RESEARCH CENTRE

Paris

2021
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

B9.10. – Privacy

1 Team members, visitors, external collaborators

Research Scientists

- Karthikeyan Bhargavan [Team leader, Inria, Senior Researcher, HDR]
- Bruno Blanchet [Inria, Senior Researcher, HDR]
- Vincent Cheval [Inria, Researcher]
- Adrien Koutsos [Inria, Researcher]
- Prasad Naldurg [Inria, Advanced Research Position, Jan 2021]
- Exequiel Rivas Gadda [Inria, Starting Research Position, until Feb 2021]
- Kristina Sojakova [Inria, Starting Research Position]

Post-Doctoral Fellow

- Aymeric Fromherz [Inria, from Oct 2021]

PhD Students

- Son Ho [Inria]
- Natalia Kulatova [Inria, until Oct 2021]
- Theo Laurent [Inria]
- Benjamin Lipp [Inria]
- Denis Merigoux [Inria]
- Marina Polubelova [Inria, until May 2021]
- Theophile Wallez [Inria, from Apr 2021]

Technical Staff

- Florian Groult [Inria, Engineer, until Oct 2021]
- Theophile Wallez [Inria, Engineer, until Mar 2021]

Interns and Apprentices

- Alain Delaet–Tixeuil [École Normale Supérieure de Lyon, from Oct 2021]
- Paul Nicolas Madelaine [Inria, from Apr 2021 until Aug 2021]
- Antonin Reitz [Inria, from Oct 2021]
- Justine Sauvage [Inria, from Oct 2021]

Administrative Assistants

- Christelle Guiziou [Inria]
- Mathieu Mourey [Inria, until Oct 2021]
- Scheherazade Rouag [Inria, from Oct 2021]

Visiting Scientist

- Aymeric Fromherz [École Normale Supérieure de Lyon, Sep 2021]

External Collaborators

- Benjamin Beurdouche [Mozilla]
- Adrien Durier [Max Planck Society, until Jun 2021]
- Marina Polubelova [Nomadic Labs, from Jun 2021]
- Jonathan Protzenko [Microsoft Research]

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 five 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. Our current focus is on designing and implementing 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 case of F* in our group is HACL*, a verified cryptographic library, and DY*, a framework for verifying protocol implementations.

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. 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 prover CryptoVerif, 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 symbolic and computational verification, such as computational soundness results that enable computational proofs by symbolic techniques.

Efficient formally secure compilers for tagged architectures We aim to leverage emerging hardware capabilities for fine-grained protection to build the first, efficient secure compilation chains for realistic low-level programming languages (the C language, and Low* a safe subset of C embedded in F* for verification). These compilation chains will provide a secure semantics for all programs and will ensure that high-level abstractions cannot be violated even when interacting with untrusted low-level code. To achieve this level of security without sacrificing efficiency, our secure compilation chains target a tagged architecture, which associates a metadata tag to each word and efficiently propagates and checks tags according to software-defined rules.

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 [57]; it has motivated a serious research effort on the formal analysis of cryptographic protocols, starting with [49] 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* [46]. 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 [43, 29], correspondences (including authentication) [44], and observational equivalences [45]. 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 [38], Signal [53], JFK [30], and Web Services Security [42], 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 [61], 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# [41]. 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. Our current focus is on designing and implementing the programming language F* [63, 33, 55], 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 [60]. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest [39] (a larger collaboration with Microsoft Research). This includes a verified implementation of TLS 1.2 and 1.3 [40] and of the underlying cryptographic primitives [65]. More recently, we have built a new symbolic protocol verification framework in F* called DY* [19] and used it to verify protocols like Signal and ACME [18].

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 [38] and Signal [53] 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 compuatationally complete symbolic adversary (CCSA) [36] 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 [50] 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 [47]. We are currently working on this approach to generate implementations in F*.

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

F* [63, 33] 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 [60]. 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 [40] and of the underlying cryptographic primitives [65, 59, 58].

3.4 Efficient Formally Secure Compilers to a Tagged Architecture

Severe low-level vulnerabilities abound in today's computer systems, allowing cyber-attackers to remotely gain full control. This happens in big part because our programming languages, compilers, and architectures were designed in an era of scarce hardware resources and too often trade off security for efficiency. The semantics of mainstream low-level languages like C is inherently insecure, and even for safer languages, establishing security with respect to a high-level semantics does not guarantee the absence of low-level attacks. Secure compilation using the coarse-grained protection mechanisms provided by mainstream hardware architectures would be too inefficient for most practical scenarios.

