SSProve: A Foundational Framework for Modular Cryptographic Proofs in Coq*

Carmine Abate[1], Philipp G. Haselwarter[2], Exequiel Rivas[3], Antoine Van Muylder[4], Théo Winterhalter[1], Cătălin Hriţcu[1], Kenji Maillard[5], and Bas Spitters[2]

[1]MPI-SP [2]Aarhus University [3]Inria Paris [4]Vrije Universiteit Brussel [5]Inria Rennes

State Separating Proofs (SSP) is a recent methodology for structuring game-based cryptographic proofs in a modular way. While very promising, this methodology was previously not fully formalized and came with little tool support. We address this by introducing SSProve, the first general verification framework for SSP. SSProve combines high-level modular proofs about composed protocols, as proposed in SSP, with a probabilistic relational program logic for formalizing the lower-level details, which together enable constructing fully machine-checked crypto proofs in the Coq proof assistant. Moreover, SSProve is itself formalized in Coq, including the algebraic laws of SSP, the soundness of the program logic, and the connection between these two verification styles.

Cryptographic proofs can be challenging to make fully precise and to rigorously check. This has caused a "crisis of rigor" [4] in cryptography, addressed by systematically structuring proofs as sequences of games. This game-based proof methodology is not only ubiquitous in provable cryptography nowadays, but also amenable to full machine-checking in proof assistants such as Coq [1, 10] and Isabelle/HOL [3]. It has also led to the development of specialized proof assistants [2] tools for crypto proofs. There are two key ideas behind these tools: (i) formally representing games and the adversaries against them as code in a probabilistic programming language, and (ii) using program verification techniques to conduct all game transformation steps in a machine-checked manner.

For a long time however, game-based proofs have lacked modularity, which made them hard to scale to large, composed protocols such as TLS or the upcoming MLS. To address this issue, Brzuska et al. [5] have recently introduced *state-separating proofs (SSP)*, a methodology for modular game-based proofs, inspired by the paper proofs in the miTLS project [7], by prior compositional cryptography frameworks, and by process algebras. In the SSP methodology, the code of cryptographic games is split into packages, which are modules made up of procedures sharing state. Packages can call each other's procedures (also known as oracles) and can operate on their own state, but cannot directly access other packages' state. Packages have natural notions of sequential and parallel composition that satisfy simple algebraic laws, such as associativity of sequential composition. These laws are used to define cryptographic reductions not only in SSP, but also in the *The Joy of Cryptography* textbook [12].

While the SSP methodology is very promising, the lack of a complete formalization makes it currently unsuitable for machine-checked proofs. The SSP paper [5] defines package composition and the syntax of a cryptographic pseudocode language for games and adversaries, but the semantics of this language is not formally defined, and the meaning of their `assert` operator is not even clear, given the probabilistic setting. Moreover, while SSP provides a good way to structure proofs at the high-level, using algebraic laws such as associativity, the low-level details of such proofs are treated very casually on paper. Yet none of the existing crypto verification tools that could help machine-check these low-level details supports the high-level part of SSP proofs: equational reasoning about composed packages (i.e., modules) is either not possible at all, or does not exactly match the SSP package abstraction [2, 8].

Contributions. The main contribution of this work is to introduce SSProve, the first general verification framework for machine-checked state-separating proofs. SSProve brings together two different proof styles into a single unified framework: (1) high-level proofs are modular, done by reasoning equationally about composed packages, as proposed in SSP [5]; (2) low-level details are formally proved in a probabilistic relational program logic [1, 2, 10]. Importantly, we show a formal connection between these two proof styles.

SSProve is, moreover, a foundational framework, fully formalized itself in Coq. For this we define the syntax of crypto pseudocode in terms of a free monad, in which external calls are represented as algebraic operations. This gives us a principled way to define sequential composition of packages based on an algebraic effect handler [11] and to give machine-checked proofs of the SSP package laws [5], some of which were treated informally on paper. We moreover make precise the minimal state-separation requirements between adversaries and the games with which

*This work has been accepted at the CSF'21 conference. A full length preprint is available at `eprint.iacr.org/2021/397`.

they are composed—this reduces the proof burden and allows us to prove more meaningful security results, that do not require the adversary's state to be disjoint from intermediate games in the proof.

