RESEARCH CENTRE

Inria Paris Centre

2023
ACTIVITY REPORT

Project-Team

PROSECCO

Programming securely with cryptography

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Security and Confidentiality

Contents

Project-Team PROSECCO

Creation of the Project-Team: 2012 July 01

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

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

A2.4. – Formal method for verification, reliability, certification

A2.4.2. – Model-checking

A2.4.3. – Proofs

A2.5. – Software engineering

A4. – Security and privacy

A4.3. – Cryptography

A4.3.3. – Cryptographic protocols

A4.5. – Formal methods for security

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

1 Team members, visitors, external collaborators

Research Scientists

- Bruno Blanchet [Team leader, INRIA, Senior Researcher, from May 2023, HDR]
- Karthikeyan Bhargavan [Team leader, INRIA, Senior Researcher, until Apr 2023, HDR]
- Bruno Blanchet [INRIA, Senior Researcher, until Apr 2023, HDR]
- Vincent Cheval [INRIA, Researcher, until Aug 2023]
- Aymeric Fromherz [INRIA, ISFP]
- Adrien Koutsos [INRIA, Researcher]
- Denis Merigoux [INRIA, Starting Research Position]
- Kristina Sojakova [INRIA, Starting Research Position, until Jun 2023]

Post-Doctoral Fellows

- Lucas Franceschino [INRIA, Post-Doctoral Fellow, until May 2023]
- Charlie Jacomme [INRIA, Post-Doctoral Fellow, until Oct 2023]

PhD Students

- Alain Delaët–Tixeuil [INRIA]
- Son Ho [INRIA]
- Théo Laurent [INRIA]
- Antonin Reitz [INRIA]
- Justine Sauvage [INRIA]
- Théo Vignon [ENS PARIS-SACLAY, from Sep 2023]
- Théophile Wallez [INRIA]

Technical Staff

- Sidney Congard [INRIA, Engineer, until Mar 2023]
- Lucas Franceschino [INRIA, Engineer, from Jun 2023]
- Louis Gesbert [INRIA, Engineer]
- Paul-Nicolas Madelaine [INRIA, Engineer]

Interns and Apprentices

- Justine Banuls [INRIA, Intern, until Jun 2023]
- Rémy Citerin [ENS Paris, from Feb 2023 until Aug 2023]

Administrative Assistants

- Christelle Guiziou [INRIA]
- Christelle Rosello [INRIA, from Apr 2023]

Visiting Scientist

- Marie Alauzen [FONDATION INRIA, from Aug 2023 until Sep 2023]

External Collaborators

- Benjamin Beurdouche [MOZILLA]
- Mathieu Durero [DGFIP, from Jul 2023]
- Caroline Flori [DGFIP, from Jul 2023]
- Caroline Fontaine [CNRS, from Oct 2023]
- Damian Poddebniak [Cryspen]
- Jonathan Protzenko [MICROSOFT RESEARCH, from Feb 2023]

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 HACL*, 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 [42].

We also design two frameworks for the analysis of Rust programs (hacspec 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 [68]; it has motivated a serious research effort on the formal analysis of cryptographic protocols, starting with [59] 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* [52]. 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 [49, 37], correspondences (including authentication) [50], and observational equivalences [51]. 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 [65], JFK [38], and Web Services Security [48], 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 [72], 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# [47]. 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], [39, 67], 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 [71]. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest [45] (a larger collaboration with Microsoft Research). This includes a verified implementation of TLS 1.2 and 1.3 [46] and of the underlying cryptographic primitives [10]. More recently, we have built a new symbolic protocol verification framework in F* called DY* [44] 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 [65] 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) [40] 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 [60] 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 [53]. 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], [39] 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 [71]. 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 [46] and of the underlying cryptographic primitives [10], [70, 69].

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 [62], 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 *app*s 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 [58] and TS* [73], 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 [64]. 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 [57]. Catala has been tested on the real-world French housing benefits [14, 29] 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 our 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

- Denis Merigoux won an honorary mention at the ERCIM Cor Baayen Early Career Award 2023.
- Internet Defense Prize at USENIX Security '23 [26] .
- The first paper on ProVerif, published at CSFW'01 [49], won a test-of-time award at CSF'23.
- Distinguished Paper Awards at CSF'23 [21] and USENIX Security'23 [26, 19, 22] .

7 New software, platforms, open data

7.1 New software

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.

News of the Year: Development of additional core libraries for Steel. Development of a verified key-value store, FastVer2, using the framework.

Publications: hal-04104143, hal-02936273, hal-03466397, hal-03626859

Contact: Aymeric Fromherz

Participant: Aymeric Fromherz

7.1.3 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 support for SIMD vectorization, and implemented and verified a generic streaming API for block-based algorithms. The instantiation of this streaming API for hashes (SHA-2, Blake2) was integrated into CPython.

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

7.1.4 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 developed the tooling around DY* to tackle more complex protocols, and did case studies using this new tooling.

In the past, message formats were written by hand in protocol analysis done with DY*, a process that is both tedious and error-prone. We developed Comparse, a tool to automatically generate messages formats in F*. The generated message formats are usable in DY*, and correspond to the ones defined in protocol specifications.

Cryptographic protocols are often analyzed in isolation. However, they are typically deployed within a stack of protocols, where each layer relies on the security guarantees provided by the protocol layer below it, and in turn provides its own security functionality to the layer above. We extended DY* with a new methodology that allows analysts to modularly analyze each layer in a way that compose to provide security for a protocol stack.