We aim to leverage emerging hardware capabilities for fine-grained protection to build the first, efficient secure compilation chains for realistic low-level programming languages (the C language, and Low* a safe subset of C embedded in F* for verification [60]). These compilation chains will provide a secure semantics for all programs and will ensure that high-level abstractions cannot be violated even when interacting with untrusted low-level code. To achieve this level of security without sacrificing efficiency, our secure compilation chains target a tagged architecture [35], which associates a metadata tag to each word and efficiently propagates and checks tags according to software-defined rules. We hope to experimentally evaluate and carefully optimize the efficiency of our secure compilation chains on realistic workloads and standard benchmark suites. We are also using property-based testing and formal verification to provide high confidence that our compilation chains are indeed secure. Formally, we are constructing machine-checked proofs of a new security criterion we call robustly safe compilation, which is defined as the preservation of safety properties even against an adversarial context [32, 31]. This strong criterion complements compiler correctness and ensures that no machine-code attacker can do more harm to securely compiled components than a component already could with respect to a secure source-level semantics.

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 [48] and TS* [62], 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).

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 standardaized at he 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.

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

This year we published 12 peer-reviewed papers, including four papers at IEEE Security and Privacy (S&P) and one paper each at ACM Conference on Computer and Communications Security (CCS), Eurocrypt, and ICFP. We would like to highlight two of these publications.

We published a paper at IEEE S&P [20] that presents major improvements in the protocol verifier ProVerif. First, we extend ProVerif with lemmas, axioms, proofs by induction, natural numbers, and temporal queries. These features not only extend the scope of ProVerif, but can also be used to improve its precision (that is, avoid false attacks) and make it terminate more often. Second, we rework and optimize many of the algorithms used in ProVerif (generation of clauses, resolution, subsumption, etc.), resulting in impressive speed-ups on large examples.

We also presented a new tool, the Squirrel Prover, for the computational verification of security protocols at IEEE S&P [14]. This tool is based upon a meta-logic as well as a proof system for deriving security of protocols with an arbitrary number of sessions. Proofs in our system only deal with high-level, symbolic representations of protocol executions, similar to proofs in the symbolic model, but providing security guarantees at the computational level. We demonstrate the applicability of the Squirrel prover by performing a number of case studies covering a variety of primitives (encryption, DH exponentiation, etc.) and security properties (secrecy, unlinkability, etc.)

7 New software and platforms

7.1 New software

7.1.1 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) Improved the precision of the computation of probabilities, to get better bounds on the probability of success of an attack.

2) Extended the language for specifying assumptions on security primitives: we allow comparisons between array indices used in the "find" construct and we allow tables (instructions "insert" and "get") as a more user-friendly alternative to "find".

3) Extended the language of games, by allowing random number generation (instruction "new") and events that stop the game (instruction "event_abort") in conditions of "find". Also added the instruction "get[unique]" which looks for elements in a table and fails in case several elements satisfy the desired condition, this instruction is translated into "find[unique]" at the beginning of the game transformations. CryptoVerif now proves that the "get[unique]" and "find[unique]" are really unique in the first game, that is, except in cases of negigible probability, there is a single element that satisfies the desired condition.

4) Calls to probability functions are now type-checked.

5) Some improvements in the specifications of Diffie-Hellman assumptions in the standard library of cryptographic primitives.

These changes are included in CryptoVerif version 2.05 available at https://cryptoverif.inria.fr.

URL: `http://cryptoverif.inria.fr/`

Publications: hal-03113251, hal-03471218, 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

7.1.2 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.

Release Contributions: New in 2020: F* continues to evolve and is actively developed on GitHub (https://github.com/FStarLang/FStar). This year, we worked on reworking the stateful core of F* using a new formal foundation called Steel (which was published at ICFP 2020) and on improving the libraries and tooling of F* to support meta-programming, which was heavily used in the EverCrypt and HACLxN projects.

New in 2021: F* now has layered effects which were used in multiple projects, including to implement a symbolic protocol verification framework called DY*.

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

Contact: Catalin Hritcu

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

7.1.3 miTLS

Keywords: Cryptographic protocol, Software Verification

Functional Description: miTLS is a verified reference implementation of the TLS protocol. Our code fully supports its wire formats, ciphersuites, sessions and connections, re-handshakes and resumptions, alerts and errors, and data fragmentation, as prescribed in the RFCs, it interoperates with mainstream web browsers and servers. At the same time, our code is carefully structured to enable its modular, automated verification, from its main API down to computational assumptions on its cryptographic algorithms.

Release Contributions: New in 2020: In 2020, we continued work on the implementation and deployment of TLS 1.3 and the QUIC protocol, which internally uses TLS 1.3. We integrated the new EverCrypt cryptographic provider to miTLS, resulting in significant performance improvements. Finally, we deployed our code within Microsoft MsQuic (https://github.com/microsoft/msquic).