Beyond just syntax, we also give a denotational semantics to crypto code in terms of stateful probabilistic functions that can signal assertion failures by sampling from the empty probability subdistribution. Finally, we prove the soundness of a probabilistic relational program logic for relating pairs of crypto code fragments.

For this soundness proof we build a semantic model based on relational weakest-precondition specifications. Our model is modular with respect to the considered side-effects (currently probabilities, state, and assertion failures). To obtain it, we follow a general recipe by Maillard et al. [9], who recently proposed to characterize such semantic models as relative monad morphisms, mapping two monadic computations to their canonical relational specification. This allows us to first define a relative monad morphism for probabilistic, potentially failing computations and then to extend this to state by simply applying a relative monad transformer. Working out this instance of Maillard et al.'s [9] recipe involved formalizing various non-standard categorical constructs in Coq, in an order-enriched context: lax functors, lax natural transformations, left relative adjunctions, lax morphisms between such adjunctions, state transformations of such adjunctions, etc. This formalization is of independent interest and should also allow us to more easily add extra side-effects and sub-effecting to SSProve in the future.

Case studies. To test our methodology, we have formalized several security proofs in SSProve. The first example looks at asymmetric encryption scheme built out of a pseudo-random function. The second example proves security of ElGamal, a popular asymmetric encryption scheme. Finally, we tackle the more challenging KEM-DEM example of [5], showing that composing a secure key-encapsulation mechanism (KEM) with a data encapsulation mechanism (DEM) results in a secure public-key encryption mechanism, following [6]. SSProve (circa 20k lines of Coq code, including comments) is available at `github.com/SSProve/ssprove`.

Acknowledgements. We are grateful to Arthur Azevedo de Amorim, Théo Laurent, Nikolaj Sidorenco, and Ramkumar Ramachandra for their technical support and discussions. This work was in part supported by the European Research Council under ERC Starting Grant SECOMP (715753), by AFOSR grant *Homotopy type theory and probabilistic computation* (12595060), and by the Concordium Blockchain Research Center at Aarhus University. Antoine Van Muylder holds a PhD Fellowship from the Research Foundation – Flanders (FWO).

References

[1] G. Barthe, B. Grégoire, and S. Zanella-Béguelin. Formal certification of code-based cryptographic proofs. *POPL*, 2009.

[2] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P. Strub. EasyCrypt: A tutorial. In *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*. 2013.

[3] D. A. Basin, A. Lochbihler, and S. R. Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *J. Cryptol.*, 33(2), 2020.

[4] M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. *IACR Cryptol. ePrint Arch.*, page 331, 2004.

[5] C. Brzuska, A. Delignat-Lavaud, C. Fournet, K. Kohbrok, and M. Kohlweiss. State separation for code-based game-playing proofs. In *ASIACRYPT*. 2018.

[6] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1), 2003.

[7] C. Fournet, M. Kohlweiss, and P. Strub. Modular code-based cryptographic verification. *CCS*. 2011.

[8] A. Lochbihler, S. R. Sefidgar, D. A. Basin, and U. Maurer. Formalizing constructive cryptography using CryptHOL. In *CSF*. 2019.

[9] K. Maillard, C. Hriţcu, E. Rivas, and A. V. Muylder. The next 700 relational program logics. *Proc. ACM Program. Lang.*, 4(POPL), 2020.

[10] A. Petcher and G. Morrisett. The foundational cryptography framework. *POST*. 2015.

[11] G. D. Plotkin and M. Pretnar. Handlers of algebraic effects. *ESOP*. 2009.

[12] M. Rosulek. The Joy of Cryptography. Online textbook, 2021.