RESEARCH CENTRE

Paris

2020

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 Team: 2012 January 01, updated into 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]
- Roberto Blanco Martinez [Inria, Starting Research Position, until May 2020]
- Vincent Cheval [Inria, Researcher, from Sep 2020]
- Catalin Hritcu [Inria, Researcher, until Apr 2020, HDR]
- Adrien Koutsos [Inria, Researcher, from Oct 2020]
- Prasad Naldurg [Inria, Advanced Research Position]
- Exequiel Rivas Gadda [Inria, Starting Research Position]
- Kristina Sojakova [Inria, Starting Research Position, from Jul 2020]

PhD Students

- Carmine Abate [Inria, until Apr 2020]
- Benjamin Beurdouche [Inria, until Jan 2020]
- Son Ho [Inria, from Sep 2020]
- Natalia Kulatova [Inria]
- Theo Laurent [Inria, from Jul 2020]
- Benjamin Lipp [Inria]
- Denis Merigoux [Inria]
- Marina Polubelova [Inria]
- Jérémy Thibault [Inria, until Apr 2020]

Technical Staff

- Adrien Durier [Inria, Engineer, until Apr 2020]
- Florian Groult [Inria, Engineer]
- Son Ho [Inria, Engineer, from Feb 2020 until Aug 2020]
- Theo Laurent [Inria, Engineer, until Jun 2020]
- Ramkumar Ramachandra [Inria, Engineer, until Jul 2020]
- Antoine Van Muylder [Inria, Engineer, until Jun 2020]
- Theophile Wallez [Inria, Engineer, from Oct 2020]

Interns and Apprentices

- Nicolas Chataing [Ecole normale supérieure Paris-Saclay, from Mar 2020 until Jul 2020]
- Maxime Legoupil [École Normale Supérieure de Paris, from Jul 2020 until Aug 2020]
- Theophile Wallez [Ecole normale supérieure Paris-Saclay, from Apr 2020 until Aug 2020]

Administrative Assistants

- Christelle Guiziou [Inria]
- Mathieu Mourey [Inria]

External Collaborators

- Cezar-Constantin Andrici [Alexandru Ioan Cuza University of Iasi, until Mar 2020]
- Benjamin Beurdouche [Mozilla, from Feb 2020]
- Victor Dumitrescu [Nomadic Labs]
- Adrien Durier [Max Planck Society, from May 2020]
- Guido Martinez [CIFASIS-CONICET and UNR, Rosario, Argentina]
- Jonathan Protzenko [Microsoft Research]
- Eric Tanter [University of Chile, until Mar 2020]

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 protocol implementations, hardware-based security APIs, smartphone- and browser-based web applications, and cloud-based web services. In particular, we aim to verify the full application: both the cryptographic core and the high-level application code. We aim to verify implementations, not just models. 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 effects 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* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest (a larger collaboration with Microsoft Research). This includes verified implementations of TLS 1.2 and 1.3 and of the underlying cryptographic primitives.

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, the reverse engineering and formal test tool Tookan, 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 three approaches:

- ProVerif: a symbolic prover for cryptographic protocol models
- Tookan: an attack-finder for PKCS#11 hardware security devices
- 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 [50]; it has motivated a serious research effort on the formal analysis of cryptographic protocols, starting with [46] 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* [43]. 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 [40, 29], correspondences (including authentication) [41], and observational equivalences [42]. 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 [35], Signal [48], JFK [30], and Web Services Security [39], 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 120 research papers (references available at http://proverif.inria.fr/proverif-users.html).

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 [52], 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# [38]. 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* [54, 33, 49], 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 [51]. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest [36] (a larger collaboration with Microsoft Research). This includes a verified implementation of TLS 1.2 and 1.3 [37] and of the underlying cryptographic primitives [55].

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 [35] and Signal [48] but more work is still needed in order to develop this approach, so that it is easier to apply to more protocols. We also design and implement techniques for proving implementations of protocols secure in the computational model. In particular, CryptoVerif can generate implementations from CryptoVerif specifications that have been proved secure [44]. We plan to continue working on this approach.

