Probabilistic relational Hoare logics for computer-aided security proofs

Gilles Barthe[1], Benjamin Grégoire[2], and Santiago Zanella Béguelin[3]

[1] IMDEA Software Institute
[2] INRIA Sophia Antipolis - Méditerranée
[3] Microsoft Research

Provable security The goal of provable security is to verify rigorously the security of cryptographic systems. A provable security argument proceeds in three steps:

1. Define a security goal and an adversarial model;
2. Define the cryptographic system and the security assumptions upon which the security of the system hinges;
3. Show by reduction that any attack against the cryptographic system can be used to build an efficient algorithm that breaks a security assumption.

The provable security paradigm originates from the work of Goldwasser and Micali [10] and plays a central role in modern cryptography. Since its inception, the focus of provable security has gradually shifted towards practice-oriented provable security [4]. The central goal of practice-oriented provable security is to develop and analyze efficient cryptographic systems that can be used for practical purposes, and to provide concrete guarantees that quantify their strength as a function of the values of their parameters (e.g. the key size of a public-key encryption scheme).

The code-based approach [5] realizes the practice-oriented provable security paradigm by means of programming-language techniques and of a systematic way of organizing proofs. In the code-based approach, security hypotheses and goals are cast in terms of the probability of events with respect to distributions induced by probabilistic programs. Typically, proofs that follow the code-based approach adopt some form of imperative pseudocode as a convenient and expressive notation to represent programs (equivalently, games). The pWhile language is a procedural, probabilistic imperative programming language that provides a precise formalism for programs. Commands in pWhile are defined as follows:

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

where $\mathcal{E}$ is a set of expressions, $\mathcal{DE}$ is a set of distribution expressions, and $\mathcal{P}$ is a set of procedures. pWhile distinguishes between concrete procedures,

whose code is defined, and abstract procedures, whose code remains unspecified. Quantification over adversaries in cryptographic proofs is achieved by representing them as abstract procedures parametrized by a set of oracles; these oracles must be instantiated as other procedures in the program.

The meaning of games is defined by a denotational semantics: given an interpretation of abstract procedures, the semantics $[\![c]\!]$ of a game c takes as input an initial memory, i.e. a mapping from variable to values, and returns a sub-distribution on memories. Since we typically consider discrete datatypes, the set $\mathcal{M}$ of memories is discrete and sub-distributions on memories are simply maps $d : \mathcal{M} \to [0,1]$ such that $\sum_{m \in \mathcal{M}} d\ m \leq 1$. We let $\mathcal{D}(\mathcal{M})$ denote the set of sub-distributions over $\mathcal{M}$, and we use the notation $\Pr[c, m : E]$ to denote the probability of the event E in the sub-distribution $[\![c]\!]\ m$.

Proofs in provable security are by reduction. For simplicity, assume that the system under consideration is proved secure under a single assumption. Let $\mathcal{A}$ be an adversary against the security of the system. The goal of the reduction proof is to show that there exists an adversary $\mathcal{B}$ such that the success probability of $\mathcal{A}$ in the attack game is upper bounded by a function of the success probability of $\mathcal{B}$ in breaking the security assumption. A recommended practice is that proofs be constructive, in the sense that the adversary $\mathcal{B}$ is given explicitly as a program that invokes $\mathcal{A}$ as a sub-procedure.

In addition to defining the security goal and hypotheses as probabilistic programs, the code-based approach recommends that proofs are structured as sequences, or trees, of games, so that transitions between two successive games are easier to justify. A typical transition between two games c_1 and c_2, requires establishing an inequality of the form

$$\Pr[c_1, m_1 : A] \leq \Pr[c_2, m_2 : B] + \epsilon \qquad (*)$$

where A and B are events whose probabilities are taken over the sub-distributions $[\![c_1]\!]\ m_1$ and $[\![c_2]\!]\ m_2$ respectively, and ϵ is an arithmetic expression that may depend on the resources allocated to the adversary or, when the transition involves a failure event F, an expression of the form $\Pr[c_i, m_i : F]$. The proof concludes by combining the inequalities proven for each transition to bound the success probability of the reduction.