URL: https://github.com/mitls/mitls-fstar

Contact: Karthikeyan Bhargavan

Participants: Alfredo Pironti, Antoine Delignat-Lavaud, Cedric Fournet, Jean-Karim Zinzindohoué, Karthikeyan Bhargavan, Pierre-Yves Strub, Santiago Zanella

7.1.4 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: Vincent Cheval and Bruno Blanchet finished their work on several extensions of ProVerif: 1) support for integer counters, with incrementation and inequality tests, 2) lemmas and axioms to give intermediate results to ProVerif, which it exploits to help proving subsequent queries, by deriving additional information in the Horn clauses that it uses to perform the proofs, 3) proofs by induction on the length of the trace, by giving as lemma the property to prove, but obviously for strictly shorter traces, 4) temporal queries, which allow to order events. The soundness of these features is proved (by hand). Moreover, they optimized many algorithms used in ProVerif (generation of clauses, resolution, subsumption ...) resulting in impressive speedups on large examples. These features are included in ProVerif 2.02pl1 and a paper by Bruno Blanchet, Vincent Cheval, and Véronique Cortier has been published at Security and Privacy 2022.

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

7.1.5 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.

Release Contributions: New in 2020: In 2020, we worked on two major updates to the HACL* library

(1) EverCrypt: We built a new cryptographic provider called EverCrypt that combines verified C code from HACL* with verified Intel assembly code from the Vale projects and integrates them under a verified agile multiplexed API that seamlessly works across multiple platforms. The resulting verified code provides best-in-class performance for popular primitives like AES-GCM and X25519. This work resulted in a paper at IEEE S&P 2020 and a new HACL* release. The resulting code was deployed in the Linux kernel, Mozilla Firefox, WireGuard VPN, and Microsoft MsQuic.

(2) HACLxN: We developed a new methodology for writing and verifying generic SIMD crypto code that can be compiled to efficient code on multiple platforms that support vector instructions, including ARM Neon and Intel AVX, AVX2, and AVX512. We used this methodology to develop high-performance verified SIMD implementations for Chacha20-Poly1305, Blake2, and SHA-2. This work resulted in a paper at ACM CCS 2020 and a new HACL* release. The resulting code is deployed in Mozilla Firefox and Tezos Blockchain.

New in 2021: We added new primitives to HACL* including RSA-PSS and FFDHE, along with a generic verified Bignum library. We also added support for IBMz and Power architectures.

URL: https://github.com/mitls/hacl-star

Contact: Karthikeyan Bhargavan

7.1.6 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.

Publications: hal-02320347, hal-03002266

Authors: Denis Merigoux, Raphaël Monat

Contact: Denis Merigoux

Partner: Direction Générale des Finances Publiques (DGFiP)

7.1.7 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.

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

Publication: hal-03176482

Contact: Karthikeyan Bhargavan

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

7.1.8 Catala

Keywords: Domain specific, Programming language, Law

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

Release Contributions: Changelog:

- Performance improvements - Better error message formatting - New Markdown syntax - Better lexer factorization - ...

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

Publications: hal-03128248, hal-03159939, hal-03128248, hal-02936606

Contact: Denis Merigoux

Partner: Université Panthéon-Sorbonne

7.1.9 Easycrypt

Keywords: Proof assistant, Cryptography

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

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

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

Publications: hal-03352062, hal-03469015

Contact: Gilles Barthe

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

7.1.10 Squirrel

Name: Squirrel Prover

Keywords: Proof assistant, Cryptographic protocol

Functional Description: Squirrel is a proof assistant based designed to prove that security protocols are computationally secure. It is based on a meta-logic built above CCSA logic.

Squirrel allows to specify security protocol in a variant of the applied pi-calculus, and handles unbounded replication and stateful protocols. It can be used to prove both correspondence (e.g. authentication) and equivalence security properties (e.g. strong secrecy, unlinkability).

News of the Year: Adrien Koutsos made several new improvements to the Squirrel prover this year: i) improved user interface, with the addition of an include system and a small general purpose library, better error messages, and a type system, ii) a new mode allowing for explicit handling of proof contexts (e.g. explicit variable and hypothesis naming, structured proofs), which permit to have more stable and maintainable proofs, iii) advanced tactics (rewrite, apply, rewriting hints) supporting a proof style a la SSReflect, which help reduce user inputs.

Publications: hal-03172119, hal-03264227

Contact: Adrien Koutsos

Participants: David Baelde, Stephanie Delaune, Charlie Jacomme, Solene Moreau, Adrien Koutsos

Partners: IRISA, ENS Rennes

7.2 New platforms

No new platforms in 2021

8 New results

8.1 Survey on computer-aided cryptography

Participants: Bruno Blanchet, Karthikeyan Bhargavan.

In collaboration with Manuel Barbosa, Gilles Barthe, Cas Cremers, Kevin Liao, and Bryan Parno, we wrote a survey on computer-aided cryptography, published at IEEE Security and Privacy 2021 [15].

8.2 Verification of security protocols in the symbolic model: tool ProVerif

Participants: Bruno Blanchet, Vincent Cheval.

In collaboration with Vincent Cheval and Véronique Cortier, we (Bruno Blanchet) wrote a paper on important improvements of ProVerif: 1) support for integer counters, with incrementation and inequality tests, 2) lemmas and axioms to give intermediate results to ProVerif, which it exploits to help proving subsequent queries, by deriving additional information in the Horn clauses that it uses to perform the proofs, 3) proofs by induction on the length of the trace, by giving as lemma the property to prove, but obviously for strictly shorter traces, 4) temporal queries, which allow to order events. The soundness of these features is proved (by hand). Moreover, many algorithms used in ProVerif (generation of clauses, resolution, subsumption ...) were optimized, resulting in impressive speedups on large examples. These features are included in ProVerif 2.02pl1 and the paper will appear at IEEE Security and Privacy 2022 [20].

