developers to deploy and sell their *apps* to a large number of customers. Third-party web-based apps for Facebook, Apple, and Google, already number in the hundreds of thousands and are likely to grow in number. Many of these applications store and manage private user data, such as health information, credit card data, and GPS locations. To protect this data, applications tend to use an ad hoc combination of cryptographic primitives and protocols. Since designing cryptographic applications is easy to get wrong even for experts, we believe this is an opportune moment to develop security libraries and verification techniques to help web application programmers.

As a typical example, consider commercial password managers, such as LastPass, RoboForm, and 1Password. They are implemented as browser-based web applications that, for a monthly fee, offer to store a user's passwords securely on the web and synchronize them across all of the user's computers and smartphones. The passwords are encrypted using a master password (known only to the user) and stored in the cloud. Hence, no-one except the user should ever be able to read her passwords. When the user visits a web page that has a login form, the password manager asks the user to decrypt her password for this website and automatically fills in the login form. Hence, the user no longer has to remember passwords (except her master password) and all her passwords are available on every computer she uses.

Password managers are available as browser extensions for mainstream browsers such as Firefox, Chrome, and Internet Explorer, and as downloadable apps for Android and Apple phones. So, seen as a distributed application, each password manager application consists of a web service (written in PHP or Java), some number of browser extensions (written in JavaScript), and some smartphone apps (written in Java or Objective C). Each of these components uses a different cryptographic library to encrypt and decrypt password data. How do we verify the correctness of all these components?

We propose three approaches. For client-side web applications and browser extensions written in JavaScript, we propose to build a static and dynamic program analysis framework to verify security invariants. To this end, we have developed two security-oriented type systems for JavaScript, Defensive JavaScript [38] and TS* [51], and used them to guarantee security properties for a number of JavaScript applications. For Android smartphone apps and web services written in Java, we propose to develop annotated JML cryptography libraries that can be used with static analysis tools like ESC/Java to verify the security of application code. For clients and web services written in F# for the .NET platform, we propose to use F* to verify their correctness. We also propose to translate verified F* web applications to JavaScript via a verified compiler that preserves the semantics of F* programs in JavaScript.

3.6 Design and Verification of next-generation protocols: identity, blockchains, and messaging

Building on our work on verifying and re-designing pre-existing protocols like TLS and Web Security in general, with the resources provided by the NEXTLEAP project, we are working on both designing and verifying new protocols in rapidly emerging areas like identity, blockchains, and secure messaging. These are all areas where existing protocols, such as the heavily used OAuth protocol, are in need of considerable re-design in order to maintain privacy and security properties. Other emerging areas, such as blockchains and secure messaging, can have modifications to existing pre-standard proposals or even a complete 'clean slate' design. As shown by Prosecco's work, newer standards, such as IETF OAuth, W3C Web Crypto, and W3C Web Authentication API, can have vulnerabilities fixed before standardization is complete and heavily deployed. We hope that the tools used by Prosecco can shape the design of new protocols even before they are shipped to standards bodies. We are currently contributing to the design and analysis of new extensions to the TLS protocol, such as Encrypted Client Hello, new secure messaging protocol such as IETF Messaging Layer Security (MLS), and to IoT protocols like the IETF Lightweight Authenticated Key Exchange (LAKE).

3.7 Formalizing Law

In France, income tax is computed from taxpayers' individual returns, using an algorithm that is authored, designed and maintained by the French Public Finances Directorate (DGFIP). Owing to the shortcomings of its custom programming language and the technical limitations of the compiler, the algorithm is proving harder and harder to maintain, relying on ad-hoc behaviors and workarounds to implement the most recent changes in tax law. As an improvement to this infrastructure, we developed Mlang, an open-source compiler toolchain that has been thoroughly validated against the private DGFIP test suite. The DGFIP is now officially transitioning to Mlang for their production system. This line of work has yielded papers at CC 2020 and JFLA, as well as a [successful industrial technology transfer](#) from Inria to DGFIP.

Building on the work on Mlang, Prosecco has seen the development of a new domain-specific language, Catala, targeted specifically for legal expert systems. This new domain-specific language has been built in close collaboration with lawyers, and advertised to that community with a number of legal-oriented papers [43]. On the formal methods side, the simple and clean design of the Catala semantics allows for extension into a proper proof platform for the law [37]. Catala has been tested on the real-world French housing benefits [17, 18] and is currently [experimented for use at DGFIP](#).

4 Application domains

4.1 High-Assurance Cryptographic Libraries

Cryptographic libraries implement algorithms for symmetric and asymmetric encryption, digital signatures, message authentication, hashing, and key exchange. Popular libraries like OpenSSL, NSS, and BoringSSL are widely used in web browsers, operating system, and cloud services. We aim to apply our tools and verification techniques to build high-assurance high-performance cryptographic libraries that can be deployed in mainstream software applications. Our flagship project is HACL*, a verified cryptographic library that is written in the F* programming language.

4.2 Design and Analysis of Protocol Standards

Cryptographic protocol standards such as TLS, SSH, IPsec, and Kerberos are the trusted base on which the security of modern distributed systems is built. Our work enables the analysis and verification of such protocols, both in their design and implementation. We participate in standards organizations like the IETF and collaborate with industry groups to help them design and deploy secure protocols. For example, we built and verified models and reference implementations for the well-known TLS 1.3 protocol, using our tools ProVerif and CryptoVerif, before it was standardized at the IETF and contributed to the protocol's final design.

4.3 Web application security

Web applications use a variety of cryptographic techniques to securely store and exchange sensitive data for their users. For example, a website may serve pages over HTTPS, authenticate users with a single sign-on protocol such as OAuth, encrypt user files on the server-side using XML encryption, and deploy client-side cryptographic mechanisms using a JavaScript cryptographic library. The security of these applications depends on the public key infrastructure (X.509 certificates), web browsers' implementation of HTTPS and the same origin policy (SOP), the semantics of JavaScript, HTML5, and their various associated security standards, as well as the correctness of the specific web application code of interest. We build analysis tools to find bugs in all these artifacts and verification tools that can analyze commercial web applications and evaluate their security against sophisticated web-based attacks.

4.4 Formalizing Law

Taxes and social benefits are cornerstones of public policies in developed countries. Concretely, in most places, citizens would fill a form describing their income and family situation, send it to the tax or benefits agency, and then receive or pay the amount the agency has determined based on the information on the form. Determining this amount involves a computation specified by the law, that describes the tax brackets, benefits ceilings, etc. Since this computation is done regularly for a large chunk of the population, it has been computerized for a long time. However, as these aging government IT systems are becoming harder and harder to maintain, the challenge of accurately computing taxes and benefits in the context of increased public algorithmic scrutiny remains. Reusing some of our high-assurance software methodology from the domain of cryptography, we have built domain-specific languages and associated tooling to help pairs of programmers and lawyers produce and maintain tax and social benefits IT systems.

5 Social and environmental responsibility

5.1 Footprint of research activities

Our team's work focuses on the design, analysis, and implementation of cryptographic protocols. As such, we are dedicated to improving the security and privacy of all Web users. The output of our research is used, for example, to protect HTTPS connections used daily by millions of Mozilla Firefox users. On the whole, we strive to perform ethical research that improves the digital lives of citizens everywhere.