We used DY* to study MLS, a novel secure group messaging protocol. Most cryptographic protocols within the reach of formal analysis are between a fixed number of participants (often two) that exchange a fixed number of messages. MLS gives new challenges for analysts, because it supports group members joining and leaving the group over time, and group members can exchange an unbounded number of messages. We used DY* to study TreeSync, a sub-protocol of MLS that specifies the shared group state, defines group management operations, and ensures consistency, integrity, and authentication for the group state across all members.

URL: `https://reprosec.org/`

Publications: hal-03178425, hal-03540403, hal-03540824, hal-04255953, hal-04310972

Contact: Karthikeyan Bhargavan

7.1.5 Hacspec

Keywords: Specification language, Rust

Functional Description: Hacspec is a domain specific language embedded inside Rust geared towards cryptographic specifications. It allows easier communication between formal methods experts and cryptographers that write their implementations in Rust. Hacspec compiles to various proof backends including F* and Coq.

News of the Year: Hacspec was re-implemented entirely and renamed as Hax (`https://github.com/hacspec/hax`).

Hacspec was aiming at being a specification language for crypto primitives in a DSL embedded in Rust. Hax extends the scope of the project, targeting a large subset of Rust, and translating it to various formal backends (such as Coq, F*, EasyCrypt or ProVerif).

We now call Hacspec the functional subset of Rust that can be used, together with a Hacspec standard library, to write succinct, executable, and verifiable specifications in Rust. These specifications can be translated into formal languages with hax.

URL: `https://hacspec.github.io/`

Publication: hal-03176482

Contact: Karthikeyan Bhargavan

Partners: Concordium Blockchain Research Center, Aarhus University, Denmark, Université de Porto

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 expanded the subset of Rust supported by Aeneas in many directions, for instance by adding better support for arrays, slices, binary operations and, more importantly, loops. We also added support for several backends: in addition to the original F* backend, it is now possible to generate Coq, Lean and HOL4 code. For the particular cases of HOL4 and Lean, we implemented an encoding which allows to define partial functions, and we provide basic proof automation for the proofs.

URL: `https://github.com/AeneasVerif/aeneas`

Publication: hal-03931572

Contact: Son Ho

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 expanded the subset supported by Charon by adding better support for arrays and slices, const generics, and function pointers and closures. We also performed minor improvements, for instance in the control flow graph reconstruction to generate more idiomatic code. Charon now also prints nice debugging messages in case of errors, in particular when it encounters unsupported features, which pinpoint the location and the cause of the error to the user, so that they can update their code or file an issue.

URL: `https://github.com/AeneasVerif/charon`

Publication: hal-03931572

Contact: Son Ho

7.1.8 mlang

Name: Mlang

Keywords: Compilers, Legality

Functional Description: 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). This algorithm relies on a legacy custom language and compiler originally designed in 1990, which unlike French wine, did not age well with time. Owing to the shortcomings of the input 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. Competence loss and aging code also mean that the system does not benefit from any modern compiler techniques that would increase confidence in the implementation. We overhaul this infrastructure and present Mlang, an open-source compiler toolchain whose goal is to replace the existing infrastructure. Mlang is based on a reverse-engineered formalization of the DGFiP's system, and has been thoroughly validated against the private DGFiP test suite. As such, Mlang has a formal semantics, eliminates previous handwritten workarounds in C, compiles to modern languages (Python), and enables a variety of instrumentations, providing deep insights about the essence of French income tax computation. The DGFiP is now officially transitioning to Mlang for their production system.

News of the Year: In 2023, Mlang is still being developed at DGFiP and tested for use in production. The income tax computation software compiled with Mlang now reproduces 100% of the behavior of the old tax computation software compiled with the legacy compiler. Extensions of the M language to soft-code rather than hard-code domain-specific terminology have been developed, and plans to add function calls are underway to finally get rid of the legacy C codebase.

Publications: hal-02320347, hal-03002266

Contact: Denis Merigoux

Participant: Denis Merigoux

Partner: Direction Générale des Finances Publiques (DGFiP)

7.1.9 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 2023, the development of Catala has sped up through the work of research engineer Louis Gesbert. The most notable addition is a module system that allows for real code modularization and separate compilation. The language has also gained its first industrial user, the DGFiP, with the start of a project to rewrite the income tax computation algorithm : `https://gitlab.adullact.net/dgfip/ir-catala`.

URL: `https://catala-lang.org/en`

Publications: hal-03712130, hal-03781578, hal-03128248, hal-03159939, hal-02936606, hal-03869335

Contact: Denis Merigoux

Participants: Denis Merigoux, Louis Gesbert, Aymeric Fromherz, Alain Delaet-tixeuil, Raphael Monat

Partner: Université Panthéon-Sorbonne

7.1.10 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.

News of the Year: We released ProVerif version 2.05. The main novelties are that constraints are allowed in assumptions of queries, lemmas, axioms, and restrictions, and that attacker, table, mess, and user-defined predicates are allowed in conclusion of lemmas, axioms, and restrictions. We are working on many extensions (liveness properties, hash consing to save memory, more equational theories, ...). Stay tuned!

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, Marc Sylvestre, Vincent Cheval

7.1.11 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) CryptoVerif can now translate its assumptions into EasyCrypt, so that these assumptions can be proved in EasyCrypt from lower-level or more standard assumptions (by Christian Doczkal, Pierre-Yves Strub, Pierre Boutry, Bruno Blanchet).

2) CryptoVerif can generate implementations of protocols in F*. It also generates F* lemmas for equational properties assumed in CryptoVerif (by Benjamin Lipp and Bruno Blanchet).

