Automating Language-Based Cryptographic Proofs

Santiago Zanella Béguelin   Benjamin Grégoire   Sylvain Heraud
INRIA Sophia Antipolis - Méditerranée
{Santiago.Zanella,Benjamin.Gregoire,Sylvain.Heraud}@inria.fr
Gilles Barthe   Federico Olmedo
IMDEA Software
{Gilles.Barthe,Federico.Olmedo}@imdea.org

***Abstract*—CertiCrypt is an automated framework to construct and verify security proofs of cryptographic systems in the computational model. We overview the main characteristics of CertiCrypt, the tools it provides to ease the construction of proofs, the degree of automation it achieves, and the case studies where it has been applied so far.**

I. Introduction

In cryptography, provable security advocates a mathematical approach where the goals and requirements of cryptographic systems are specified precisely, and where the security proof is carried out rigorously and makes explicit the assumptions it relies upon. Security objectives are expressed in complexity-theoretical terms and refer to the probability of an efficient adversary to thwart a security objective, whereas security proofs have the form of reductionist arguments. A typical reductionist proof assumes the existence of an adversary $\mathcal{A}$ that breaks the security of the system with a certain probability ϵ and within a certain amount of time t, and exhibits another adversary $\mathcal{B}$ that uses $\mathcal{A}$ to solve an instance of a computational hard problem with probability $\epsilon - \Delta\epsilon$ within time $t + \Delta t$. The smaller $\Delta\epsilon$ and Δt are, the tighter the proof, and the closer the problem of breaking the security of the system is to the computational hard problem under consideration. The tightness of a reduction is of paramount practical importance, it allows to improve the efficiency of a cryptographic system while maintaining strong security guarantees.

The game-playing technique is a general method to structure and unify reductionist proofs that has been widely applied in the literature. In essence, the game-playing technique suggests to view the interaction between adversary $\mathcal{A}$ and the cryptosystem as a probabilistic game. The winning probability of the adversary corresponds to the probability of breaking security. In the same way, the probability of adversary $\mathcal{B}$ in solving a computational hard problem is specified as the probability of an event in another game. Relating the two probabilities directly can be a complex, error-prone task that involves nested case analysis and reasoning about conditional probabilities. Instead, the proof is broke down into simpler steps of manageable complexity, using a sequence of intermediate games (Fig. 1). Most of the time these steps fall into recurring patterns that can be stated as general lemmas and proven correct once and for all.

Although the adoption of provable security and the game-playing technique has significantly enhanced confidence in cryptographic systems, the community is increasingly wary about security proofs. This is partly due to the fact that proofs are rather involved and rely on different kinds of mathematical reasoning including complexity theory, probability theory and group theory. However, the main reason is to be found in the difficulty in pinpointing hypotheses in proofs and in isolating the creative and original parts from the uninteresting steps occurring in every other proof.

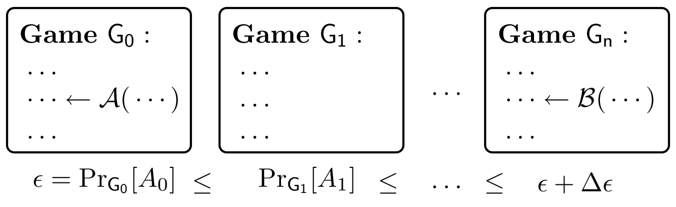
Game G_0 :
$\cdots$
$\cdots \leftarrow \mathcal{A}(\cdots)$
$\cdots$
Game G_1 :
$\cdots$
$\cdots$
$\cdots$
$\cdots$
Game G_n :
$\cdots$
$\cdots \leftarrow \mathcal{B}(\cdots)$
$\cdots$
$\epsilon = \Pr_{\mathsf{G}_0}[A_0] \leq \quad \Pr_{\mathsf{G}_1}[A_1] \leq \quad \ldots \leq \quad \epsilon + \Delta\epsilon$

Fig. 1. Typical structure of a game-based reductionist argument

Bellare and Rogaway [1], and Halevi [2] propose the game-playing technique as a natural solution for taming the complexity of proofs but recognize the need for a fully-specified programming language to represent games. Inspired by these ideas, we took a language-based approach and developed CertiCrypt [3]–[5], a framework built on top of the Coq proof assistant to construct and verify game-based security proofs of cryptographic systems. We put particular emphasis on the following three principles:

- *Generality*: the tool should be sufficiently general not to limit the conduct of cryptographic proofs: its underlying language should be sufficiently expressive to capture all notions and assumptions used by cryptographers, and its underlying logic should be sufficiently powerful so that all forms of mathematical judgments can be formalized and checked;
- *High assurance*: the tool should produce proof objects that are verifiable independently by small and trustworthy proof checkers. In particular, the proof objects should justify all steps in the proof, and should guarantee that side conditions which arise in the application of game transformations are verified;
- *Automation*: the tool should assist the user in proving transitions between games, or even to find the sequence of games. Ideally the tool should provide sufficient support

to handle automatically all routine steps so that users can focus on the creative part of the proof.