Our research does not by itself have any environmental impact, but our team does travel to conferences, and we regularly host international visitors, which incurs multiple international flights each year.

6 Highlights of the year

6.1 Awards

Pierre Goutagny, Aymeric Fromherz, and Raphaël Monat received a distinguished artefact award for CUTEcat at the European Symposium on Programming (ESOP'25) [12].

Son Ho received an honorable mention for the **thesis prize of the GDR-SI** (computer security research groupment of the CNRS), for his PhD on the verification of Rust programs, defended on December 9, 2024 [19].

7 Latest software developments, platforms, open data

7.1 Latest software developments

7.1.1 F*

Name: FStar

Keywords: Programming language, Software Verification

Functional Description: F* is a new higher order, effectful programming language (like ML) designed with program verification in mind. Its type system is based on a core that resembles System Fw (hence the name), but is extended with dependent types, refined monadic effects, refinement types, and higher kinds. Together, these features allow expressing precise and compact specifications for programs, including functional correctness properties. The F* type-checker aims to prove that programs meet their specifications using an automated theorem prover (usually Z3) behind the scenes to discharge proof obligations. Programs written in F* can be translated to OCaml, F#, or JavaScript for execution.

URL: <https://www.fstar-lang.org/>

Contact: Aymeric Fromherz

Participants: Antoine Delignat-Lavaud, Catalin Hritcu, Cedric Fournet, Chantal Keller, Karthikeyan Bhargavan, Pierre-Yves Strub, Aymeric Fromherz

7.1.2 Steel

Name: Steel

Keywords: Program verification, Separation Logic

Functional Description: Steel is a framework for the verification of low-level, concurrent software, and is implemented in the F* dependently-typed programming language. Steel combines a strong, expressive concurrent separation logic to reason about complex concurrency patterns with a high level of automation, mixing custom separation logic decision procedures implemented as F* tactics with generic SMT solving to provide safety and functional correctness guarantees about Steel programs. Steel programs can be translated to executable C code to be integrated in unverified projects.

URL: <https://github.com/FStarLang/steel>

Publications: [hal-04104143](#), [hal-02936273](#), [hal-03466397](#), [hal-03626859](#)

Contact: Aymeric Fromherz

Participant: Aymeric Fromherz

Partner: Microsoft

7.1.3 StarMalloc

Keywords: Memory Allocation, Program verification, Separation Logic

Functional Description: StarMalloc is a concurrent, security-oriented, formally verified memory allocator. The functional correctness and memory safety of StarMalloc have been formally established in F*, using the Steel concurrent separation logic. StarMalloc provides all APIs expected from a modern allocator, and was tested with a wide range of applications, including Firefox, Redis, or Z3.

News of the Year: StarMalloc was publicly released, including an entire formally verified development using the Steel concurrent separation logic, as the extracted, executable C code packaged as a standalone, easily deployable artefact. In addition to the APIs and functionalities expected from a modern allocator, the work included the verified development of several security features, including segregated metadata, quarantine, guard pages, and canaries.

StarMalloc was presented at OOPSLA 2024.

URL: <https://github.com/Inria-Prosecco/StarMalloc>

Publication: [hal-04837244](#)

Contact: Aymeric Fromherz

Participants: Aymeric Fromherz, Antonin Reitz

7.1.4 HACL*

Name: High Assurance Cryptography Library

Keywords: Cryptography, Software Verification

Functional Description: HACL* is a formally verified cryptographic library in F*, developed by the Prosecco team at INRIA Paris in collaboration with Microsoft Research, as part of Project Everest.

HACL stands for High-Assurance Cryptographic Library and its design is inspired by discussions at the HACS series of workshops. The goal of this library is to develop verified C reference implementations for popular cryptographic primitives and to verify them for memory safety, functional correctness, and secret independence.

News of the Year: We extended the capabilities of the F* extraction to offer a safe Rust version of HACL* for most algorithms, with the exception of vectorized implementations.

URL: <https://github.com/hacl-star/hacl-star>

Publications: [hal-04301439](#), [hal-03154278](#), [hal-03154275](#), [hal-02294935](#), [hal-01588421](#)

Contact: Aymeric Fromherz

Participants: Karthikeyan Bhargavan, Aymeric Fromherz

7.1.5 DY*

Name: DY*

Keywords: Software Verification, Cryptographic protocol

Functional Description: DY* is a recently proposed formal verification framework for the symbolic security analysis of cryptographic protocol code written in the F* programming language. Unlike automated symbolic provers, DY* accounts for advanced protocol features like unbounded loops and mutable recursive data structures as well as low-level implementation details like protocol state machines and message formats, which are often at the root of real-world attacks. Protocols modeled in DY* can be executed, and hence, tested, and they can even interoperate with real-world counterparts. DY* extends a long line of research on using dependent type systems but takes a fundamentally new approach by explicitly modeling the global trace-based semantics within the framework, hence bridging the gap between trace-based and type-based protocol analyses. With this, one can uniformly, precisely, and soundly model, for the first time using dependent types, long-lived mutable protocol state, equational theories, fine-grained dynamic corruption, and trace-based security properties like forward secrecy and post-compromise security. DY* has been applied to the formal analysis of advanced cryptographic protocols such as Signal, ACME, Noise and MLS.

News of the Year: We added support for MACs in DY*.

URL: <https://reprosec.org/>

Publications: [hal-03178425](#), [hal-03540403](#), [hal-03540824](#), [hal-04255953](#), [hal-04310972](#)

Contact: Theophile Wallez

Participants: Theophile Wallez, Abhishek Bichhawat, Tim Würtele, Pedram Hosseyni

7.1.6 Aeneas

Keywords: Rust, Compilers, Program verification

Functional Description: Aeneas is a compilation pipeline for safe Rust programs. Aeneas leverages the Rust type system to compile Rust programs to pure, executable models (i.e., pure, functional versions of the original Rust programs). The key idea behind Aeneas' compilation is that, under the proper restrictions, a Rust function is fully characterized by a forward function, which computes its return value at call site, and a set of backward functions (one per lifetime), which propagate changes back into the environment upon ending lifetimes, thus accounting for side effects. Such forward and backward functions behave similarly to lenses. Relying on theorem provers to state and prove lemmas about those models, it is then possible to enforce guarantees about the original programs. For instance, one can prove panic freedom and functional correctness, but also security guarantees like authentication and confidentiality in the case of cryptographic protocols, and potentially more.

News of the Year: We further extended the support for proof automation in Lean, adding several custom tactics, as well as relying more broadly on recently developed Lean automation such as the grind tactic. Furthermore, we also extended the subset of Rust supported, generalizing Aeneas' support for loops, extending the Aeneas standard library with models of more standard Rust stdlib functions, improving the existing documentation, and adding support for features such as dynamically sized types.

URL: <https://github.com/AeneasVerif/aeneas>

Publications: [hal-04837250](#), [hal-03931572](#)

Contact: Aymeric Fromherz

Participants: Son Ho, Aymeric Fromherz, Guillaume Boisseau

7.1.7 Charon

Keywords: Rust, Compilers

Functional Description: Charon is a driver which retrieves the Rust compiler output (more precisely, the generated MIR) and translates it to an intermediate language called LLBC (Low Level Borrow Calculus - an "easy-to-use" version of MIR in practice). Charon is meant as a user-friendly, stable interface with the Rust compiler, for the purpose of analyzing Rust programs.