3) The oracle front-end is now the main front-end. Processes input in the channel front-end are translated into the oracle front-end (by Charlie Jacomme and Bruno Blanchet).

4) We added notions of secrecy as reachability and secrecy for a bit.

5) We added the diff[.,.] construct, already present in ProVerif, which allows to prove indistinguishability between two similar protocols.

6) We extended the "move" command so that it can move random number generations and assignments upwards in the game.

These changes are included in CryptoVerif version 2.08 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, tel-01112630, hal-01102382, hal-01528752, hal-01575920, hal-01575861, hal-01575923

Contact: Bruno Blanchet

Participants: Bruno Blanchet, David Cadé, Benjamin Lipp, Pierre-Yves Strub, Christian Doczkal, Pierre Boutry

7.1.12 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: Support for higher-order functions and reasoning. Extension of Squirrel cryptographic rules to handle key corruption. New user documentation at: https://squirrel-prover.github.io/documentation/

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

Publications: hal-03981949, hal-03620358, hal-03172119, hal-03264227

Contact: Adrien Koutsos

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

Partners: IRISA, ENS Rennes

7.1.13 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 versions introduces a new logic (ehoare) allowing to bound 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

7.1.14 IPDL

Name: Interactive Probabilistic Dependency Logic

Keywords: Verification, Cryptographic protocol

Functional Description: A process calculus for computationally sound approximate reasoning about distributed cryptographic protocols in a manner both close to the Universal Composability-style simulation paradigm and amenable for formal verification.

Contact: Kristina Sojakova

8 New results

8.1 Verification of security protocols in the symbolic model

Participants: Vincent Cheval, Charlie Jacomme.

Session equivalence TAMARIN and PROVERIF are both able to verify equivalence properties but are intuitively limited to the fine-grained *diff-equivalence*, that is often not well suited for privacy-type properties such as anonymity and unlinkability. This limitation has always been a hard hurdle to overcome as the technique did not evolve in more than 15 years. In a paper at CSF'23 [21], we (Vincent Cheval, with Itsaka Rakotonirina) finally introduced in PROVERIF a new proof technique for verifying equivalence in a general framework. Our technique is based on the notion of *session decomposition*, inspired from the symmetry reductions in [56]. We applied our results on several protocols such as LAK, PACE, a simplified TLS, . . .

Advanced primitive modelings Real life implementations of cryptographic primitives have many behaviors and weaknesses that are not captured by classical symbolic modelings. To bridge this gap, we extensively studied the concretely used hash functions and Authenticated Encryptions with Additional Data (AEADs) and proposed for each of those primitives novel and more faithful models leading to two publications at USENIX Security'23, [19] by Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer and [22] by Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. Those new models were notably deployed in PROVERIF and TAMARIN, and were used to analyze several new protocols for which new subtle weaknesses were discovered.

Applications We performed several major case studies using the new features introduced in PROVERIF. In EuroS&P'23 [20], we (Vincent Cheval, with José Moreira and Mark Ryan) provided the first formal verification of two transparency protocols with a precise model of the Merkle tree data structure: transparent decryption (sometimes called accountable decryption), and certificate transparency. In CSF'23 [18], we (Vincent Cheval, with Véronique Cortier and Alexandre Debant) proved the end-to-end verifiability of multiple electronic voting systems: Helios, Belenios, CHVote and SwissPost. All these proofs relied on a new complete characterization of end-to-end verifiability, a new generic election framework in which voting protocols can be expressed, and a library of lemmas for the automatic proof of end-to-end verifiability.

In USENIX Security'23 [25], we (Charlie Jacomme, with Elise Klein, Steve Kremer, and Maïwenn Racouchot) verified an IETF draft for LAKE EDHOC using our platform SAPIC+ [55]. This platform allows us to leverage the complementary strengths of PROVERIF, TAMARIN, and DEEPSEC from a single model, by translating that model to inputs of the various tools. The analysis was carried out with a fine-grained modeling of possible threat models and compromise, and lead to the discovery of multiple weaknesses, with the proposal of fixes that were integrated into new versions of the draft.

In USENIX Security'23 [23], we (Charlie Jacomme, with Cas Cremers and Aurora Naska) verified the session management layer of the Signal application, Sesame, using TAMARIN. This lead to the discovery that guarantees formally proved on the lower layers protocol are not preserved when considering the full application. We were able to experimentally demonstrate the identified weaknesses, and we proposed and verified simple fixes to the protocol.

Protocols with probabilistic choice In general, symbolic models are purely non-deterministic (or possibilistic). For instance, random numbers are abstracted and are assumed to be impossible to guess by the attacker. While this is generally sensible, it is not always possible to eliminate probabilities altogether as some protocols specifically rely on non-negligible choices in their specification. In [54], we (Vincent Cheval, with Raphaëlle Crubillé and Steve Kremer) considered a framework for symbolic protocol analysis with probabilistic choice. We showed the relations between possibilistic and probabilistic equivalences and designed a decision procedure for probabilistic trace equivalence. This procedure has been implemented in DEEPSEC. In 2023, we published a long version of this paper in the Journal of Computer Security [12].

8.2 Verification of security protocols in the computational model

Participants: Karthikeyan Bhargavan, Bruno Blanchet, Charlie Jacomme, Adrien Koutsos, Justine Sauvage, Kristina Sojakova, Théo Vignon.

CryptoVerif We (Bruno Blanchet) continued the development of CRYPTOVERIF, adding new game transformations in order to deal with the dynamic compromise of keys, which allowed us to complete missing proofs of forward secrecy in major previous case studies (TLS 1.3 [3] and WireGuard [66]). We also added game transformations that guess values, a step often used in cryptographic proofs [31, 30]. This work is accepted at CSF'24.