8.3 Verification of security protocols in the computational model: tool CryptoVerif

Participants: Karthikeyan Bhargavan, Bruno Blanchet, Benjamin Lipp.

Bruno Blanchet continued the development of our protocol verification tool CryptoVerif. The new features of this year are detailed in the section on software.

In collaboration with Joël Alwen, Eduard Hauck, Eike Kiltz, and Doreen Riepel, team members Bruno Blanchet and Benjamin Lipp verified the *Hybrid Public Key Encryption* (HPKE) scheme. It is an emerging standard currently under consideration by the Crypto Forum Research Group (CFRG) of the IETF as a candidate for formal approval. Of the four modes of HPKE, we analyse the authenticated mode $\mathsf{HPKE}_{\mathsf{Auth}}$ in its single-shot encryption form as it contains what is, arguably, the most novel part of HPKE. $\mathsf{HPKE}_{\mathsf{Auth}}$'s intended application domain is captured by a new primitive which we call Authenticated Public Key Encryption (APKE). We provide syntax and security definitions for APKE schemes, as well as for the related Authenticated Key Encapsulation Mechanisms (AKEMs). We prove security of the AKEM scheme $\mathsf{DH-AKEM}$ underlying $\mathsf{HPKE}_{\mathsf{Auth}}$ based on the Gap Diffie-Hellman assumption and provide general AKEM/DEM composition theorems with which to argue about $\mathsf{HPKE}_{\mathsf{Auth}}$'s security. To this end, we also formally analyse $\mathsf{HPKE}_{\mathsf{Auth}}$'s key schedule and key derivation functions. All these proofs are done using the automatic computational verifier CryptoVerif. As an independent contribution, we propose the new framework of *nominal groups* that allows us to capture abstract syntactical and security properties of practical elliptic curves, including the Curve25519 and Curve448 based groups (which do not constitute cyclic groups). This work appeared at EuroCrypt 2021 [13] (long version [34]).

Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan implemented a compiler from CryptoVerif models to executable F* implementations. This compiler extends a previous translation from CryptoVerif to OCaml by generating lemmas for equational assumptions made on primitives in CryptoVerif. These assumptions can then be proved in F*. For instance, we prove that message parsing composed with the corresponding serialization is the identity. In the future, we plan to translate the security properties proved by CryptoVerif into F*, so that these properties can be assumed in subsequent F* proofs.

In collaboration with Pierre-Yves Strub (Ecole Polytechnique), Pierre Boutry, Christian Doczkal, et Benjamin Grégoire (Inria Sophia), team member Bruno Blanchet worked on a translation from CryptoVerif security assumptions on primitives to EasyCrypt. These assumptions can then be proven in EasyCrypt from lower-level cryptographic assumptions. This is useful since CryptoVerif provides a high degree of automation but is better at proving protocols than primitives, while EasyCrypt has the expressive power needed to reason both about protocols and about the correctness of primitives, but this expressive power comes at the cost of providing less automation. Hence, proving primitives in EasyCrypt and protocols in CryptoVerif provides the best trade-off. We applied our approach to the proof of the Computational Diffie-Hellman (CDH) model in CryptoVerif (which includes several Diffie-Hellman pairs and allows key compromise) from the standard CDH assumption for one Diffie-Hellman pair. We are currently working to apply it to the Gap Diffie-Hellman model and to an authenticated Key Encapsulation Mechanism (AKEM).

8.4 Verification of Security Protocols in the Computational Model: Squirrel

Participants: Adrien Koutsos.

In collaboration with David Baelde, Stephanie Delaune, Charlie Jacomme and Solene Moreau, Adrien Koutsos designed a new framework and an interactive prover – the Squirrel Prover – allowing to mechanize proofs of security protocols for an arbitrary number of sessions in the computational model. This framework consists of a meta-logic based upon the CCSA logic [37], and a proof system for deriving security properties. Proofs in Squirrel only deal with high-level, symbolic representations of protocol executions, similar to proofs in the symbolic model, but providing security guarantees at the computational level. We have performed a number of case studies in Squirrel, covering a variety of primitives (e.g. hashes, encryption) and security properties (e.g. authentication, unlinkability). This work has been published at IEEE Security and Privacy 2021 [14], and the Squirrel Prover is open source, and continues to be developed.

Squirrel approach to protocol security is complementary w.r.t. existing tools in the computational model (e.g. CryptoVerif or EasyCrypt). Squirrel particularly shines when analyzing stateful protocols, which are often out-of-reach of existing tools, or require a very large amount of work. Work in that direction has been started, and early results have been presented at the LFMTP'21 workshop [25].