News of the Year: We further extended support for Rust features in Charon, including supporting later MIRs, enabling monomorphization of a crate contents, reworking the representation of closures, emitting layout information for most types, adding pointer metadata information to types (for DSTs), fixing the semantics of drops, and broadly improving the support of generics as well as the performance of the framework.

Additionally, Charon was presented to the CAV conference in July 2025.

URL: <https://github.com/AeneasVerif/charon>

Publications: [hal-03931572](#), [hal-05175922](#)

Contact: Aymeric Fromherz

Participants: Son Ho, Guillaume Boisseau, Aymeric Fromherz, Yoann Prak

7.1.8 Catala

Keywords: Domain specific, Programming language, Law

Functional Description: Catala is a domain-specific programming language designed for deriving correct-by-construction implementations from legislative texts. Its specificity is that it allows direct translation from the text of the law using a literate programming style, that aims to foster interdisciplinary dialogue between lawyers and software developers. By enjoying a formal specification and a proof-oriented design, Catala also opens the way for formal verification of programs implementing legislative specifications.

News of the Year: In 2025, the development of Catala was further pursued by a team split between the Apollo program and the Prosecco team. On the research side, we also developed CUTEcat, a concolic execution engine for Catala which is able to automatically detect uncovered cases and ambiguities in existing Catala implementations.

URL: <https://catala-lang.org/en>

Publications: [hal-04391612](#), [hal-03712130](#), [hal-03781578](#), [hal-04907935](#), [hal-03128248](#), [hal-03159939](#), [hal-02936606](#), [hal-03869335](#)

Contact: Denis Merigoux

Participants: Vincent Botbol, Romain Primet, Denis Merigoux, Louis Gesbert, Aymeric Fromherz, Alain Delaet-Tixeuil, Raphael Monat

Partner: Université Panthéon-Sorbonne

7.1.9 ProVerif

Keywords: Security, Verification, Cryptographic protocol

Functional Description: ProVerif is an automatic security protocol verifier in the symbolic model (so called Dolev-Yao model). In this model, cryptographic primitives are considered as black boxes. This protocol verifier is based on an abstract representation of the protocol by Horn clauses. Its main features are:

It can verify various security properties (secrecy, authentication, process equivalences).

It can handle many different cryptographic primitives, specified as rewrite rules or as equations.

It can handle an unbounded number of sessions of the protocol (even in parallel) and an unbounded message space.

URL: <http://proverif.inria.fr/>

Publications: [hal-03366962](#), [hal-01947972](#), [hal-01423742](#), [hal-01306440](#), [hal-01423760](#), [hal-01102136](#), [hal-01575920](#), [hal-01528752](#), [hal-01575923](#), [hal-01527671](#), [hal-01575861](#)

Contact: Bruno Blanchet

Participants: Bruno Blanchet, Vincent Cheval, an anonymous participant

7.1.10 CryptoVerif

Name: Cryptographic protocol verifier in the computational model

Keywords: Security, Verification, Cryptographic protocol

Functional Description: CryptoVerif is an automatic protocol prover sound in the computational model. In this model, messages are bitstrings and the adversary is a polynomial-time probabilistic Turing machine. CryptoVerif can prove secrecy and correspondences, which include in particular authentication. It provides a generic mechanism for specifying the security assumptions on cryptographic primitives, which can handle in particular symmetric encryption, message authentication codes, public-key encryption, signatures, hash functions, and Diffie-Hellman key agreements. It also provides an explicit formula that gives the probability of breaking the protocol as a function of the probability of breaking each primitives, this is the exact security framework.

News of the Year: The main new features of the year are:

- 1) We allow proving that, if some events happened, then other events did not happen, in addition to proving that if some events happened, then other events happened. (Teams involved: Pesto, Prosecco.)
- 2) We allow proving security properties on a subset of the traces of the analyzed protocol. The considered subset of traces is defined by so-called restrictions, which specify that certain events must happen or not happen. Restrictions are useful in particular to model complex compromise scenarios. (Teams involved: Pesto, Prosecco.)
- 3) We considerably improved the transformation that merges branches that execute the same code. In particular, the "same code" is now defined up to reorganizations of assignments and random number generations, up to reordering of branches and indices of find, and up to the candidate merges that are possible in each of the branches. We also allow merging the then branches of a find into a single branch, when the else branch is unreachable. Finally, we allow the user to specify which branches to merge, for a more fine-grained guidance. (Team involved: Prosecco.)

These changes are included in CryptoVerif version 2.12 available at <https://cryptoverif.inria.fr>.

URL: <http://cryptoverif.inria.fr/>

Publications: [hal-03113251](#), [hal-03471218](#), [hal-04246199](#), [hal-04253820](#), [hal-01947959](#), [hal-01764527](#), [hal-02396640](#), [hal-02100345](#), [hal-04321656](#), [hal-04271666](#), [hal-04577912](#), [tel-01112630](#), [hal-01102382](#), [hal-01528752](#), [hal-01575920](#), [hal-01575861](#), [hal-01575923](#)

Contact: Bruno Blanchet

Participants: Bruno Blanchet, Pierre Boutry, David Cade, Christian Doczkal, Aymeric Fromherz, Charlie Jacomme, Benjamin Lipp, Pierre-Yves Strub

7.1.11 Squirrel

Name: Squirrel Prover

Keywords: Proof assistant, Cryptographic protocol

Functional Description: Squirrel is an interactive proof assistant dedicated to the formal verification of cryptographic protocols in the computational model. It is based on a higher-order probabilistic logic which supports generic mathematical reasoning as well as cryptographic-specific reasoning.

Concretely, Squirrel allows to specify security protocols in a variant of the applied pi-calculus, and properties of those protocols using its probabilistic logic. Then, these properties are to be proved by the users through tactics. Squirrel supports protocols with unbounded replication and persistent state, and can express both correspondence (e.g. authentication) and indistinguishability properties (e.g. strong secrecy, unlinkability).

News of the Year: We added support for user-defined functions which can use probabilistic constructs, mutual recursion, system-dependency and pattern matching. (Teams implied: Pesto, Prosecco.)

We improved the simulator synthesis procedure behind the 'crypto' tactic in Squirrel, by adding support for synthesizing memoizing simulators, and by allowing to infer time-sensitive memory invariant. (Team implied: Prosecco.)

We completely re-designed and re-implemented the post-quantum variant of Squirrel, making it more powerful and more maintainable. (Teams implied: Pesto, Prosecco.)

URL: <https://squirrel-prover.github.io/>

Publications: [hal-04884758](#), [hal-04577828](#), [hal-04511718](#), [hal-04579038](#), [hal-03981949](#), [hal-03620358](#), [hal-03172119](#), [hal-03264227](#)

Contact: Adrien Koutsos

Participants: Joseph Lallemand, David Baelde, Stephanie Delaune, Clément Herouard, Charlie Jacomme, Adrien Koutsos, Solene Moreau, Thomas Rubiano, Justine Sauvage, Theo Vignon

Partners: IRISA, ENS Rennes

8 New results

8.1 Verification of security protocols in the computational model

Participants: Bruno Blanchet, Aymeric Fromherz, Adrien Koutsos, Emmanuel Mera, Justine Sauvage, Théo Vignon.