We (Bruno Blanchet and Charlie Jacomme) showed that CRYPTOVERIF is sound against quantum adversaries [33].

We (Karthikeyan Bhargavan, Bruno Blanchet, with Benjamin Lipp) developed a translation from CRYPTOVERIF to F⋆, 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⋆ lemmas for equations used as assumptions in CRYPTOVERIF; these lemmas are then proved in F⋆. We (Bruno Blanchet, with Pierre Boutry, Christian Dockzal, Benjamin Grégoire, and Pierre-Yves Strub) also developed a translation from the security assumptions used in CRYPTOVERIF to EASYCRYPT [32]. Indeed, the security assumptions in CRYPTOVERIF are often stated in a way that differs from the usual cryptographic assumptions. This translation allows us to prove the CRYPTOVERIF assumptions from more standard or lower-level assumptions in EASYCRYPT. This work is accepted at CSF'24. All these developments are included in CRYPTOVERIF 2.08.

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

At LICS'23 [17], we (Adrien Koutsos, with David Baelde and Joseph Lallemand) proposed a complete re-design of SQUIRREL theoretical foundations, which is less ad hoc, simpler, more general, and supports higher-order reasoning. Additionally, this new logical presentation was used to allow SQUIRREL cryptographic rules to support key corruption.

We also have work in progress on techniques to allow users to easily add new cryptographic primitives and assumptions to SQUIRREL without modifying the tool itself (PhD thesis of Justine Sauvage).

Currently, SQUIRREL only supports the verification of cryptographic protocols in the asymptotic security model, which limits SQUIRREL reasoning capabilities (some cryptographic arguments are out-of-scope, e.g. polynomial hybrid arguments) and expressivity (SQUIRREL cannot give a precise bound on the probability of breaking protocol security). We are working on overcoming these limitations by adapting SQUIRREL to a concrete security setting (PhD thesis of Théo Vignon).

EasyCrypt EASYCRYPT is a theorem prover designed for reasoning about properties of probabilistic computations with adversarial code, using imperative program logics implemented on top of a higher-order ambient logic. Its main application is the construction and verification of game-based cryptographic primitives.

While EASYCRYPT is mainly developed outside of Prosecco at PQShield and the SPLiTS Inria team at Sophia-Antipolis, we (Adrien Koutsos, with Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, and Pierre-Yves Strub) have presented in CCS'21 [41] an extension of EASYCRYPT with a new program logic allowing us to reason about the computational worst-case complexity of adversarial programs. This logic can be used to prove precise relationships between the complexity of cryptographic adversaries and their success probability, allowing us to obtain fully mechanized cryptographic reductions. We showcased our approach through a new formalization of the Universal Composability framework. In 2023, we published a journal version this work at ACM ToPS [11]).

IPDL In [24], we (Kristina Sojakova, with Joshua Gancher, Xiong Fan, Elaine Shi, and Greg Morrisett) present a novel technique for proving UC-security of a class of cryptographic protocols, based on a equational calculus at the protocol level, named IPDL.

8.3 High-Assurance High-Performance Crypto

Participants: Karthikeyan Bhargavan, Aymeric Fromherz, Son Ho.

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.

In ICFP 2023, we (Son Ho, Aymeric Fromherz, and Jonathan Protzenko) published a paper [13], on using advanced features of F⋆, such as verified meta-programming, to build generic streaming APIs for cryptographic constructions in HACL⋆.

We (Karthikeyan Bhargavan, with Daniel de Almeida Braga, Natalia Kulatova, Mohammed Sabt, Pierre-Alain Fouque) also used HACL⋆ to help improve the security of WPA3 implementations that were vulnerable to side-channel attacks, resulting in a paper at EuroS&P 2023 [15].

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

Participants: Karthikeyan Bhargavan, 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, Jonathan Protzenko, Benjamin Beurdouche, and Karthikeyan Bhargavan) used DY⋆ to formally model, implement, and analyze the Messaging Layer Security (MLS) standard. Our work uncovered vulnerabilities [43] and contributed to the published standard (our PhD student Benjamin Beurdouche is a co-author of the RFC standard). Our research was published at USENIX Security'23 [26], where it won a Distinguished Paper Award and the 2023 Internet Defense Prize.

We (Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan) also worked on formally analyzing the security of message formats used in cryptographic protocols and applied it to several protocols including TLS 1.3, MLS, and cTLS. We embedded this format reasoning methodology in DY⋆ and published a paper in ACM CCS 2023 [27].

8.5 Extensions to F*

Participants: Aymeric Fromherz, Antonin Reitz.

Since 2010, our group contributes to the design, implementation, and application of the F* programming language and verification work.

Since 2020, we work on a framework, called Steel [61], based on concurrent separation logic, for developing and proving concurrent programs embedded in F*. Steel proposes a new formalism of Hoare quintuples which involve both separation logic and first-order logic, and enable an efficient verification condition generation and proof discharge using a combination of tactics and SMT solving.

In collaboration with Microsoft Research, we (Aymeric Fromherz) published a paper at CPP 2023 [16] about a verified, low-level, concurrent implementation of a key-value store implemented and verified in Steel.

Antonin Reitz also gave a presentation at the Annual Meeting of the WG "Formal Methods for Security" about ongoing work on StarMalloc, a verified, hardened, concurrent memory allocator that he develops in Steel as part of his PhD in collaboration with Aymeric Fromherz.

8.6 Formalizing and Implementing Law

Participants: Justine Banuls, Alain Delaët-Tixeuil, Aymeric Fromherz, Louis Gesbert, Denis Merigoux.