8.5 Mechanized Complexity Proofs for Cryptographic Reductions

Participants: Adrien Koutsos.

In collaboration with Manuel Barbosa, Gilles Barthe, Benjamin Gregoire and Pierre-Yves Strub, Adrien Koutsos enhanced the EasyCrypt proof assistant with a Hoare logic for reasoning about computational complexity (execution time and oracle calls) of adversarial computations. Our Hoare logic is built on top of EasyCrypt module system used for modeling adversaries. We proved that our logic is sound w.r.t. EasyCrypt semantics — incidentally, our work provided the first full semantics for the EasyCrypt module system, which was previously lacking. We exemplified this extension by revisiting the security proofs of some well-known cryptographic constructions, and by conducting a new formalization of

Universal Composability (UC). This work was published at ACM CCS'21 [16], and has been integrated in the EasyCrypt tool.

8.6 Journal Paper: Decidability Result

Participants: Adrien Koutsos.

A journal version of the IEEE CSF'19 paper [54]:

Decidability of a Sound Set of Inference Rules for Computational Indistinguishability

has been published at ACM Transactions on Computational Logic [11]. In this work, we prove the decidability of a set of first-order axioms which are computationally sound, though incomplete, for protocols with a bounded number of sessions whose security is based on an IND-CCA_2 encryption scheme. Alternatively, our result can be viewed as the decidability of a family of cryptographic game transformations. Our proof relies on term rewriting and automated deduction techniques.

8.7 High-Assurance High-Performance Crypto

Participants: Karthikeyan Bhargavan, Marina Polubelova, Benjamin Beurdouche, Natalia Kulatova, Jonathan Protzenko, Adrien Koutsos.

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.

While HACL* includes best-in-class C implementations of many popular cryptographic algorithms, on certain platforms, significantly improved performance can be obtained by exploiting low-level instructions that are only available to assembly code. The EverCrypt API brings together verified C code from HACL* with verified Intel assembly code from the Vale project and packages them in a verified multiplexed agile API that can be conveniently used by applications. Our work resulted in a new release of HACL* and a paper at the IEEE Security and Privacy conference [59].

In a second line of work, we propose a new methodology called HACLxN for building formally verified cryptographic code that exploits single-instruction multiple data (SIMD) parallelism. We show how to write and verify code once and then compile it to multiple platforms that support vector instructions, including ARM Neon and Intel AVX, AVX2, and AVX512. We apply our methodology to obtain verified vectorized implementations in C on all these platforms for the ChaCha20 encryption algorithm, the Poly1305 one-time MAC, and the SHA-2 and Blake2 families of hash algorithms. The resulting C code has performance that is comparable to hand-optimized assembly for these platforms. This work led to a new release of HACL* and a paper at ACM CCS 2020 [58]. This year we extended HACL* with code for more primitives, such as RSA-PSS and FFDHE, based on a generic verified Bignum library. We also added support for IBMz and Power architectures.

High-assurance cryptography aims at building efficient cryptographic software with machine-checked proofs of memory safety, functional correctness, provable security, and absence of timing leaks. Traditionally, these guarantees are established under a sequential execution semantics. However, this semantics does not correspond to the behavior of modern processors, that make use of speculative execution to improve performance. This semantics mismatch has been exploited to conduct attacks (e.g. Spectre). In a new line of work, published at IEEE Security and Privacy 2021 [17], we showed that the benefits of high-assurance cryptography can be extended to speculative execution, costing only a modest performance overhead. To do this, we built atop the Jasmin verification framework an end-to-end approach for proving properties of cryptographic software under speculative execution. This approach has been validated experimentally with efficient, functionally correct assembly implementations of ChaCha20 and Poly1305, which have been proved secure against both traditional timing and speculative execution attacks.

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

Participants: Karthikeyan Bhargavan.

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.

Our work extends a long line of research on using dependent type systems for this task, 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. This approach enables us to 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* is built as a library of F* modules that includes a model of low-level protocol execution, a Dolev-Yao symbolic attacker, and generic security abstractions and lemmas, all verified using F*. The library exposes a high-level API that facilitates succinct security proofs for protocol code. We demonstrate the effectiveness of this approach through a detailed symbolic security analysis of the Signal protocol that is based on an interoperable implementation of the protocol from prior work, and is the first mechanized proof of Signal to account for forward and post-compromise security over an unbounded number of protocol rounds.

The design of DY* and a detailed analysis of Signal was published at IEEE Euro S&P 2021 [19] and a tutorial introduction to the framework appeared in [24]. We also applied DY* to an in-depth analysis of the ACME protocol, which is used by the LetsEncrypt CA to issue the majority of certificates on the Web [18].

8.9 Extensions to F*

Participants: Denis Merigoux.