Verified security Verified security [2, 1] is an emerging approach to security proofs of cryptographic systems. It adheres to the same principles as (practice-oriented) provable security, but revisits its realization from a formal verification perspective. When taking a verified security approach, proofs are mechanically built and verified with the help of state-of-the-art verification tools. The idea of verified security appears in inspiring articles from Halevi [11] and Bellare and Rogaway [5], and is realized by tools such that CertiCrypt [2] and EasyCrypt [1]. Both support the code-based approach, and capture many common reasoning patterns in cryptographic proofs. CertiCrypt and EasyCrypt have been used to verify examples of prominent cryptographic constructions, including encryption schemes, signature schemes, hash function designs, and zero-knowledge proofs. Although they rely

on the same foundations, the two tools have a complementary design: the CertiCrypt framework is entirely programmed and machine-checked in the Coq proof assistant, from which it inherits expressiveness and strong guarantees. In contrast, EasyCrypt implements a verification condition generator that sends proof obligations to SMT solvers, and inherits from them a high degree of automation.

Relational Hoare Logics and liftings It is important to note that inequalities of the form $(*)$ involve two programs, and hence go beyond program verification. The common foundation of CertiCrypt and EasyCrypt is pRHL, a relational logic to reason about probabilistic programs. Its starting point is relational Hoare logic [6], a variant of Hoare logic that reasons about two programs. Judgments of Benton's relational Hoare logic are of the form:

$$\models c_1 \sim c_2 : \Psi \Rightarrow \Phi$$

where c_1 and c_2 are WHILE programs and Ψ and Φ are relations on memories, respectively called the pre-condition and the post-condition. The above judgment is valid if the post-condition is valid for all executions of c_1 and c_2 starting from initial memories that satisfy the pre-condition, i.e. for every pair of initial memories m_1, m_2 such that $m_1 \ \Psi \ m_2$, if the evaluations of c_1 in m_1 and c_2 in m_2 terminate with final memories m_1' and m_2' respectively, then $m_1' \ \Phi \ m_2'$ holds.

Probabilistic relational Hoare logic (pRHL) considers similar judgments

$$\models c_1 \sim c_2 : \Psi \Rightarrow \Phi$$

except that c_1 and c_2 are pWHILE programs. Since the evaluation of a pWHILE program w.r.t. an initial memory yields a sub-distribution over memories, giving a meaning to a pRHL judgment requires interpreting post-conditions as relations over sub-distributions. To this end, pRHL relies on a lifting operator $\mathcal{L}$ which transforms a binary relation into a binary relation on the space of sub-distributions over its underlying sets. Lifting can be used to define the validity of a pRHL judgment: for any two pWHILE programs c_1 and c_2 and relations on memories Ψ and Φ, the judgment $\models c_1 \sim c_2 : \Psi \Rightarrow \Phi$ is valid if for every pair of memories m_1 and m_2, $m_1 \ \Psi \ m_2$ implies $(\llbracket c_1 \rrbracket \ m_1) \, \mathcal{L}(\Phi) \, (\llbracket c_2 \rrbracket \ m_2)$. The ability to derive probability claims from valid pRHL judgments is essential to justify its use as an intermediate tool to prove security of cryptographic systems. Formally, if $\models c_1 \sim c_2 : \Psi \Rightarrow \Phi$ and $\Phi \Rightarrow A\langle 1 \rangle \Rightarrow B\langle 2 \rangle$, then for all memories m_1 and m_2, $m_1 \, \Psi \, m_2$ implies $\Pr[c_1, m_1 : A] \leq \Pr[c_2, m_2 : B]$.

The definition of lifting is adopted from probabilistic process algebra [12]. Let A and B be two discrete sets, and let $R \subseteq A \times B$. The lifting $\mathcal{L}(R)$ of R is the relation on $\mathcal{D}(A) \times \mathcal{D}(B)$ such that, for every sub-distribution d_1 over A and d_2 over B, $d_1 \, \mathcal{L}(R) \, d_2$ if there exists $d \in \mathcal{D}(A \times B)$ such that:

1. for every $(a, b) \in A \times B$, if $d(a, b) > 0$ then $a \ R \ b$
2. for every $a \in A$, $d_1(a) = \sum_{b \in B} d(a, b)$
3. for every $b \in B$, $d_2(b) = \sum_{a \in A} d(a, b)$