Since 2021, we develop a new domain-specific language, Catala, targeted specifically for legal expert systems. This language has been built in close collaboration with lawyers, and advertised to that community with a number of legal-oriented papers [64, 63]. On the formal methods side, the simple and clean design of the Catala semantics [8] allows for extension into a proper proof platform for the law [57].

Catala has been tested on the real-world French housing benefits [14, 29] and is currently experimented for use at DGFiP.

8.7 Verification of Rust programs: Aeneas and hacspec

Participants: Son Ho, Aymeric Fromherz, Karthikeyan Bhargavan, Lucas Franceschino.

In collaboration with Jonathan Protzenko and Aymeric Fromherz, Son Ho developed on a new verification toolchain for Rust programs called Aeneas. 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. Aeneas was presented at ICFP 2022 [62] and RW 23 [35].

Karthikeyan Bhargavan and Lucas Franceschino worked on the development of hacspec, a purely functional subset of Rust that is used to specify and verify cryptographic algorithms. Specifications in hacspec can be compiled to F* and Coq, and an EasyCrypt backend is being developed. The hacspec framework has been used to specify a large set of cryptographic algorithms and is being used as part of new standardization efforts.

9 Bilateral contracts and grants with industry

9.1 Bilateral grants with industry

Evolution, Semantics, and Engineering of the F* Verification System

Participants: Aymeric Fromherz, Karthikeyan Bhargavan, Théo Laurent.

- Grant from Nomadic Labs - Inria
- PIs: Aymeric Fromherz (earlier, Catalin Hritcu, Exequiel Rivas)
- Duration: March 2019 - April 2023
- Abstract: While the F* verification system shows great promise in practice, many challenging conceptual problems remain to be solved, many of which can directly inform the further evolution and design of the language. Moreover, many engineering challenges remain in order to build a truly usable verification system. This proposal promises to help address this by focusing on the following 5 main topics:

  (1) *Generalizing Dijkstra monads*, i.e., a program verification technique for arbitrary monadic effects; (2) *Relational reasoning in F**: devising scalable verification techniques for properties of multiple program executions (e.g., confidentiality, noninterference) or of multiple programs (e.g., program equivalence); (3) *Making F*'s effect system more flexible*, by supporting tractable forms of effect polymorphism and allowing some of the effects of a computation to be hidden if they do not impact the observable behavior; (4) Working out more of the *F* semantics and metatheory*; (5) Solving the *engineering challenges* of building a usable verification system.

9.2 Other funding

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.

10 Partnerships and cooperations

10.1 International initiatives

10.1.1 Inria associate team not involved in an IIL or an international program

VeriSPro

Participants: Karthikeyan Bhargavan, Aymeric Fromherz, Théophile Wallez.

Title: Verifying Security Properties of Group Messaging Protocols

Duration: From 2022

Coordinator: Abhishek Bichhawat (abhishek.b@iitgn.ac.in)

Partner: IIT Gandhinagar (Inde)

Inria contact: Aymeric Fromherz

Summary: Modern instant messaging systems allow multiple parties to communicate with each other in pairs and in groups. The security of these conversations depends on complex cryptographic protocols with subtle security guarantees. These protocols allow the addition and deletion of members, as well as the exchange of confidential and authentic messages between (current) members. It is difficult to be confident that these protocols and their implementations are correct. Formal mechanized security analysis of protocols has been widely accepted as a necessary step for designing robust cryptographic protocols but has not been previously used to analyze group messaging. This proposal focuses on formally verifying security properties (like forward secrecy and post-compromise security) of group messaging protocols. We propose to extend DY*, a symbolic verification tool, to build a generic formal framework to model group-messaging protocols. We will model various group messaging protocols and verify different security properties ranging from authentication of peers in a group to the confidentiality of messages exchanged between the peers. Finally, we will use our generic framework to empirically compare of performance of those protocols.

10.2 European initiatives

10.2.1 Horizon Europe

CRYSPEN CRYSPEN project on cordis.europa.eu

Participants: Karthikeyan Bhargavan.

Title: Custom Cryptographic Solutions with Formal Security Guarantees

Duration: From April 1, 2022 to April 30, 2023

Partners: Inria, France

Coordinator: Karthikeyan Bhargavan

Summary: Modern web applications routinely rely on standardized cryptographic protocols and algorithms to protect sensitive user data. Furthermore, with the advent of blockchains, the imminence of quantum computers, and widespread concerns about privacy in an era of surveillance and machine learning algorithms, enterprises are increasingly turning to sophisticated non-standard cryptographic solutions customized for specific usage scenarios. Unfortunately, cryptographic design and implementation is notoriously error-prone with a long history of design flaws, implementation bugs, and high-profile attacks. This leaves software companies with a difficult choice: every time they deploy a new crypto standard or an innovative cryptographic application that improves the security and privacy of their users, they risk exposing embarrassing flaws in their design or code.

The research results of ERC Circus offer a way out of this conundrum by advocating the use of formal verification to build cryptographic software with machine-checked proofs of security and correctness. A landmark output from this project is HACL*, a verified high-performance cryptographic library which is currently used by mainstream software like Mozilla Firefox, Linux Kernel, Tezos Blockchain, and ElectionGuard. We propose to establish a company (called Cryspen) that will transition the research software developed in ERC Circus towards a production-quality ready-to-use verified cryptographic software stack. In addition, Cryspen will offer a developer-friendly verification framework that can be used to build new custom cryptographic solutions in C, Rust, and JavaScript. The goal of this Proof-of-Concept proposal is to fund the initial technical transfer of research software to Cryspen and the business development of this company. Once this transfer is complete, Cryspen will be able to offer long-term service contracts to existing and new users of HACL*, and offer software contracts to enterprises interested in deploying verified cryptographic software.