CryptoVerif We (Bruno Blanchet and Charlie Jacomme) continued the development of **CRYPTOVERIF** as summarized in Section 7.1. We also continued the development of our translation from **CRYPTOVERIF** to **F*** (by Karthikeyan Bhargavan, Bruno Blanchet, Aymeric Fromherz, Benjamin Lipp), which allows us to generate running implementations of protocols verified in **CRYPTOVERIF**. An important addition with respect to a previous translation to OCaml is that we generate **F*** axioms for security properties proved by **CRYPTOVERIF** (these axioms can then be used in **F*** proofs) and lemmas for equations used as assumptions in **CRYPTOVERIF** (these lemmas are then proved in **F***). In particular, this year, Emmanuel Mera, supervised by Bruno Blanchet, Charlie Jacomme, and Aymeric Fromherz, worked on a case study inspired by the messaging protocol Signal and its session management protocol Sesame, to illustrate the combination of **CRYPTOVERIF** and **F*** proofs in this framework. The core of the cryptographic protocol is proved secure in **CRYPTOVERIF**, and its **F*** implementation is generated by our framework, together with associated security axioms; the session management part is implemented directly in **F***; the security of the whole system is then proved in **F***, by relying on the security axioms obtained from the **CRYPTOVERIF** proof of the cryptographic core.

Squirrel **SQUIRREL** is an interactive theorem prover dedicated to the verification of cryptographic protocols in the computational model. This tool, first introduced in [2], encodes cryptographic protocols and their properties into a pure probabilistic logic, and supports generic as well as cryptographic-specific reasoning.

Cryptographic proofs proceed in large part by reductions to cryptographic assumptions expressed as games. These reductions rely on simulators which are often tedious to write and involve a significant amount of trivial code. Thus, simulators are only sketched in pen-and-paper proofs, which is error-prone. Mechanized cryptographic proofs remove the risk of errors, but requiring users to explicitly write simulators is an unreasonable burden. In **SQUIRREL**, we solve the simulator synthesis problem using cryptographic bi-deduction, a technique recently developed and integrated in the tool [23]. More precisely, the bi-deduction technique introduced in [23] takes the form of a proof system and a simple proof-search procedure for it. In recent work, we showed that this procedure suffers from systematic failures when working with games such as IND-CCA2. We provided a significantly improved procedure, that can re-use oracle calls across recursive iterations, and generates precise invariants to justify it. We implemented this procedure in **SQUIRREL** and validate it in a proof of ballot privacy for the FOO e-voting protocol, which is the first computational mechanized proof for FOO, and the most complex **SQUIRREL** proof to date. This work has been accepted and will appear at Usenix'26 [11].

The advent of quantum computers has initiated both research and standardization efforts toward the development of cryptographic primitives and communication protocols that remain secure against attackers

equipped with quantum computers. To validate novel post-quantum cryptographic designs, it is necessary to adapt computer-aided verification tools to a post-quantum setting. In recent work, we presented a novel post-quantum extension of **SQUIRREL**. Our extension is fully embedded in the higher-order logic underlying **SQUIRREL**, and allows for logical terms to directly represent quantum values. This design choice makes the extension generic, preserves compatibility with the latest features of **SQUIRREL**, and supports its long-term integration into the tool. We implemented our approach within **SQUIRREL** and validated it through several case studies. In particular, we obtain post-quantum security guarantees for multiple KEM combinators, as well as for two hybrid key-exchange protocols. This work is under submission.

8.2 Verification of cryptographic protocol implementations in the symbolic model: the DY* framework

Participants: Théophile Wallez.

In collaboration with colleagues at the University of Stuttgart and IIT Gandhinagar, we developed DY*, a new formal verification framework for the symbolic security analysis of cryptographic protocol code written in the F* programming language. Unlike automated symbolic provers, our framework accounts for advanced protocol features like unbounded loops and mutable recursive data structures, as well as low-level implementation details like protocol state machines and message formats, which are often at the root of real-world attacks.

This year, we (Théophile Wallez) extended DY* with support for message authentication codes (MACs).

We (Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan) also used DY* to analyze the Messaging Layer Security (MLS) protocol standard [25]. MLS proposes a novel tree-based protocol that enables efficient end-to-end encrypted messaging over large groups with thousands of members. Its functionality can be divided into three components: TreeSync for authenticating and synchronizing group state, TreeKEM for the core group key agreement, and TreeDEM for group message encryption. While previous works have analyzed the security of abstract models of TreeKEM, they do not account for the precise low-level details of the protocol standard. We presented the first machine-checked security proof for TreeKEM. Our proof is in the symbolic Dolev-Yao model and applies to a bit-level precise, executable, interoperable specification of the protocol. Furthermore, our security theorem for TreeKEM composes naturally with a previous result for TreeSync [52] to provide a strong modular security guarantee for the published MLS standard. This work appears at IEEE S&P'25 [14].

Théophile Wallez defended his PhD thesis [16] on DY* and its application to the analysis of MLS.

8.3 High-Assurance High-Performance Cryptography

Participants: Aymeric Fromherz, Florian Duzes.

Since 2017, we maintain and distribute the HACL* verified cryptographic library, which is currently deployed in many mainstream software applications and high-performance networking stacks including Mozilla Firefox, Linux Kernel, WireGuard VPN, Microsoft WinQuic, Tezos Blockchain, and ElectionGuard.

HACL* formal guarantees include the safety, functional correctness, and side-channel resistance of the library. However, these guarantees apply to the source C program; the compilation process (using mainstream compilers such as GCC or Clang) is part of the trusted computing base. Unfortunately, even when starting from constant-time, secret-independent code, C compilers can reintroduce side-channel vulnerabilities at the binary level.

To address this issue, we investigated the application of binary analysis techniques to validate that HACL*'s side-channel resistance guarantees are preserved after compilation. As part of Florian Duzes's internship, we (Aymeric Fromherz, in collaboration with Yanis Sellami, Frédéric Recoules, and Sébastien Bardin from CEA) developed a framework to automatically apply the BINSEC binary analyzer on HACL*

binaries compiled with different compilers, optimization levels, and for different architectures. This work is still ongoing.

8.4 Formal Verification of Rust Programs

Participants: Aymeric Fromherz, Guillaume Boisseau, Yoann Prak, Fernando Leal Sanchez.

In collaboration with Son Ho and Jonathan Protzenko, we (Aymeric Fromherz) develop Aeneas, a new verification toolchain for Rust programs. Aeneas leverages Rust’s rich region-based type system to eliminate memory reasoning for a large class of Rust programs, as long as they do not rely on interior mutability or unsafe code. Doing so, Aeneas relieves the proof engineer of the burden of memory-based reasoning, allowing them to instead focus on functional properties of their code. Aeneas proposes a new Low-Level Borrow Calculus (LLBC) that captures a large subset of Rust programs, and a translation from LLBC to a pure lambda-calculus, which enables the verification of Rust programs through different theorem provers, most notably Lean.