A different approach is to directly verify cryptographic applications in the computational model by typing. A recent work [47] 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 two approaches, typechecking and game-based proofs, are complementary. Understanding how to combine these approaches remains an open and active topic of research.

An alternative to direct computation proofs is to identify the cryptographic assumptions under which symbolic proofs, which are typically easier to derive automatically, can be mapped to computational proofs. This line of research is sometimes called computational soundness and the extent of its applicability to real-world cryptographic protocols is an active area of investigation.

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

F* [54, 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 [51]. 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 [37] and of the underlying cryptographic primitives [55].

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 [51]). 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 [34], 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 [45] and TS* [53], 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 have seen considerable progress in identity with the UnlimitID design and with messaging via the IETF MLS effort, with new work on blockchain technology underway.

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

- We published 9 papers at top-tier conferences and journals such as IEEE S&P (2), ACM CCS, ACM POPL, ICFP, LICS, and JFP.
- We publicly released a second version of our cryptographic library HACL*, which was incorporated within the Linux Kernel, Microsoft MsQuic, Mozilla Firefox, WireGuard VPN, Tezos Blockchain, and the ElectionGuard election software.
- We released new versions of the ProVerif and CryptoVerif tools.
- Karthikeyan Bhargavan served as Program Chair of IndoCrypt 2020.
- Benjamin Beurdouche defended his thesis on the "Formal Verification for High Assurance Security Software in F*" in December 202.

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) The command "use_variable x1 ... xn", which tells CryptoVerif to replace all terms equal to the value of the variable xi with xi.

2) The command "guess i" to guess the tested session, a technique often used in cryptographic proofs.

3) The command "assume replace", which replaces a term with another term in a game like "replace", but does not check that the replacement is correct. This is useful to experiment. (Obviously, CryptoVerif does not consider that the protocol is proved when this command is used.)

4) The command "insert" has been extended to insert several tests, assignments as well as event_abort in a single command. For instance, inserting find ... then/else let x = ... in one step allows to test in the condition of the inserted find whether x is defined, and thus which branch of the inserted find was taken.

5) The game indistinguishability axioms used to model the assumptions ROM (random oracle model), PRF (pseudo-random function), PRP (pseudo-random permutation), SPRP (super pseudo-random permutation), and ICM (ideal cipher model) are now generated on-demand by CryptoVerif. That allows more flexibility to optimize these axioms.

6) We improved the probabilities for Diffie-Hellman assumptions that use random self reducibility. In particular, we introduced a probability of distinguishing a rerandomized key from an honestly generated key, so that the transformations can also be applied to Curve25519 and Curve448.

7) When a proof of indistinguability fails, CryptoVerif explains where the two games differ. That makes it easier for the user to know what to do next in the proof, especially when the two games are large and similar.

8) CryptoVerif can now use the total number of calls to an oracle in the probability formulas that it computes, in addition to the number of copies of each replication. That allows CryptoVerif to provide more precise probabilities in case oracles are under several replications.

These changes are included in CryptoVerif version 2.04 available at https://cryptoverif.inria.fr.

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

Publications: 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.

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

Contacts: Catalin Hritcu, Karthikeyan Bhargavan

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 is under submission.

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

Publications: hal-01947972, hal-01423742, hal-01306440, hal-01423760, hal-01102136, hal-01575920, hal-01528752, hal-01575923, hal-01527671, hal-01575861

Authors: Bruno Blanchet, Vincent Cheval, Marc Sylvestre

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.

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

Contact: Karthikeyan Bhargavan

8 New results

8.1 Verification of security protocols

Participants Bruno Blanchet, Karthikeyan Bhargavan, Vincent Cheval, Benjamin Lipp.

In collaboration with Manuel Barbosa, Gilles Barthe, Cas Cremers, Kevin Liao, and Bryan Parno, we (team members Karthikeyan Bhargavan Bruno Blanchet) wrote a survey on computer-aided cryptography, to appear at IEEE Security and Privacy 2021 [14].

We also continued the development of our protocol verification tools ProVerif and CryptoVerif. The new features of this year are detailed in the section on software.