10.3 National initiatives

10.3.1 PEPR

PEPR Cybersecurity SVP

Participants: Karthikeyan Bhargavan, Bruno Blanchet (local PI), Vincent Cheval, Sidney Congard, Lucas Franceschino, Aymeric Fromherz, Son Ho, Charlie Jacomme, Adrien Koutsos, Théo Laurent, Antonin Reitz, Justine Sauvage, Kristina Sojakova.

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: Karthikeyan Bhargavan (local PI until April 2023), Bruno Blanchet (local PI from May 2023), Charlie Jacomme.

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 2026

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.

11 Dissemination

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

General chair, scientific chair

- Denis Merigoux co-chaired the Workshop on Programming Languages and the Law 2023.

Member of the organizing committees

- Karthikeyan Bhargavan was a co-organizer for the High-Assurance Cryptographic Software (HACS) workshop 2023.
- Charlie Jacomme was a co-organizer for the GT-MFS (French working group on formal methods for security) 2023.
- Adrien Koutsos is a co-organizer for the GT-MFS 2024.
- Aymeric Fromherz was a co-organizer for the ICFP Programming Contest 2023.

11.1.2 Scientific events: selection

Member of the conference program committees

- Aymeric Fromherz: PC member for JFLA’23, POPL’24, USENIX Security’24.
- Charlie Jacomme: PC member for USENIX Security’24.
- Adrien Koutsos: PC member for CCS’23.
- Denis Merigoux: PC member for CRCL’23, OOPSLA’23, ICAIL’23, JURIX’23, POPL’24.

11.1.3 Journal

Member of the editorial boards

- Karthikeyan Bhargavan: Associate Editor of ACM Transaction on Privacy and Security (TOPS)
- Denis Merigoux: Editor of the Journal of Cross-disciplinary Research in Computational Law (CRCL)

11.1.4 Invited talks

- Denis Merigoux: Invited talk at CRCL’23

11.2 Teaching - Supervision - Juries

11.2.1 Teaching

- Master: Bruno Blanchet, Cryptographic protocols: formal and computational proofs, 27h equivalent TD, master M2 MPRI, université Paris VII
- Master: Adrien Koutsos, Cryptographic protocols: formal and computational proofs, 27h equivalent TD, master M2 MPRI, université Paris VII

11.2.2 Supervision

- 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: Alain Delaët-Tixeuil, Interactive verification for programs deriving from legal specifications, since September 2022, supervised by Sandrine Blazy and Denis Merigoux.
- PhD in progress: Antonin Reitz, A Methodology for Programming and Verifying Secure Systems, since November 2022, supervised by Bruno Blanchet and Aymeric Fromherz.
- PhD in progress: Justine Sauvage, Games and Logic for the Verification of Cryptographic Protocols, since September 2022, supervised by Bruno Blanchet, David Baelde, and Adrien Koutsos.
- PhD in progress: Son Ho, Verification of Rust Programs, since September 2020, supervised by Karthikeyan Bhargavan, Bruno Blanchet, and Jonathan Protzenko.
- PhD in progress: Théophile Wallez, Verification of Cryptographic Protocols, since September 2021, supervised by Karthikeyan Bhargavan, Bruno Blanchet, and Jonathan Protzenko.
- PhD in progress: Théo Laurent, Dependent types and subtyping, since July 2020, supervised by David Delahaye, Bruno Blanchet, and Kenji Maillard.
- M2 internship: Rémy Citerin, Coinduction in F* and application to interaction trees, supervised by Aymeric Fromherz and Théo Laurent.

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. 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 Fransisco / Virtual, United States, 23rd May 2021. URL: https://hal.archives-ouvertes.fr/hal-03172119.

[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. URL: https://hal.inria.fr/hal-01575920.

[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. 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. URL: https://inria.hal.science/hal-03159939.

[9] N. Swamy, C. Hriţcu, 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.

[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.

12.2 Publications of the year

International journals

[11] M. Barbosa, G. Barthe, B. Grégoire, A. Koutsos and P.-Y. Strub. ‘Mechanized Proofs of Adversarial Complexity and Application to Universal Composability: Journal pre-print: full version’. In: *ACM Transactions on Privacy and Security* 26.3 (30th Aug. 2023), pp. 1–34. DOI: 10.1145/3589962. URL: https://inria.hal.science/hal-04048217.

[12] V. Cheval, R. Crubillé and S. Kremer. ‘Symbolic protocol verification with dice: Process equivalences in the presence of probabilities’. In: *Journal of Computer Security* (12th June 2023), pp. 1–38. DOI: 10.3233/JCS-230037. URL: https://inria.hal.science/hal-04179875.

[13] S. Ho, A. Fromherz and J. Protzenko. ‘Modularity, Code Specialization, and Zero-Cost Abstractions for Program Verification’. In: *Proceedings of the ACM on Programming Languages* 7.ICFP (31st Aug. 2023), pp. 385–416. DOI: 10.1145/3607844. URL: https://hal.science/hal-04301439.

[14] 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.

International peer-reviewed conferences

[15] D. de Almeida Braga, N. Kulatova, M. Sabt, P.-A. Fouque and K. Bhargavan. ‘From Dragondoom to Dragonstar: Side-channel Attacks and Formally Verified Implementation of WPA3 Dragonfly Handshake’. In: EuroS&P 2023 - IEEE 8th European Symposium on Security and Privacy. Delft, Netherlands: IEEE, 3rd July 2023, pp. 707–723. DOI: 10.1109/EuroSP57164.2023.00048. URL: https://hal.science/hal-04175322.