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

In new work this year, we developed a semantics for concurrent separation logic (CSL) within the F* proof assistant in a manner that enables dependently-typed, effectful F* programs to make use of concurrency and to be specified and verified using a full-featured, extensible CSL. In contrast to prior approaches, we directly derive the partial-correctness Hoare rules for CSL from the denotation of computations in the effectful semantics of non-deterministically interleaved atomic actions. Demonstrating the flexibility of our semantics, we build generic, verified libraries that support various concurrency constructs, ranging from dynamically allocated, storable spin locks, to protocol-indexed channels. This work has led to on ongoing re-architecture of the core of the F* framework, and was published at ICFP 2020 [64].

8.10 Formalizing and Implementing Tax Law

Participants: Denis Merigoux, Aymeric Fromherz.

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. 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. This line of work has yielded papers at CC 2020 [22] and JFLA [56], as well as a successful industrial technology transfer from Inria to DGFiP.

2021 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 [52, 51]. On the formal methods side, the simple and clean design of the Catala semantics [12] allows for extension into a proper proof platform for the law [23].

9 Bilateral contracts and grants with industry

9.1 Bilateral grants with industry

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

- Grant from Nomadic Labs - Inria
- PIs: Catalin Hritcu and 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.

10 Partnerships and cooperations

10.1 National initiatives

10.1.1 ANR

TECAP

Title: TECAP: Protocol Analysis - Combining Existing Tools (ANR générique 2017.)

Other partners: Inria Nancy/EPI PESTO, Inria Sophia Antipolis/EPI MARELLE, IRISA, LIX, LSV - ENS Cachan.

Duration: January 2018 - June 2022

Coordinator: Vincent Cheval, EPI Prosecco, Inria Paris (France)

Participants: Bruno Blanchet, Benjamin Lipp, Vincent Cheval

Summary: A large variety of automated verification tools have been developed to prove or find attacks on security protocols. These tools differ in their scope, degree of automation, and attacker models. The aim of this project is to get the best of all these tools, meaning, on the one hand, to improve the theory and implementations of each individual tool towards the strengths of the others and, on the other hand, build bridges that allow the cooperations of the methods/tools. We will focus in this project on the tools CryptoVerif, EasyCrypt, Scary, ProVerif, Tamarin, AKiSs and APTE.

11 Dissemination

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

Member of the organizing committees

- Karthikeyan Bhargavan: Co-Organizer for VeriCrypt and HACS workshops

11.1.2 Scientific events: selection

Member of the conference program committees

- Karthikeyan Bhargavan: PC Member for IEEE S&P, ACM CCS, IEEE CSF, Indocrypt
- Vincent Cheval: PC Member for ACM CCS

11.1.3 Journal

Member of the editorial boards

- Associate Editor of ACM Transactions on Privacy and Security (TOPS) - Karthikeyan Bhargavan
- Associate Editor of the International Journal of Applied Cryptography (IJACT) – Inderscience Publishers: Bruno Blanchet

11.1.4 Invited talks

- Karthikeyan Bhargavan: Invited talk at Developing Secure Systems Summit (DS3)
- Karthikeyan Bhargavan: Invited talk at Indocrypt 2021

11.1.5 Scientific expertise

- Bruno Blanchet was a scientific consultant for Nomadic Labs, regarding the development of the blockchain Tezos.

11.1.6 Research administration

- Bruno Blanchet was a representative of Inria Paris at the DIM RFSI (Domaine d'Intérêt Majeur, Réseau Francilien en Sciences Informatiques).

11.2 Teaching - Supervision - Juries

11.2.1 Teaching

- Master: Karthikeyan Bhargavan, Cryptographic protocols: formal and computational proofs, 18h equivalent TD, master M2 MPRI, université Paris VII
- Master: Vincent Cheval, Cryptographic protocols: formal and computational proofs, 18h equivalent TD, master M2 MPRI, université Paris VII
- Master: Adrien Koutsos, Cryptographic protocols: formal and computational proofs, 18h equivalent TD, master M2 MPRI, université Paris VII
- PhD: Karthikeyan Bhargavan, Introduction: Formal Verification meets Applied Cryptography; Bruno Blanchet, Introduction to Symbolic and Computational Proofs; Vincent Cheval, ProVerif tutorial, in VeriCrypt: An Introduction to Tools for Verified Cryptography, 18-19 December 2021
- Master and PhD: Karthikeyan Bhargavan, Formal Software Analysis, IIT Delhi

11.2.2 Supervision

- PhD in progress: Benjamin Lipp, On Mechanised Cryptographic Proofs of Protocols and their Link with Verified Implementations, ENS Paris, since October 2018, supervised by Bruno Blanchet and Karthikeyan Bhargavan.
- PhD in progress: Natalia Kulatova, Formal Analysis of Security Devices, PSL, since September 2017, supervised by Karthikeyan Bhargavan and Graham Steel.
- PhD in progress: Marina Polubelova, Formal Verification of a Cryptographic Library, PSL, since September 2017, supervised by Karthikeyan Bhargavan.
- PhD defended: Denis Merigoux, Verification framework for performance-oriented memory-safe programming languages, defended on December 13, 2021, supervised by Karthikeyan Bhargavan and Jonathan Protzenko.