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 is under submission.

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 $\mathsf{DH}-\mathsf{AKEM}$ 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 is accepted at EuroCrypt 2021 [23]; a preliminary work on this topic appears as [26].

8.2 Foundations of Software Verification

Participants Catalin Hritcu, Exequiel Rivas, Kenji Maillard, Antoine Van Muylder.

Relational program logics have a broad range of practical applications, such as proving security properties like non-interference. In our new paper at POPL 2020 [11], we extend our previous work on verification of unary properties of effectful code, Dijkstra monads, to the setting of relational properties, and show how to obtain relational program logics for general effects.

Algebraic effects and handlers are modern programming constructions that allow to write monadic effectful programs in a modular way. In a new article in JFP [12], we extend the theory of algebraic effects and handlers to other popular notions of computation: applicative functors and arrows. Doing so, we support different degrees of static analysis on effectful code that were not possible only with the monadic interface.

Certain computer systems can be conveniently understood as entities dialoguing in a client/server fashion. In a new article at LICS 2020 [16], we take an abstract view of these systems, and use monads and comonads to represent computer programs and environments. The interaction between a program (or client) and environment (or server) is studied from a theoretical point of view, showing a number of mathematical properties, and relating these interactions to previously well-known constructions in category theory.

8.3 High-Assurance High-Performance Crypto

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

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 [19].

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 [18].

8.4 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 [13].

8.5 Formalizing and Implementing Tax Law

Participants Denis Merigoux.

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 [17] and JFLA [21], and has initiated new work on legal expert systems [20, 28].

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 International initiatives

10.1.1 Inria international partners

Informal international partners We have a range of long- and short-term collaborations with various universities and research labs. We summarize them by project:

- Protocol analysis: Microsoft Research (Cambridge), Mozilla, University of Rennes, University of Stuttgart
- F*: Microsoft Research (Redmond, Cambridge, Bangalore), MSR-Inria, CMU, MIT, University of Ljubljana, Nomadic Labs, Zen Protocol, Princeton University
- SECOMP: MPI-SWS, CISPA, Stanford University, CMU, University of Pennsylvania, Portland State University, University of Virginia, University of Iași

10.1.2 Participation in other international programs

SSITH/HOPE

Title: Advanced New Hardware Optimized for Policy Enforcement, A New HOPE

Program: DARPA SSITH

Duration: December 2017 - February 2021

Coordinator: Charles Stark Draper Laboratory

Other Participants: Inria Paris, University of Pennsylvania, MIT, Portland State University, Dover Microsystems, DornerWorks

Prosecco Participants: Catalin Hritcu, Roberto Blanco, Jérémy Thibault

Summary: A New HOPE builds on results from the Inherently Secure Processor (ISP) project that has been internally funded at Draper. Recent architectural improvements decouple the tagged architecture from the processor pipeline to improve performance and flexibility for new processors. HOPE securely maintains metadata for each word in application memory and checks every instruction against a set of installed security policies. The HOPE security architecture exposes tunable parameters that support Performance, Power, Area, Software compatibility and Security (PPASS) search space exploration. Flexible software-defined security policies cover all 7 SSITH CWE vulnerability classes, and policies can be tuned to meet PPASS requirements; for example, one can trade granularity of security checks against performance using different policy configurations. HOPE will design and formalize a new high-level domain-specific language (DSL) for defining security policies, based on previous research and on extensive experience with previous policy languages. HOPE will formally verify that installed security policies satisfy system-wide security requirements.

A secure boot process enables policies to be securely updated on deployed HOPE systems. Security policies can adapt based on previously detected attacks. Over the multi-year, multi-million dollar Draper ISP project, the tagged security architecture approach has evolved from early prototypes based on results from the DARPA CRASH program towards easier integration with external designs, and is better able to scale from micro to server class implementations. A New HOPE team is led by Draper and includes faculty from University of Pennsylvania (Penn), Portland State University (PSU), INRIA, and MIT, as well as industry collaborators from DornerWorks and Dover Microsystems. In addition to Draper's in-house expertise in hardware design, cyber-security (defensive and offensive, hardware and software) and formal methods, the HOPE team includes experts from all domains relevant to SSITH, including (a) computer architecture: DeHon (Penn), Shrobe (MIT); (b) formal methods including programming languages and security: Pierce (Penn), Tolmach (PSU), Hritcu (INRIA); and (c) operating system integration (DornerWorks). Dover Microsystems is a spin-out from Draper that will commercialize concepts from the Draper ISP project.