[16] A. Arasu, T. Ramananandro, A. Rastogi, N. Swamy, A. Fromherz, K. Hietala, B. Parno and R. Ramamurthy. ‘FastVer2: A Provably Correct Monitor for Concurrent, Key-Value Stores’. In: CPP ’23 - 12th ACM SIGPLAN International Conference on Certified Programs and Proofs. Boston (MA), United States: ACM, 16th Jan. 2023, pp. 30–46. DOI: 10.1145/3573105.3575687. URL: https://inria.hal.science/hal-04104143.

[17] D. Baelde, A. Koutsos and J. Lallemand. ‘A Higher-Order Indistinguishability Logic for Cryptographic Reasoning’. In: LICS. Boston, United States: IEEE, 26th June 2023, pp. 1–13. DOI: 10.1109/LICS56636.2023.10175781. URL: https://inria.hal.science/hal-03981949.

[18] V. Cheval, V. Cortier and A. Debant. 'Election Verifiability with ProVerif'. In: CSF 2023 - 36th IEEE Computer Security Foundations Symposium. Dubrovnik, Croatia, 9th July 2023. URL: https://inria.hal.science/hal-04177268.

[19] ***Best Paper***
V. Cheval, C. Cremers, A. Dax, L. Hirschi, C. Jacomme and S. Kremer. 'Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses'. In: 32nd USENIX Security Symposium. Anaheim, United States, 2023. URL: https://hal.science/hal-03795715.

[20] V. Cheval, J. Moreira and M. Ryan. 'Automatic verification of transparency protocols'. In: *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P). Delft, Netherlands: IEEE, 3rd July 2023. DOI: 10.1109/EuroSP57164.2023.00016. URL: https://inria.hal.science/hal-04219234.

[21] ***Best Paper***
V. Cheval and I. Rakotonirina. 'Indistinguishability Beyond Diff-Equivalence in ProVerif'. In: *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*. 2023 IEEE 36th Computer Security Foundations Symposium (CSF). Dubrovnik, Croatia: IEEE, 10th July 2023, pp. 184–199. DOI: 10.1109/CSF57540.2023.00036. URL: https://inria.hal.science/hal-04219230.

[22] ***Best Paper***
C. Cremers, A. Dax, C. Jacomme and M. Zhao. 'Automated Analysis of Protocols that use Authenticated Encryption:How Subtle AEAD Differences can impact Protocol Security'. In: 32nd USENIX Security Symposium 2023. Anaheim, United States, 9th Aug. 2023. URL: https://inria.hal.science/hal-04126116.

[23] C. Cremers, C. Jacomme and A. Naska. 'Formal Analysis of Session-Handling in Secure Messaging: Lifting Security from Sessions to Conversations'. In: USENIX Security 2023 - 32nd USENIX Security Symposium. Anaheim, United States, 2023. URL: https://hal.science/hal-03996689.

[24] J. Gancher, K. Sojakova, X. Fan, E. Shi and G. Morrisett. 'A Core Calculus for Equational Proofs of Cryptographic Protocols'. In: POPL 2023 - 50th ACM SIGPLAN Symposium on Principles of Programming Languages. Boston, United States, 15th Jan. 2023. DOI: 10.1145/3571223. URL: https://inria.hal.science/hal-03917005.

[25] C. Jacomme, E. Klein, S. Kremer and M. Racouchot. 'A comprehensive, formal and automated analysis of the EDHOC protocol'. In: USENIX Security '23 - 32nd USENIX Security Symposium. Anaheim, CA, United States, 9th Aug. 2023. URL: https://inria.hal.science/hal-03810102.

[26] ***Best Paper***
T. Wallez, J. Protzenko, B. Beurdouche and K. Bhargavan. 'TreeSync: Authenticated Group Management for Messaging Layer Security'. In: USENIX Security '23. Anaheim, United States, 9th Aug. 2023. URL: https://hal.science/hal-04255953.

[27] T. Wallez, J. Protzenko and K. Bhargavan. 'Comparse: Provably Secure Formats for Cryptographic Protocols'. In: CCS '23: ACM SIGSAC Conference on Computer and Communications Security. Copenhagen, Denmark: ACM, 26th Nov. 2023, pp. 564–578. DOI: 10.1145/3576915.3623201. URL: https://hal.science/hal-04310972.

National peer-reviewed Conferences

[28] T. Wallez. 'Vérification symbolique de protocoles cryptographiques en F*: application au sous-protocole TreeSync de MLS'. In: *Journées Francophones des Langages Applicatifs*. JFLA 2023 - 34èmes Journées Francophones des Langages Applicatifs. Praz-sur-Arly, France, 16th Jan. 2023, pp. 243–263. URL: https://inria.hal.science/hal-03936726.

Conferences without proceedings

[29] 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.

Reports & preprints

[30] B. Blanchet. *CryptoVerif: a Computationally-Sound Security Protocol Verifier (Initial Version with Communications on Channels)*. RR-9525. Inria Paris, 17th Oct. 2023, p. 166. URL: https://inria.hal.science/hal-04246199.

[31] B. Blanchet. *Dealing with Dynamic Key Compromise in CryptoVerif.* 6th Nov. 2023. URL: https://inria.hal.science/hal-04271666.

[32] 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.