The definition of lifting has close connections with the Kantorovich metric (see e.g. [7] for a recent overview), and with flow networks. The relationship with flow networks is best explained pictorially. Figure 1 represents two sub-distributions d_1 (over some set A) and d_2 (over some set B) as a source and a sink respectively. In both cases, the capacity of an edge between an element of the set and the source/sink is the probability of this element; we let p_i denote $d_1(a_i)$ and q_i denote $d_2(b_i)$. Then, let R be a relation between elements of A and elements of B. We use dashed arrows to represent edges (a, b) such that $a\ R\ b$. The definition of lifting requires exhibiting a sub-distribution d, called the witness, such that for every (a, b), if $d(a, b) > 0$ then $a\ R\ b$. In the picture below, it amounts to asserting that d is completely defined by the values of the probabilities $r_1 \ldots r_6$ that are used to decorate the dashed arrows; more precisely, $r_6 = d(a_5, b_5)$, $r_5 = d(a_5, b_4)$, $r_4 = d(a_3, b_4)$, $r_3 = d(a_3, b_3)$, $r_2 = d(a_2, b_2)$ and $r_1 = d(a_1, b_1)$. The remaining constraints can be interpreted as an assertion that d is a maximal flow. Consider that edges are oriented from left to right; then we can define the incoming (resp. outgoing) flow of a node as the sum of the probabilities attached to its incoming (resp. outgoing) edges. Then, constraints 2 and 3 assert that the incoming flow is equal to the outcoming flow for each node; in other words, $d_1\,\mathcal{L}(R)\,d_2$ can be reduced to a maximal flow problem in the network induced by d_1, d_2 and R. The constraints for the maximal flow problem are:

$$\begin{array}{ll} p_5 = r_5 + r_6 & q_5 = r_6 \\ & q_4 = r_4 + r_5 \\ p_3 = r_3 + r_4 & q_3 = r_3 \\ p_2 = r_2 & q_2 = r_2 \\ p_1 = r_1 & q_1 = r_1 \end{array}$$

Any sub-distribution that satisfies the constraints is a witness of the relationship of d_1 and d_2 w.r.t. the lifting of R. Interestingly, the connection between lifting and flow networks opens the possibility of relying on existing flow network algorithms to check whether two distributions are related by lifting.

While pRHL is sufficient for many purposes, a number of cryptographic notions, such as statistical zero-knowledge proofs, require reasoning about approximate equivalence. Recall that the statistical distance between two distributions d_1 and d_2 is defined as

$$\Delta(d_1, d_2) \stackrel{\text{def}}{=} \max_A |d_1\ A - d_2\ A|$$

One can define an extension of pRHL that supports approximate reasoning, via judgments of the form $\models c_2 \sim_\delta \Psi : c_1 \Rightarrow \Phi$, where $\delta \in [0, 1]$. Such a judgment is valid if for every pair of memories m_1 and m_2, $m_1\ \Psi\ m_2$ implies $(\llbracket c_1 \rrbracket\ m_1)\,\mathcal{L}_\delta(\Phi)\,(\llbracket c_2 \rrbracket\ m_2)$, where the approximate lifting $\mathcal{L}_\delta(R)$ of R is defined by the clause $d_1\,\mathcal{L}_\delta(R)\,d_2$ if there exists $d \in \mathcal{D}(A \times B)$ such that:

1. for every $(a, b) \in A \times B$, if $d(a, b) > 0$ then $a\ R\ b$
2. $\pi_1(d) \leq d_1$ and $\Delta(d_1, \pi_1(d)) \leq \delta$
3. $\pi_2(d) \leq d_2$ and $\Delta(d_2, \pi_2(d)) \leq \delta$