Everest Expedition

Program: Microsoft Expedition and MSR-Inria Collaborative Research Project

Expedition Participants: Microsoft Research (Cambridge, Redmond, Bangalore), Inria, MSR-Inria, CMU, University of Edinburgh

Duration: October 2017 – October 2020

Prosecco Participants: Karthikeyan Bhargavan, Catalin Hritcu, Danel Ahman, Benjamin Beurdouche, Victor Dumitrescu, Nadim Kobeissi, Théo Laurent, Guido Martínez, Denis Merigoux, Marina Polubelova, Jean-Karim Zinzindohoué

Other Participants: from other Inria teams: David Pichardie (Celtique), Jean-Pierre Talpin (TEA)

Summary: The HTTPS ecosystem (HTTPS and TLS protocols, X.509 public key infrastructure, crypto algorithms) is the foundation on which Internet security is built. Unfortunately, this ecosystem is brittle, with headline-grabbing attacks such as FREAK and LogJam and emergency patches many times a year.

Project Everest addresses this problem by constructing a high-performance, standards-compliant, formally verified implementation of components in HTTPS ecosystem, including TLS, the main protocol at the heart of HTTPS, as well as the main underlying cryptographic algorithms such as AES, SHA2 or X25519.

At the TLS level, for instance, we are developing new implementations of existing and forthcoming protocol standards and formally proving, by reduction to cryptographic assumptions on their core algorithms, that our implementations provide a secure-channel abstraction between the communicating endpoints. Implementations of the core algorithms themselves are also verified, producing performant portable C code or highly optimized assembly language.

We aim for our verified components to be drop-in replacements suitable for use in mainstream web browsers, servers, and other popular tools and are actively working with the community at large to improve the ecosystem.

https://project-everest.github.io

10.2 European initiatives

10.2.1 FP7 & H2020 Projects

10.2.2 ERC Consolidator Grant: CIRCUS

Title: CIRCUS: An end-to-end verification architecture for building Certified Implementations of Robust, Cryptographically Secure web applications

Duration: April 2016 - March 2021

Coordinator: Karthikeyan Bhargavan, INRIA

Summary: The security of modern web applications depends on a variety of critical components including cryptographic libraries, Transport Layer Security (TLS), browser security mechanisms, and single sign-on protocols. Although these components are widely used, their security guarantees remain poorly understood, leading to subtle bugs and frequent attacks. Rather than fixing one attack at a time, we advocate the use of formal security verification to identify and eliminate entire classes of vulnerabilities in one go.

CIRCUS proposes to take on this challenge, by verifying the end-to-end security of web applications running in mainstream software. The key idea is to identify the core security components of web browsers and servers and replace them by rigorously verified components that offer the same functionality but with robust security guarantees.

10.2.3 ERC Starting Grant: SECOMP

Title: SECOMP: Efficient Formally Secure Compilers to a Tagged Architecture

Duration: Jan 2017 - December 2021

Coordinator: Catalin Hritcu, INRIA

Summary: The SECOMP project is aimed at leveraging 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. We will use property-based testing and formal verification to provide high confidence that our compilers are indeed secure.

10.3 National initiatives

10.3.1 ANR

SafeTLS

Title: SafeTLS: La sécurisation de l’Internet du futur avec TLS 1.