[33] B. Blanchet and C. Jacomme. *CryptoVerif: a Computationally-Sound Security Protocol Verifier*. RR-9526. Inria, 23rd Oct. 2023, p. 194. URL: https://inria.hal.science/hal-04253820.

[34] V. Cheval, R. Crubillé and S. Kremer. *Symbolic protocol verification with dice: process equivalences in the presence of probabilities (extended version)*. 30th May 2023. URL: https://inria.hal.science/hal-03683907.

[35] S. Ho, J. Protzenko and A. Fromherz. *Aeneas: Rust Verification by Functional Translation*. Inria Paris, 23rd Apr. 2023. URL: https://hal.science/hal-04136056.

[36] T. Laurent, M. Lennon-Bertrand and K. Maillard. *Definitional Functoriality for Dependent (Sub)Types*. 23rd Oct. 2023. URL: https://hal.science/hal-04160858.

12.3 Cited publications

[37] 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.

[38] 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.

[39] 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. URL: https://www.fstar-lang.org/papers/dm4free/.

[40] G. Bana, P. Adaõ and H. Sakurada. 'Computationally Complete Symbolic Adversary and Computationally Sound Veri?cation of Security Protocols (in Japanese)'. In: *Proceedings of The 30th Symposium on Cryptography and Information Security*. CD-ROM (4D1-3), Jan. 2013.

[41] M. Barbosa, G. Barthe, B. Grégoire, A. Koutsos and P. Strub. 'Mechanized Proofs of Adversarial Complexity and Application to Universal Composability'. In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Y. Kim, J. Kim, G. Vigna and E. Shi. ACM, 2021, pp. 2541–2563. DOI: 10.1145/3460120.3484548. URL: https://doi.org/10.1145/3460120.3484548.

[42] E. D. Berger. 'Software needs seatbelts and airbags'. In: *Communications of the ACM* 55.9 (2012), pp. 48–53.

[43] K. Bhargavan, B. Beurdouche and P. Naldurg. *Formal Models and Verified Protocols for Group Messaging: Attacks and Proofs for IETF MLS*. Research Report. Inria Paris, Dec. 2019. URL: https://inria.hal.science/hal-02425229.

[44] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseyni, 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.

[45] 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.

[46] 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.

[47] K. Bhargavan, C. Fournet, R. Corin and E. Zalinescu. 'Verified Cryptographic Implementations for TLS'. In: *ACM Transactions Inf. Syst. Secur.* 15.1 (Mar. 2012), 3:1–3:32. DOI: 10.1145/2133375.2133378. URL: http://doi.acm.org/10.1145/2133375.2133378.

[48] 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.

[49] B. Blanchet. 'An Efficient Cryptographic Protocol Verifier Based on Prolog Rules'. In: *14th IEEE Computer Security Foundations Workshop (CSFW'01)*. 2001, pp. 82–96.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] V. Cheval, R. Crubillé and S. Kremer. 'Symbolic protocol verification with dice: process equivalences in the presence of probabilities'. In: *CSF'22 - 35th IEEE Computer Security Foundations Symposium*. Haifa, Israel, Aug. 2022, pp. 319–334. URL: https://hal.inria.fr/hal-03700492.

[55] V. Cheval, C. Jacomme, S. Kremer and R. Künnemann. 'Sapic+ : protocol verifiers of the world, unite!' In: *USENIX 2022 - 31st USENIX Security Symposium*. Boston, United States, Aug. 2022. URL: https://hal.inria.fr/hal-03693843.

[56] V. Cheval, S. Kremer and I. Rakotonirina. 'Exploiting Symmetries When Proving Equivalence Properties for Security Protocols'. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by L. Cavallaro, J. Kinder, X. Wang and J. Katz. ACM, 2019, pp. 905–922. DOI: 10.1145/3319535.3354260.

[57] 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.

[58] A. Delignat-Lavaud, K. Bhargavan and S. Maffeis. '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.

[59] 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.

[60] C. Fournet, M. Kohlweiss and P.-Y. Strub. 'Modular Code-Based Cryptographic Verification'. In: *ACM Conference on Computer and Communications Security*. 2011.

[61] A. Fromherz, A. Rastogi, N. Swamy, S. Gibson, G. Martínez, D. Merigoux and T. Ramananandro. 'Steel: proof-oriented programming in a dependently typed concurrent separation logic'. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (Aug. 2021), pp. 1–30. DOI: 10.1145/3473590. URL: https://hal.inria.fr/hal-03466397.

[62] 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. URL: https://hal.science/hal-03931572.

[63] 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. URL: https://hal.inria.fr/hal-02936606.

[64] 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.

[65] 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. URL: https://hal.inria.fr/hal-01575923.

[66] B. Lipp, B. Blanchet and K. Bhargavan. 'A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol'. In: *IEEE European Symposium on Security and Privacy (EuroS&P'19)*. Stockholm, Sweden: IEEE Computer Society, June 2019, pp. 231–246. URL: https://hal.inria.fr/hal-02100345/document.

[67] 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. URL: https://arxiv.org/abs/1903.01237.

[68] 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.

[69] 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.

[70] J. Protzenko, B. Parno, A. Fromherz, C. Hawblitzel, M. Polubelova, K. Bhargavan, B. Beurdouche, J. Choi, A. Delignat-Lavaud, C. Fournet, N. Kulatova, T. Ramananandro, 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. URL: https://hal.inria.fr/hal-03154278.

[71] J. Protzenko, J.-K. Zinzindohoué, A. Rastogi, T. Ramananandro, 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. URL: http://arxiv.org/abs/1703.00053.

[72] T. Ramananandro, 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.

[73] 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.