This year, we particularly worked on improvements to the Lean backend of Aeneas. As part of Fernando Leal Sanchez’s internship, we extended support for proof automation in Lean, including the development of new tactics targeting the specific shape of models extracted by Aeneas, and experimented with the recently released grind tactic, which provides SMT-like automation inside the proof assistant. Furthermore, we also extended the extraction facilities of Aeneas to get closer to translating existing Rust projects as-is. This required generalizing Aeneas’ support for loops, nested borrows, and extending the Aeneas standard library to support a wider part of the Rust stdlib.

Additionally, to simplify the development of analysis and verification tools for Rust, we (Guillaume Boisseau, Aymeric Fromherz, Yoann Prak, in collaboration with Jonathan Protzenko and Son Ho) develop Charon, a framework that serves as an interface to the Rust compiler. Charon is able to leverage the cargo infrastructure widely used in Rust projects to drive the rustc compiler, and output an intermediate representation of Rust programs extracted from MIR (an intermediate representation of the Rust compiler), where additional, analysis-relevant semantic information has been reconstructed. Charon was presented this year at the CAV conference [13].

8.5 Formalizing and Implementing Law

Participants: Aymeric Fromherz, Vincent Botbol.

The Catala domain-specific language, whose development started in the Prosecco team in 2021, aims to simplify the implementation of legal expert systems centered on computational law (e.g., taxesh or social benefits). The language has been built in close collaboration with lawyers, and advertised to that community with a number of legal-oriented papers [43, 42]. This year, Vincent Botbol, jointly with the Catala development team led by Denis Merigoux as part of the Apollo project released Catala 1.0, the first major version of Catala.

As part of the AVoCat exploratory action, we (Aymeric Fromherz, in collaboration with Raphael Monat and Pierre Goutagny) also explore the application of formal verification techniques to Catala programs. This year, we proposed a concolic execution tool for Catala, called CUTEcAt, which is able to automatically reason about the default logic at the heart of the Catala language, and to identify both uncovered cases and ambiguities in legal implementations, while generating user-friendly testcases when issues occur. This work was presented at ESOP 25 [12], and received a Distinguished Artifact Award.

9 Bilateral contracts and grants with industry

Cybercampus CIRCUS funding Creating Innovative and Robust Cryptographic Solutions.

Participants: Aymeric Fromherz.

- Partners: Inria Paris/EPI Prosecco, Cryspen.
- Prosecco PI: Aymeric Fromherz
- Abstract: This project aims to build a new integrated development and verification environment (IDVE) called Circus that is targeted at software developers and security architects. The main partner for this technical transfer is Cryspen, a new company spun off from Prosecco that aims to create innovative cryptographic solutions. The Circus IDVE targets the Rust language, and consists of several tools developed at Prosecco, such as hacspec and Aeneas, while relying on well-established verification tools for the verification of safety, correctness, and security properties about critical software.

Amazon Research Award

Participants: Aymeric Fromherz.

- Partner: Inria Paris/EPI Prosecco
- PI: Aymeric Fromherz
- Abstract: This project aims to improve the Aeneas framework, which enables the verification of Rust programs by translating them to semantically equivalent functional models in proof assistants such as Lean. The goal of the project is to explore the development of custom, extensible proof automation in Lean, by leveraging Lean’s metaprogramming facilities.

10 Partnerships and cooperations

10.1 National initiatives

10.1.1 PEPR

PEPR Cybersecurity SVP

Participants: Bruno Blanchet (local PI), Aymeric Fromherz, Adrien Koutsos, Justine Sauvage, Théo Vignon.

Title: SVP – Verification of Security Protocols

Other partners: IRISA/team SPICY, Inria Nancy/EPI PESTO, Inria Sophia Antipolis/EPI STAMP, LMP - ENS Paris-Saclay/team INSPIRE.

Duration: July 2022–June 2028

Coordinator: Stéphanie Delaune, IRISA/Équipe SPICY

Summary: The SVP project aims at enabling the analysis of protocols (either already deployed or in the design phase) at the level of abstract specifications, both symbolic and computational, as well as implementations. We want to develop techniques and tools allowing the implementation of solutions whose security will not be questioned in a cyclic way. To achieve this challenge, we (i) develop new functionalities in existing tools to allow the analysis of more and more complex protocols ; (ii) build bridges between the different existing proof techniques and associated tools in order to take advantage of the strengths of each of them ; (iii) validate the techniques and tools developed within this project on widely deployed protocols and on more recent, fast-growing applications, such as Internet voting.

PEPR Quantic PQ-TLS

Participants: Bruno Blanchet (local PI), Aymeric Fromherz.

Title: PQ-TLS: Post-quantum padlock for web browsers

Other partners: Université Rennes I, Université de Limoges, Université de Rouen, Université de Bordeaux, Université de Saint-Quentin-en-Yvelines, Université de Saint-Étienne, ENS de Lyon, Inria (EPI Grace, Caramba, Cosmiq, Cascade), CEA LETI, CNRS (IMB, IRISA, LABSTICC, LHC, LIP, LIX, LMV, LORIA, XLIM), ANSSI, CryptoNext, PQShield SAS, CryptoExperts

Duration: January 2022–December 2028

Coordinator: Pierre Alain-Fouque, Université Rennes I

Summary: The famous “padlock” appearing in browsers when one visits websites whose address is preceded by “https” relies on cryptographic primitives that would not withstand a quantum computer. This integrated project aims to develop in 5 years post-quantum primitives in a prototype of “post-quantum lock” that will be implemented in an open source browser. The evolution of cryptographic standards has already started, the choice of new primitives will be made quickly, and the transition will be made in the next few years. The objective is to play a driving role in this evolution and to make sure that the French actors of post-quantum cryptography, already strongly involved, are able to influence the cryptographic standards of the decades to come.

10.1.2 ANR

HOPR

Participants: Adrien Koutsos (local PI) .

Title: HOPR – Higher-Order Probabilistic and resource-aware Reasoning

Other partners: CNRS (IRISA), Inria (Sophia Antipolis/EPI SPLITS, Lille/EPC SyCoMoRES)

Duration: January 2025–December 2028

Coordinator: Patrick Baillot, Inria Lille

Participants: Adrien Koutsos

Summary: The HOPR project aims at: i) developing formal logical framework allowing for the **combination of higher-order computation with both probabilities and resources**; and ii), the application of these novel frameworks to improve practical verification tools for cryptography (notably **SQUIRREL** and EasyCrypt). Combining higher-order computation, probabilistic reasoning and representation of complexity-bounded computation is inherently difficult because these features have non-trivial interactions. Nonetheless, we believe that doing so will bring greater *scalability* (developing proofs for larger systems), *expressivity* (expressing properties against new classes of attackers) and *extensibility* (allowing existing tools to handle new notions of privacy) to verification tools.

11 Dissemination

11.1 Promoting scientific activities

11.1.1 Scientific events: selection

Member of the conference program committees

- Bruno Blanchet: PC member for CSF'26.
- Aymeric Fromherz: PC member for CSF'25, CSF'26, CPP'26, ITP'25, PriSc'26, USENIX Security'25.
- Adrien Koutsos: PC member for CCS'25, S&P'26, CSF'25, CSF'26.