11.2.3 Juries

- Karthikeyan Bhargavan was on the PhD jury of Darius Mercadier and Lucas Franceschino

11.3 Popularization

11.3.1 Interventions

- Bruno Blanchet, Vérification de protocoles cryptographiques. ENS, Paris, September 2021.

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] V. Cheval, S. Kremer and I. Rakotonirina. 'DEEPSEC: Deciding Equivalence Properties in Security Protocols - Theory and Practice'. In: *39th IEEE Symposium on Security and Privacy*. San Francisco, United States, May 2018. URL: https://hal.inria.fr/hal-01763122.

[8] 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.

[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] A. Koutsos. 'Decidability of a Sound Set of Inference Rules for Computational Indistinguishability'. In: *ACM Transactions on Computational Logic* 22.1 (22nd Jan. 2021), pp. 1–44. DOI: 10.1145/3423169. URL: https://hal.archives-ouvertes.fr/hal-03469091.

[12] 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://hal.inria.fr/hal-03159939.

International peer-reviewed conferences

[13] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp and D. Riepel. 'Analysing the HPKE Standard'. In: Eurocrypt 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Vol. 12696. Lecture Notes in Computer Science. Zagreb, Croatia: Springer International Publishing, 16th June 2021, pp. 87–116. DOI: 10.1007/978-3-030-77870-5_4. URL: https://hal.inria.fr/hal-03471218.

[14] 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.

[15] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao and B. Parno. 'SoK: Computer-Aided Cryptography'. In: SP 2021 - 42nd IEEE Symposium on Security and Privacy. Virtual Conference, United States, 23rd May 2021. URL: https://hal.inria.fr/hal-03046757.

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

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

[18] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseyni, R. Küsters, G. Schmitz and T. Würtele. 'An In-Depth Symbolic Security Analysis of the ACME Standard'. In: CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event Republic of Korea, France: ACM, 15th Nov. 2021, pp. 2601–2617. DOI: 10.1145/3460120.3484588. URL: https://hal.inria.fr/hal-03540403.

[19] 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, 6th Sept. 2021. URL: https://hal.inria.fr/hal-03178425.

[20] 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.

[21] M. J. Gabbay, A. Jakobsson and K. Sojakova. 'Money Grows on (Proof-)Trees: The Formal FA1.2 Ledger Standard'. In: 3rd International Workshop on Formal Methods for Blockchains. Los Angeles, United States, 18th July 2021. DOI: 10.4230/OASIcs.FMBC.2021.2. URL: https://hal.inria.fr/hal-03498062.

[22] D. Merigoux, R. Monat and J. Protzenko. 'A Modern Compiler for the French Tax Code'. In: CC '21: 30th ACM SIGPLAN International Conference on Compiler Construction. CC 2021: Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. Virtual, South Korea: ACM, Mar. 2021, pp. 71–82. DOI: 10.1145/3446804.3446850. URL: https://hal.inria.fr/hal-03002266.

Conferences without proceedings

[23] A. Delaët, D. Merigoux and A. Fromherz. 'Turning Catala into a Proof Platform for the Law'. In: POPL 2022 - Programming Languages and the Law. Philadelphia, United States, 16th Jan. 2022. URL: https://hal.inria.fr/hal-03447072.

Scientific book chapters

[24] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseyni, R. Küsters, G. Schmitz and T. Würtele. 'A Tutorial-Style Introduction to DY*\$'. In: *Protocols, Strands, and Logic*. Vol. 13066. Lecture Notes in Computer Science. Springer International Publishing, 19th Nov. 2021, pp. 77–97. DOI: 10.1007/978-3-030-91631-2_4. URL: https://hal.inria.fr/hal-03540824.

Reports & preprints

[25] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos and S. Moreau. *Extending the SQUIRREL meta-logic for reasoning over security protocols: Work in Progress.* 18th June 2021. URL: https://hal.archives-ouvertes.fr/hal-03264227.

[26] S. HO, J. Protzenko, A. Bichhawat and K. Bhargavan. *Noise*: A Library of Verified High-Performance Secure Channel Protocol Implementations.* Inria, 10th Dec. 2021. URL: https://hal.inria.fr/hal-03474303.

[27] L. Huttner and D. Merigoux. *Catala: Moving Towards the Future of Legal Expert Systems*. 8th Jan. 2022. URL: https://hal.inria.fr/hal-02936606.

[28] D. Merigoux, F. Kiefer and K. Bhargavan. *Hacspec: succinct, executable, verifiable specifications for high-assurance cryptography embedded in Rust*. Inria, 22nd Mar. 2021. URL: https://hal.inria.fr/hal-03176482.

12.3 Cited publications

[29] 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: http://prosecco.gforge.inria.fr/personal/bblanche/publications/AbadiBlanchetJACM7037.pdf.

[30] 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: http://prosecco.gforge.inria.fr/personal/bblanche/publications/AbadiBlanchetFournetTISSEC07.pdf.