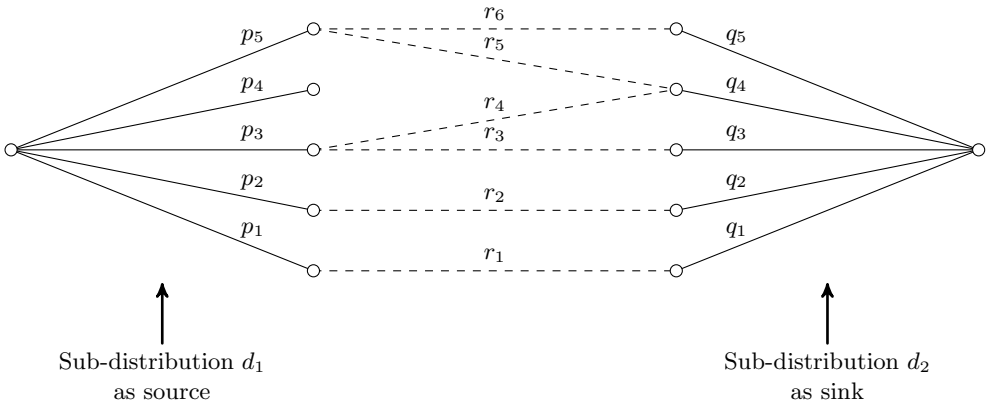

Fig. 1. Lifting as a maximal flow problem

where the sub-distributions $\pi_1(d)$ and $\pi_2(d)$ are defined by the clauses

$$\pi_1(d) = \sum_{b \in B} d\ (a, b) \qquad\qquad \pi_2(d) = \sum_{a \in A} d\ (a, b)$$

and $\leq$ denotes the pointwise extension of inequality on the reals, i.e. $d \leq d'$ iff for every $a \in A$, we have $d\ a \leq d'\ a$. The definition of approximate lifting was also considered in the context of probabilistic algebra [13, 8], and admits a characterization in terms of flow networks.

CertiCrypt and EasyCrypt implement apRHL [3], an extension of pRHL whose judgments are of the form $\models c_2 \sim_{\alpha,\delta} \Psi : c_1 \Rightarrow \Phi$, where $\alpha \geq 1$ and $\delta \in [0, 1]$. Such a judgment is valid if for every pair of memories m_1 and m_2, $m_1\ \Psi\ m_2$ implies $(\llbracket c_1 \rrbracket\ m_1)\ \mathcal{L}_{\alpha,\delta}(\Phi)\ (\llbracket c_2 \rrbracket\ m_2)$, where $\mathcal{L}_{\alpha,\delta}(R)$ denotes the approximate lifting of R. Formally, $d_1\ \mathcal{L}_{\alpha,\delta}(R)\ d_2$ if there exists $d \in \mathcal{D}(A \times B)$ such that:

1. for every $(a, b) \in A \times B$, if $d(a, b) > 0$ then $a\ R\ b$
2. $\pi_1(d) \leq d_1$ and $\Delta_\alpha(d_1, \pi_1(d)) \leq \delta$
3. $\pi_2(d) \leq d_2$ and $\Delta_\alpha(d_2, \pi_2(d)) \leq \delta$

where

$$\Delta_\alpha(d_1, d_2) \stackrel{\text{def}}{=} \max_A(\max\{d_1\ A - \alpha\ (d_2\ A), d_2\ A - \alpha\ (d_1\ A), 0\})$$

This definition coincides with δ-lifting for the case $\alpha = 1$. The resulting logic apRHL allows reasoning about statistical distance between programs and about (computational) differential privacy [9].

References

1. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in*

Cryptology – CRYPTO 2011, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Heidelberg, 2011. Springer.

2. Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2009*, pages 90–101, New York, 2009. ACM.
3. Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic reasoning for differential privacy. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012*. ACM, 2012.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st ACM conference on Computer and Communications Security, CCS 1993*, pages 62–73, New York, 1993. ACM.
5. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, Heidelberg, 2006. Springer.
6. Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In *31st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2004*, pages 14–25, New York, 2004. ACM.
7. Yuxin Deng and Wenjie Du. Logical, metric, and algorithmic characterisations of probabilistic bisimulation. Technical Report CMU-CS-11-110, Carnegie Mellon University, March 2011.
8. Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate analysis of probabilistic processes: Logic, simulation and games. In *5th International Conference on Quantitative Evaluation of Systems, QEST 2008*, pages 264–273. IEEE Computer Society, 2008.
9. Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, ICALP 2006*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12, Heidelberg, 2006. Springer.
10. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
11. Shai Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005.
12. Bengt Jonsson, Wang Yi, and Kim G. Larsen. Probabilistic extensions of process algebras. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 685–710. Elsevier, Amsterdam, 2001.
13. Roberto Segala and Andrea Turrini. Approximated computationally bounded simulation relations for probabilistic automata. In *20th IEEE Computer Security Foundations symposium, CSF 2007*, pages 140–156, 2007.