11.1.2 Journal

Member of the editorial boards

- Adrien Koutsos: Editor of the ACM Transactions on Privacy and Security (ACM TOPS).

11.1.3 Invited talks

- Bruno Blanchet gave an invited talk at the cryptography seminar of Université de Rennes.

11.1.4 Research administration

- Adrien Koutsos: member of the selection committee for “maître de conférence” at Université Paris-Saclay/LMF, spring 2025.

11.2 Teaching - Supervision - Juries - Educational and pedagogical outreach

11.2.1 Teaching

- Master: Bruno Blanchet, Proofs of security protocols, 9h equivalent TD, master M2 MPRI, Université Paris VII
- Master: Adrien Koutsos, Proofs of security protocols, 18h equivalent TD, master M2 MPRI, Université Paris VII
- Master: Aymeric Fromherz, Proofs of security protocols, 18h equivalent TD, master M2 MPRI, Université Paris VII
- Summer School: Aymeric Fromherz, High-Assurance Verification of Security-critical Low-level code using F^* and Low^* , Summer School on Security Testing and Verification, 3h.

11.2.2 Supervision

- PhD in progress: Marco Bertoni, Automated and Incremental Side-Channel Analysis, since December 2025, supervised by Aymeric Fromherz, Yanis Sellami, and Bruno Blanchet.
- PhD in progress: Théo Vignon, Exploring the limits of the CCSA approach to computational security, since September 2023, supervised by Caroline Fontaine, Guillaume Scerri, and Adrien Koutsos.
- PhD in progress: Justine Sauvage, Games and Logic for the Verification of Cryptographic Protocols, since September 2022, supervised by Adrien Koutsos, David Baelde, and Bruno Blanchet.
- PhD in progress: Pierre Goutagny, PhD student in EPC SyCoMoRES (Inria, University of Lille, and CNRS) on Automated Verification of Catala Programs, since September 2024, supervised by Patrick Baillot, Raphael Monat, and Aymeric Fromherz.
- PhD defended: Antonin Reitz, A Methodology for Programming and Verifying Secure Systems, defended on December 5, 2025, supervised by Aymeric Fromherz and Bruno Blanchet.
- PhD defended: Théophile Wallez, Verification of Cryptographic Protocols, defended on June 24, 2025, supervised by Karthikeyan Bhargavan, Bruno Blanchet, and Jonathan Protzenko [16].
- PhD defended: Théo Laurent, Structural Subtyping in MLTT, defended on April 3, 2025, supervised by David Delahaye and Kenji Maillard [15].

- M2 internship: Emmanuel Mera, case study in CV2F*, supervised by Bruno Blanchet, Aymeric Fromherz, and Charlie Jacomme.
- M2 internship: Florian Duzes, Automated binary analysis for HACL*, supervised by Aymeric Fromherz, Yanis Sellami, and Frédéric Recoules.
- M2 internship: Vivien Gachet, Static Analysis of Rust programs, supervised by Aymeric Fromherz.
- M1 internship: Fernando Leal Sanchez, Formal Verification of Rust Post-Quantum Cryptographic Implementations, supervised by Aymeric Fromherz and Son Ho.

11.2.3 Juries

- Bruno Blanchet: reviewer of Clément Hérouard’s PhD thesis, Université de Rennes.

12 Scientific production

12.1 Major publications

- [1] M. Abadi, B. Blanchet and C. Fournet. ‘The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication’. In: *Journal of the ACM (JACM)* 65.1 (Oct. 2017), pp. 1–103. doi: [10.1145/3127586](https://doi.org/10.1145/3127586). URL: <https://hal.inria.fr/hal-01636616>.
- [2] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos and S. Moreau. ‘An Interactive Prover for Protocol Verification in the Computational Model’. In: SP 2021 - 42nd IEEE Symposium on Security and Privacy. Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P’21). San Francisco / Virtual, United States, 23rd May 2021. URL: <https://hal.archives-ouvertes.fr/hal-03172119> (cit. on p. 17).
- [3] K. Bhargavan, B. Blanchet and N. Kobeissi. ‘Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate’. In: *38th IEEE Symposium on Security and Privacy*. San Jose, United States, May 2017, pp. 483–502. doi: [10.1109/SP.2017.26](https://doi.org/10.1109/SP.2017.26). URL: <https://hal.inria.fr/hal-01575920> (cit. on p. 8).
- [4] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti and P.-Y. Strub. ‘Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS’. In: *IEEE Symposium on Security and Privacy (Oakland)*. 2014, pp. 98–113. URL: <https://hal.inria.fr/hal-01102259>.
- [5] B. Blanchet. ‘Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif’. In: *Foundations and Trends in Privacy and Security* 1.1–2 (Oct. 2016), pp. 1–135. URL: <https://hal.inria.fr/hal-01423760>.
- [6] B. Blanchet, V. Cheval and V. Cortier. ‘ProVerif with Lemmas, Induction, Fast Subsumption, and Much More’. In: S&P’22 - 43rd IEEE Symposium on Security and Privacy. San Francisco, United States, 22nd May 2022. URL: <https://hal.inria.fr/hal-03366962>.
- [7] A. Koutsos. ‘The 5G-AKA Authentication Protocol Privacy’. In: *EuroS&P 2019 - IEEE European Symposium on Security and Privacy*. Stockholm, Sweden: IEEE, June 2019, pp. 464–479. doi: [10.1109/EuroSP.2019.00041](https://doi.org/10.1109/EuroSP.2019.00041). URL: <https://hal.inria.fr/hal-03155483>.
- [8] D. Merigoux, N. Chataing and J. Protzenko. ‘Catala: A Programming Language for the Law’. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (Aug. 2021), 77:1–29. doi: [10.1145/3473582](https://doi.org/10.1145/3473582). URL: <https://inria.hal.science/hal-03159939>.
- [9] N. Swamy, C. Hrițcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J. K. Zinzindohoué and S. Zanella-Béguelin. ‘Dependent Types and Multi-Monadic Effects in F*’. In: *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, Jan. 2016, pp. 256–270. URL: <https://hal.inria.fr/hal-01265793> (cit. on pp. 8, 9).

- [10] J. K. Zinzindohoué, K. Bhargavan, J. Protzenko and B. Beurdouche. ‘HACL*: A Verified Modern Cryptographic Library’. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017, pp. 1789–1806. URL: <https://hal.inria.fr/hal-01588421> (cit. on pp. 8, 9).