Other partners: Université Rennes 1, IRMAR, INRIA Sophia Antipolis, SGDSN/ANSSI

Duration: October 2016 - September 2020

Coordinator: Pierre-Alain Fouque, Université de Rennes 1 (France)

Participants: Karthikeyan Bhargavan

Summary: Our project, SafeTLS, addresses the security of both TLS 1.3 and of TLS 1.2 as they are (expected to be) used, in three important ways: (1) A better understanding: We will provide a better understanding of how TLS 1.2 and 1.3 are used in real-world applications; (2) Empowering clients: By developing a tool that will show clients the quality of their TLS connection and inform them of potential security and privacy risks; (3) Analyzing implementations: We will analyze the soundness of current TLS 1.2 implementations and use automated verification to provide a backbone of a secure TLS 1.3 implementation.

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 - December 2021

Coordinator: Vincent Cheval, EPI PESTO/EPI Prosecco, Inria Nancy/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

General chair, scientific chair

- Karthikeyan Bhargavan was a co-organizer for the VeriCrypt tutorial on Formal Methods for Cryptographic Systems at IndoCrypt, in December 2020.
- Karthikeyan Bhargavan was a co-organizer for the High-Assurance Cryptographic Software workshop in January 2020.

11.1.2 Scientific events: selection

Chair of conference program committees

- Karthikeyan Bhargavan was a program chair at IndoCrypt, in December 2020.

Member of the conference program committees

- Bruno Blanchet was PC member of CCS 2020 and IndoCrypt 2020.
- Karthikeyan Bhargavan was a PC member for ACM CCS IEEE S&P, IEEE CSF, and ACNS, in 2020.

11.1.3 Journal

Member of the editorial boards

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

11.1.4 Scientific expertise

- Bruno Blanchet is a member of the specialized temporary scientific committee of ANSM (Agence nationale de sécurité du médicament et des produits de santé), on the cybersecurity of software medical devices.
- Bruno Blanchet was a scientific consultant for Nomadic Labs, regarding the development of the blockchain Tezos.
- Karthikeyan Bhargavan was a scientific consultant for CryptoSense, Facebook, and Apple.

11.1.5 Research administration

- Bruno Blanchet was a member of the hiring scientific jury for Inria researchers (*chargé de recherche* and Inria Starting Faculty Positions) of the Inria Saclay center.
- Bruno Blanchet was a member of the jury for Inria Prizes.
- 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: Bruno Blanchet, Cryptographic protocols: formal and computational proofs, 4.5h equivalent TD, master M2 MPRI, université Paris VII
- 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
- PhD: Bruno Blanchet and Benjamin Lipp, CryptoVerif: Mechanizing Game-Based Proofs, VeriCrypt: An Introduction to Tools for Verified Cryptography, 10-11 December 2020
- PhD: Marina Polubelova, F*, VeriCrypt: An Introduction to Tools for Verified Cryptography, 10-11 December 2020

11.2.2 Supervision

- PhD: Benjamin Beurdouche, "Formal Verification for High Assurance Security Software in F*: Application to communication protocols and cryptographic primitives", supervised by Karthikeyan Bhargavan, defended at PSL on December 18, 2020.
- 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 in progress: Denis Merigoux, Verification framework for performance-oriented memory-safe programming languages, since September 2018, supervised by Karthikeyan Bhargavan and Jonathan Protzenko.

- PhD in progress: Son Ho, Verification tools for low-level software, since September 2020, supervised by Karthikeyan Bhargavan and Jonathan Protzenko.
- PhDs in progress: Carmine Abate, Jérémy Thibault, Guido Martínez (CIFASIS-CONICET Rosario). All three left the team with the departure of Catalin Hritcu.

11.2.3 Juries

- Bruno Blanchet was reviewer of Alexandre Debant's PhD thesis (Université de Rennes 1).
- Bruno Blanchet was member of the PhD jury of Charlie Jacomme (ENS Paris-Saclay).
- Karthikeyan Bhargavan was a reviewer of the PhD thesis of Raphael Rieu-Helft (University of Paris-Saclay)
- Karthikeyan Bhargavan was a reviewer of the PhD thesis of Darius Mercadier (Sorbonne)
- Karthikeyan Bhargavan was a reviewer of the PhD thesis of Ilya Grischenko (University of Wien)

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] 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)*. https://arxiv.org/abs/1802.00588. Toronto, Canada: ACM, Oct. 2018, pp. 1351–1368. DOI: 10.1145/3243734.3243745. URL: https://hal.archives-ouvertes.fr/hal-01949202.