[31] C. Abate, A. Azevedo de Amorim, R. Blanco, A. N. Evans, G. Fachini, C. Hriţcu, T. Laurent, B. C. Pierce, M. Stronati and A. Tolmach. 'When Good Components Go Bad: Formally Secure Compilation Despite Dynamic Compromise'. In: *25th ACM Conference on Computer and Communications Security (CCS)*. ACM, Oct. 2018, pp. 1351–1368. URL: https://arxiv.org/abs/1802.00588.

[32] C. Abate, R. Blanco, D. Garg, C. Hriţcu, M. Patrignani and J. Thibault. 'Journey Beyond Full Abstraction: Exploring Robust Property Preservation for Secure Compilation'. In: *32nd IEEE Computer Security Foundations Symposium (CSF)*. IEEE, June 2019, pp. 256–271. DOI: 10.1109/CSF.2019.00025. URL: https://arxiv.org/abs/1807.04603.

[33] D. Ahman, C. Hriţcu, 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/.

[34] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp and D. Riepel. *Analysing the HPKE Standard*. Research Report. IACR Cryptology ePrint Archive, Nov. 2020. URL: https://hal.inria.fr/hal-03113251.

[35] A. Azevedo de Amorim, M. Dénès, N. Giannarakis, C. Hritcu, B. C. Pierce, A. Spector-Zabusky and A. Tolmach. 'Micro-Policies: Formally Verified, Tag-Based Security Monitors'. In: *36th IEEE Symposium on Security and Privacy (Oakland S&P)*. IEEE Computer Society, May 2015, pp. 813–830. DOI: 10.1109/SP.2015.55. URL: http://prosecco.gforge.inria.fr/personal/hritcu/publications/micro-policies.pdf.

[36] 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.

[37] G. Bana and H. Comon-Lundh. 'A Computationally Complete Symbolic Attacker for Equivalence Properties'. In: *ACM Conference on Computer and Communications Security (CCS'14)*. New York, NY, USA: ACM, Nov. 2014, pp. 609–620.

[38] 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.

[39] K. Bhargavan, B. Bond, A. Delignat-Lavaud, C. Fournet, C. Hawblitzel, C. Hriţcu, S. Ishtiaq, M. Kohlweiss, R. Leino, J. Lorch, K. Maillard, J. Pan, 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.

[40] 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.

[41] 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.

[42] 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.

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

[44] B. Blanchet. 'Automatic Verification of Correspondences for Security Protocols'. In: *Journal of Computer Security* 17.4 (July 2009), pp. 363–434. URL: http://prosecco.gforge.inria.fr/personal/bblanche/publications/BlanchetJCS08.pdf.

[45] B. Blanchet, M. Abadi and C. Fournet. 'Automated Verification of Selected Equivalences for Security Protocols'. In: *Journal of Logic and Algebraic Programming* 75.1 (Feb. 2008), pp. 3–51. URL: http://prosecco.gforge.inria.fr/personal/bblanche/publications/BlanchetAbadiFournetJLAP07.pdf.

[46] 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: http://prosecco.gforge.inria.fr/personal/bblanche/publications/BlanchetPodelskiTCS04.html.

[47] 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.

[48] 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.

[49] 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.

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

[51] L. Huttner and D. Merigoux. 'Catala: Moving Towards the Future of Legal Expert Systems'. working paper or preprint. Jan. 2022. URL: https://hal.inria.fr/hal-02936606.

[52] 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://hal.inria.fr/hal-03128248.

[53] 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.

[54] A. Koutsos. 'Decidability of a Sound Set of Inference Rules for Computational Indistinguishability'. In: *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, pp. 48–61. DOI: 10.1109/CSF.2019.00011. URL: https://doi.org/10.1109/CSF.2019.00011.

[55] K. Maillard, D. Ahman, R. Atkey, G. Martínez, C. Hrițcu, 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.

[56] D. Merigoux, R. Monat and C. Gaie. 'Étude formelle de l'implémentation du code des impôts'. In: *JFLA 2020 - 31ème Journées Francophones des Langages Applicatifs*. Gruissan, France, Jan. 2020. URL: https://hal.inria.fr/hal-02320347.

[57] 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.

[58] 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.

[59] 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.

[60] J. Protzenko, J. K. Zinzindohoué, A. Rastogi, T. Ramananandro, P. Wang, S. Zanella-Béguelin, A. Delignat-Lavaud, C. Hriţcu, 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.

[61] 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.

[62] 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.

[63] 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://www.fstar-lang.org/papers/mumon/.

[64] N. Swamy, A. Rastogi, A. Fromherz, D. Merigoux, D. Ahman and G. Martínez. ‘SteelCore: an extensible concurrent separation logic for effectful dependently typed programs’. In: *Proceedings of the ACM on Programming Languages* 4.ICFP (Aug. 2020), pp. 1–30. DOI: 10.1145/3409003. URL: https://hal.inria.fr/hal-02936273.

[65] 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: http://doi.acm.org/10.1145/3133956.3134043.