12.2 Publications of the year

International peer-reviewed conferences

- [11] D. Baelde, A. Koutsos and J. Sauvage. ‘Leveraging Cryptographic Simulator Synthesis for Formally Verifying the FOO E-Voting Protocol’. In: Usenix 2026 - 35th Usenix security symposium. Baltimore, United States, 12th Aug. 2026. URL: <https://inria.hal.science/hal-05453231> (cit. on p. 17).
- [12] P. Goutagny, A. Fromherz and R. Monat. ‘CUTECat: Concolic Execution for Computational Law’. In: ESOP 2025 - 34th European Symposium on Programming. Hamilton, ON, Canada, 3rd May 2025. URL: <https://inria.hal.science/hal-04907935> (cit. on pp. 11, 19).
- [13] S. Ho, G. Boisseau, L. Franceschino, Y. Prak, A. Fromherz and J. Protzenko. ‘Charon: An Analysis Framework for Rust’. In: CAV 2025 - International Conference on Computer Aided Verification. Vol. 15934. Lecture Notes in Computer Science. Zagreb, Croatia: Springer Nature Switzerland, 23rd July 2025, pp. 377–391. DOI: [10.1007/978-3-031-98685-7_18](https://doi.org/10.1007/978-3-031-98685-7_18). URL: <https://inria.hal.science/hal-05175922> (cit. on p. 19).
- [14] T. Wallez, J. Protzenko and K. Bhargavan. ‘TreeKEM: A Modular Machine-Checked Symbolic Security Analysis of Group Key Agreement in Messaging Layer Security’. In: SP 2025 IEEE Symposium on Security and Privacy. San Francisco, United States: IEEE, 12th May 2025, pp. 4375–4390. DOI: [10.1109/SP61157.2025.00228](https://doi.org/10.1109/SP61157.2025.00228). URL: <https://inria.hal.science/hal-05441655> (cit. on p. 18).

Doctoral dissertations and habilitation theses

- [15] T. Laurent. ‘Structural Subtyping in MLTT’. Université de montpellier, 3rd Apr. 2025. URL: <https://theses.hal.science/tel-05144931> (cit. on p. 22).
- [16] T. Wallez. ‘A Verification Framework for Secure Group Messaging’. École Normale Supérieure Paris, 24th June 2025. URL: <https://hal.science/tel-05455122> (cit. on pp. 18, 22).

12.3 Cited publications

- [17] D. Merigoux, M. Alauzen and L. Slimani. ‘Rules, Computation and Politics: Scrutinizing Unnoticed Programming Choices in French Housing Benefits’. In: *Journal of Cross-disciplinary Research in Computational Law* 1.4 (2023). URL: <https://inria.hal.science/hal-03712130> (cit. on p. 10).
- [18] D. Merigoux. ‘Experience report: implementing a real-world, medium-sized program derived from a legislative specification’. In: Programming Languages and the Law 2023 (affiliated with POPL). Boston (MA), United States, 15th Jan. 2023. URL: <https://inria.hal.science/hal-03933574> (cit. on p. 10).
- [19] S. Ho. ‘Formal Verification of Rust Programs by Functional Translation’. Université PSL (Paris Sciences & Lettres), 9th Dec. 2024. URL: <https://theses.hal.science/tel-05004605> (cit. on p. 11).
- [20] M. Abadi and B. Blanchet. ‘Analyzing Security Protocols with Secrecy Types and Logic Programs’. In: *Journal of the ACM* 52.1 (Jan. 2005), pp. 102–146. URL: <https://bblanche.gitlabpages.inria.fr/publications/AbadiBlanchetJACM7037.pdf> (cit. on p. 7).
- [21] M. Abadi, B. Blanchet and C. Fournet. ‘Just Fast Keying in the Pi Calculus’. In: *ACM Transactions on Information and System Security (TISSEC)* 10.3 (July 2007), pp. 1–59. URL: <https://bblanche.gitlabpages.inria.fr/publications/AbadiBlanchetFournetTISSEC07.pdf> (cit. on p. 8).

- [22] D. Ahman, C. Hritcu, K. Maillard, G. Martínez, G. Plotkin, J. Protzenko, A. Rastogi and N. Swamy. ‘Dijkstra Monads for Free’. In: *44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, Jan. 2017, pp. 515–529. doi: [10.1145/3009837.3009878](https://doi.org/10.1145/3009837.3009878). URL: <https://www.fstar-lang.org/papers/dm4free/> (cit. on pp. 8, 9).
- [23] D. Baelde, A. Koutsos and J. Sauvage. ‘Foundations for Cryptographic Reductions in CCSA Logics’. In: *CCS*. ACM, 2024, pp. 2814–2828 (cit. on p. 17).
- [24] G. Bana, P. Adaõ and H. Sakurada. ‘Computationally Complete Symbolic Adversary and Computationally Sound Verification of Security Protocols (in Japanese)’. In: *Proceedings of The 30th Symposium on Cryptography and Information Security*. CD-ROM (4D1-3), Jan. 2013 (cit. on p. 8).
- [25] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara and K. Cohn-Gordon. *The Messaging Layer Security (MLS) Protocol*. RFC 9420, available at <https://datatracker.ietf.org/doc/rfc9420/>. July 2024 (cit. on p. 18).
- [26] E. D. Berger. ‘Software needs seatbelts and airbags’. In: *Communications of the ACM* 55.9 (2012), pp. 48–53 (cit. on p. 6).
- [27] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseini, R. Küsters, G. Schmitz and T. Würtele. ‘DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code’. In: *EuroS&P 2021 - 6th IEEE European Symposium on Security and Privacy*. Virtual, Austria, Sept. 2021. URL: <https://hal.inria.fr/hal-03178425> (cit. on p. 8).
- [28] K. Bhargavan, B. Bond, A. Delignat-Lavaud, C. Fournet, C. Hawblitzel, C. Hritcu, S. Ishtiaq, M. Kohlweiss, R. Leino, J. Lorch, K. Maillard, J. Pang, B. Parno, J. Protzenko, T. Ramananandro, A. Rane, A. Rastogi, N. Swamy, L. Thompson, P. Wang, S. Zanella-Béguelin and J.-K. Zinzindohoué. ‘Everest: Towards a Verified, Drop-in Replacement of HTTPS’. In: *2nd Summit on Advances in Programming Languages (SNAPL)*. May 2017. URL: <http://drops.dagstuhl.de/opus/volltexte/2017/7119/pdf/LIPICs-SNAPL-2017-1.pdf> (cit. on p. 8).
- [29] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Pan, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Béguelin and J. K. Zinzindohoué. ‘Implementing and Proving the TLS 1.3 Record Layer’. In: *IEEE Symposium on Security and Privacy (Oakland)*. 2017 (cit. on pp. 8, 9).
- [30] K. Bhargavan, C. Fournet, R. Corin and E. Zalescu. ‘Verified Cryptographic Implementations for TLS’. In: *ACM Transactions Inf. Syst. Secur.* 15.1 (Mar. 2012), 3:1–3:32. doi: [10.1145/2133375.2133378](https://doi.org/10.1145/2133375.2133378). URL: <http://doi.acm.org/10.1145/2133375.2133378> (cit. on p. 8).
- [31] K. Bhargavan, C. Fournet, A. D. Gordon and N. Swamy. ‘Verified implementations of the information card federated identity-management protocol’. In: *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. 2008, pp. 123–135 (cit. on p. 8).
- [32] B. Blanchet. ‘An Efficient Cryptographic Protocol Verifier Based on Prolog Rules’. In: *14th IEEE Computer Security Foundations Workshop (CSFW’01)*. 2001, pp. 82–96 (cit. on p. 7).
- [33] B. Blanchet. ‘Automatic Verification of Correspondences for Security Protocols’. In: *Journal of Computer Security* 17.4 (July 2009), pp. 363–434. URL: <https://bblanche.gitlabpages.inria.fr/publications/BlanchetJCS08.pdf> (cit. on p. 7).
- [34] B. Blanchet, M. Abadi and C. Fournet. ‘Automated Verification of Selected Equivalences for Security Protocols’. In: *Journal of Logic and Algebraic Programming* 75.1 (2008), pp. 3–51. URL: <https://bblanche.gitlabpages.inria.fr/publications/BlanchetAbadiFournetJLAP07.pdf> (cit. on p. 7).
- [35] B. Blanchet and A. Podelski. ‘Verification of Cryptographic Protocols: Tagging Enforces Termination’. In: *Theoretical Computer Science* 333.1-2 (Mar. 2005). Special issue FoSSaCS’03., pp. 67–90. URL: <https://bblanche.gitlabpages.inria.fr/publications/BlanchetPodelskiTCS04.html> (cit. on p. 7).
- [36] D. Cadé and B. Blanchet. ‘Proved Generation of Implementations from Computationally Secure Protocol Specifications’. In: *Journal of Computer Security* 23.3 (2015), pp. 331–402 (cit. on p. 9).
- [37] A. Delaët, D. Merigoux and A. Fromherz. ‘Turning Catala into a Proof Platform for the Law’. In: *Programming Languages and the Law, workshop affiliated with POPL 2022*. Jan. 2022. URL: <https://hal.inria.fr/hal-03447072> (cit. on p. 10).