[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] 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.

[7] 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.

[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] K. Maillard, C. Hritcu, E. Rivas and A. Van Muylder. 'The Next 700 Relational Program Logics'. In: *Proceedings of the ACM on Programming Languages* 4.POPL (2020). URL: https://hal.archives-ouvertes.fr/hal-02398927.

[12] R. Pieters, E. Rivas and T. Schrijvers. 'Generalized monoidal effects and handlers'. In: *Journal of Functional Programming* 30 (2020). DOI: 10.1017/S0956796820000106. URL: https://hal.inria.fr/hal-03112837.

[13] 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 (2nd Aug. 2020), pp. 1–30. DOI: 10.1145/3409003. URL: https://hal.inria.fr/hal-02936273.

International peer-reviewed conferences

[14] 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.

[15] T. Díaz, F. Olmedo and É. Tanter. 'A Mechanized Formalization of GraphQL'. In: CPP 2020 - 9th ACM SIGPLAN International Conference on Certified Programs and Proofs. New Orleans, United States: https://popl20.sigplan.org/home/CPP-2020, 20th Jan. 2020. DOI: 10.1145/3372885.3373822. URL: https://hal.archives-ouvertes.fr/hal-02422532.

[16] S.-y. Katsumata, E. Rivas and T. Uustalu. 'Interaction Laws of Monads and Comonads'. In: LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science. Saarbrücken / Virtual, Germany, 8th July 2020, pp. 604–618. DOI: 10.1145/3373718.3394808. URL: https://hal.inria.fr/hal-03112866.

[17] 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: https://conf.researchr.org/home/CC-2021, Mar. 2021, pp. 71–82. DOI: 10.1145/3446804.3446850. URL: https://hal.inria.fr/hal-03002266.

[18] 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, 9th Nov. 2020. URL: https://hal.inria.fr/hal-03154275.

[19] 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, 18th May 2020, pp. 983–1002. DOI: 10.1109/SP40000.2020.00114. URL: https://hal.inria.fr/hal-03154278.

National peer-reviewed Conferences

[20] L. Huttner and D. Merigoux. 'Traduire la loi en code grâce au langage de programmation Catala'. In: Intelligence artificielle et finances publiques. Nice, France, 28th Oct. 2020. URL: https://hal.inria.fr/hal-03128248.

[21] 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: http://jfla.inria.fr/jfla2020.html, 29th Jan. 2020. URL: https://hal.inria.fr/hal-02320347.

Conferences without proceedings

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

Reports & preprints

[23] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp and D. Riepel. *Analysing the HPKE Standard.* IACR Cryptology ePrint Archive, 30th Nov. 2020. URL: https://hal.inria.fr/hal-03113251.

[24] B. Beurdouche. *MLS Architecture: analysis of the security, privacy and functional requirements.* 14th Jan. 2020. URL: https://hal.inria.fr/hal-02439526.

[25] R. Blanco, M. Manighetti and D. Miller. *FPC-Coq: Using ELPI to elaborate external proof evidence into Coq proofs.* Inria Saclay, 5th July 2020. URL: https://hal.inria.fr/hal-02974002.

[26] B. Lipp. *An Analysis of Hybrid Public Key Encryption.* IACR Cryptology ePrint Archive, 23rd Feb. 2020. URL: https://hal.inria.fr/hal-03113221.

[27] D. Merigoux, N. Chataing and J. Protzenko. *Catala: A Programming Language for the Law.* 4th Mar. 2021. URL: https://hal.inria.fr/hal-03159939.

[28] D. Merigoux and L. Huttner. *Catala: Moving Towards the Future of Legal Expert Systems.* 11th Sept. 2020. URL: https://hal.inria.fr/hal-02936606.

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] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

[49] K. Maillard, D. Ahman, R. Atkey, G. Martínez, C. Hriţcu, 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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/.

[55] 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.