II. An automated framework

To describe games we use a typed probabilistic WHILE language with procedure calls:

$$\begin{array}{rcll}
\mathcal{C} & ::= & \mathsf{skip} & \text{nop} \\
& \mid & \mathcal{C};\ \mathcal{C} & \text{sequence} \\
& \mid & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\
& \mid & \mathcal{V} \overset{\$}{\leftarrow} \mathcal{D} & \text{random sampling} \\
& \mid & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\
& \mid & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{while loop} \\
& \mid & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call}
\end{array}$$

We formalized a deep embedding of this language into Coq, which allows us to mechanize program transformation by purely syntactical manipulation. The language of expressions ($\mathcal{E}$) and distribution expressions ($\mathcal{D}$) admits user-defined extensions. It is worth noting that, in contrast to other approaches, our formalization does not impose any restriction in the form of adversaries. Adversaries are simply represented as uninterpreted procedures, which gives them the full expressive power of the language.

Building upon previous work on the bisimulation of probabilistic automata, we formalized a notion of program equivalence. Intuitively, we say that two games G_1 and G_2 are observational equivalent with respect to an input set of variables I and an output set of variables O, and we note it $\models \mathsf{G}_1 \simeq_O^I \mathsf{G}_2$, when starting from initial memories that coincide on I, the projection on O of the distributions resulting from the execution of each of the games coincide. The following rule allows then to relate observational equivalence to probability:

$$\frac{m_1 =_I m_2 \quad \models \mathsf{G}_1 \simeq_O^I \mathsf{G}_2 \quad \mathsf{fv}(A) \subseteq O}{\Pr_{\mathsf{G}_1, m_1}[A] = \Pr_{\mathsf{G}_2, m_2}[A]}$$

This notion can be naturally generalized to arbitrary pre and postconditions, which leads to a Probabilistic Relational Hoare Logic. Using these tools we mechanized in one hand most common program transformations as reflection-based Coq tactics, including dead code elimination, constant folding and propagation, procedure call inlining and code movement. The tactics work in a goal oriented manner; for each transformation T, we proved a soundness rule of the form

$$\frac{T(\mathsf{G}_1, \mathsf{G}_2, I, O) = (\mathsf{G}_1', \mathsf{G}_2', I', O') \quad \models \mathsf{G}_1' \simeq_{O'}^{I'} \mathsf{G}_2'}{\models \mathsf{G}_1 \simeq_O^I \mathsf{G}_2}$$

On the other hand, to automate proofs of equivalence with arbitrary pre and postconditions we implemented a relational weakest precondition calculus for probabilistic programs.

Other recurring patterns in game-based proof have been automated to a great extent. Notably, this includes the lazy sampling technique that allows one to defer random choices in games until they are actually needed, or conversely, to make random choices as early as possible, and a syntactic criterion to apply the Fundamental Lemma of Game-Playing that allows one to bound the difference in the probability of some event in two games whose behavior differs only after some particular event happens.

III. Case Studies

The CertiCrypt framework has been put into practice in several case studies. Namely, we have proven the pseudorandom function/pseudroandom permutation switching lemma [1], [3], the semantic security of the ElGamal and Hashed ElGamal encryption schemes in the standard and random oracle models [3], [5], [6], the existential unforgeability of Full-Domain Hash signatures [4], [7], and an improved bound on the exact semantic security of the widely-deployed OAEP scheme [3], [8]. Regarding protocols, we have proven correctness, completeness, and the Honest-Verifier Zero-Knowledge property of several Zero-Knowledge protocols, as well as general theorems allowing to AND/OR compose arbitrary Zero-Knowledge proofs.

IV. Conclusion

By building the framework on top of the Coq proof assistant, we excel in generality and high-assurance. Any conceivable argument could be used in a proof and every hypothesis and side condition has to be made explicit. Furthermore, a proof in CertiCrypt produces a certificate that can be independently and automatically verified using just the highly trustworthy Coq kernel. To ease the construction of proofs, we have mechanized most of the techniques commonly used in game-based reductionist arguments. Still, constructing a proof in CertiCrypt is a time-consuming task that requires a high level of familiarity not only with the tools the framework provides, but with the Coq proof assistant in general as well. We believe this burden could be alleviated by providing a user interface to describe the general structure of a game-based proof, leaving the more involved proof arguments as lemmas to be proven later interactively.

References

[1] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *Advances in Cryptology – EUROCRYPT'06*, ser. Lecture Notes in Computer Science, vol. 4004. Springer-Verlag, 2006, pp. 409–426.

[2] S. Halevi, "A plausible approach to computer-aided cryptographic proofs," Cryptology ePrint Archive, Report 2005/181, 2005.

[3] G. Barthe, B. Grégoire, and S. Zanella Béguelin, "Formal certification of code-based cryptographic proofs," in *Proceedings of the 36th ACM Symposium on Principles of Programming Languages*. ACM Press, 2009, pp. 90–101.

[4] S. Zanella Béguelin, B. Grégoire, G. Barthe, and F. Olmedo, "Formally certifying the security of digital signature schemes," in *30th IEEE Symposium on Security and Privacy, S&P 2009*, 2009.

[5] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin, "Formal certification of ElGamal encryption. A gentle introduction to CertiCrypt," in *5th International Workshop on Formal Aspects in Security and Trust, FAST 2008*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2008.

[6] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, 2004.

[7] J.-S. Coron, "On the exact security of Full Domain Hash," in *Advances in Cryptology*, ser. Lecture Notes in Computer Science, vol. 1880. Springer-Verlag, 2000, pp. 229–235.

[8] V. Shoup, "OAEP reconsidered," in *Proceedings of the 21st Annual International Cryptology Conference*, ser. Lecture Notes in Computer Science, vol. 2139. Springer-Verlag, 2001, pp. 239–259.