- [38] A. Delignat-Lavaud, K. Bhargavan and S. Maffei. ‘Language-Based Defenses Against Untrusted Browser Origins’. In: *Proceedings of the 22th USENIX Security Symposium*. 2013. URL: <http://prosecco.inria.fr/personal/karthik/pubs/language-based-defenses-against-untrusted-origins-sec13.pdf> (cit. on p. 10).
- [39] D. Dolev and A. Yao. ‘On the security of public key protocols’. In: *IEEE Transactions on Information Theory* IT-29.2 (1983), pp. 198–208 (cit. on p. 7).
- [40] C. Fournet, M. Kohlweiss and P.-Y. Strub. ‘Modular Code-Based Cryptographic Verification’. In: *ACM Conference on Computer and Communications Security*. 2011 (cit. on p. 8).
- [41] S. Ho and J. Protzenko. ‘Aeneas: Rust verification by functional translation’. In: *Proceedings of the ACM on Programming Languages* 6.ICFP (Aug. 2022), pp. 711–741. DOI: [10.1145/3547647](https://doi.org/10.1145/3547647). URL: <https://hal.science/hal-03931572> (cit. on p. 9).
- [42] L. Huttner and D. Merigoux. ‘Catala: Moving Towards the Future of Legal Expert Systems’. In: *Artificial Intelligence and Law* (Aug. 2022). DOI: [10.1007/s10506-022-09328-5](https://doi.org/10.1007/s10506-022-09328-5). URL: <https://hal.inria.fr/hal-02936606> (cit. on p. 19).
- [43] L. Huttner and D. Merigoux. ‘Traduire la loi en code grâce au langage de programmation Catala’. In: *Intelligence artificielle et finances publiques*. Nice, France, Oct. 2020. URL: <https://inria.hal.science/hal-03128248> (cit. on pp. 10, 19).
- [44] N. Kobeissi, K. Bhargavan and B. Blanchet. ‘Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach’. In: *2nd IEEE European Symposium on Security and Privacy*. Paris, France, Apr. 2017, pp. 435–450. DOI: [10.1109/EuroSP.2017.38](https://doi.org/10.1109/EuroSP.2017.38). URL: <https://hal.inria.fr/hal-01575923> (cit. on p. 8).
- [45] K. Maillard, D. Ahman, R. Atkey, G. Martínez, C. Hritcu, E. Rivas and É. Tanter. ‘Dijkstra Monads for All’. In: *PACMPL* 3.ICFP (2019), 104:1–104:29. DOI: [10.1145/3341708](https://doi.org/10.1145/3341708). URL: <https://arxiv.org/abs/1903.01237> (cit. on p. 8).
- [46] R. Needham and M. Schroeder. ‘Using encryption for authentication in large networks of computers’. In: *Communications of the ACM* 21.12 (1978), pp. 993–999 (cit. on p. 7).
- [47] M. Polubelova, K. Bhargavan, J. Protzenko, B. Beurdouche, A. Fromherz, N. Kulatova and S. Zanella-Béguelin. ‘HACLxN: Verified Generic SIMD Crypto (for all your favourite platforms)’. In: *CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security*. Virtual Event, United States, Nov. 2020. URL: <https://hal.inria.fr/hal-03154275> (cit. on p. 9).
- [48] J. Protzenko, B. Parno, A. Fromherz, C. Hawblitzel, M. Polubelova, K. Bhargavan, B. Beurdouche, J. Choi, A. Delignat-Lavaud, C. Fournet, N. Kulatova, T. Ramanandaro, A. Rastogi, N. Swamy, C. Wintersteiger and S. Zanella-Béguelin. ‘EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider’. In: *SP 2020 - IEEE Symposium on Security and Privacy*. San Francisco / Virtual, United States: IEEE, May 2020, pp. 983–1002. DOI: [10.1109/SP40000.2020.00114](https://doi.org/10.1109/SP40000.2020.00114). URL: <https://hal.inria.fr/hal-03154278> (cit. on p. 9).
- [49] J. Protzenko, J.-K. Zinzindohoué, A. Rastogi, T. Ramanandaro, P. Wang, S. Zanella-Béguelin, A. Delignat-Lavaud, C. Hritcu, K. Bhargavan, C. Fournet and N. Swamy. ‘Verified Low-Level Programming Embedded in F*’. In: *PACMPL* 1.ICFP (Sept. 2017), 17:1–17:29. DOI: [10.1145/3110261](https://doi.org/10.1145/3110261). URL: <http://arxiv.org/abs/1703.00053> (cit. on pp. 8, 9).
- [50] T. Ramanandaro, A. Delignat-Lavaud, C. Fournet, N. Swamy, T. Chajed, N. Kobeissi and J. Protzenko. ‘EverParse: Verified Secure Zero-Copy Parsers for Authenticated Message Formats’. In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. Ed. by N. Heninger and P. Traynor. USENIX Association, 2019, pp. 1465–1482. URL: <https://www.usenix.org/conference/usenixsecurity19/presentation/delignat-lavaud> (cit. on p. 8).
- [51] N. Swamy, C. Fournet, A. Rastogi, K. Bhargavan, J. Chen, P.-Y. Strub and G. M. Bierman. ‘Gradual typing embedded securely in JavaScript’. In: *41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. 2014, pp. 425–438. URL: <http://prosecco.inria.fr/personal/karthik/pubs/tsstar-popl14.pdf> (cit. on p. 10).

- [52] T. Wallez, J. Protzenko, B. Beurdouche and K. Bhargavan. ‘TreeSync: Authenticated Group Management for Messaging Layer Security’. In: *USENIX Security ’23*. Anaheim, United States, Aug. 2023. URL: <https://hal.science/hal-04255953> (cit. on p. 18).