Computer-Aided Security Proofs for the Working Cryptographer*

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

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

Abstract. We present an automated tool for elaborating security proofs of cryptographic systems from proof sketches—compact, formal representations of the essence of a proof as a sequence of games and hints. Proof sketches are checked automatically using off-the-shelf SMT solvers and automated theorem provers, and then compiled into verifiable proofs in the CertiCrypt framework. The tool supports most commonly used reasoning patterns, is significantly easier to use than its predecessors, and is a plausible candidate for adoption by working cryptographers. We illustrate its application to proofs of the Cramer-Shoup cryptosystem and Hashed ElGamal encryption.

Keywords: Provable security, verifiable security, game-based proofs, Cramer-Shoup, ElGamal.

1 Introduction

The game-playing technique [8, 17, 20] is an established methodology for structuring cryptographic proofs. Its essence lies in giving precise mathematical descriptions of the interaction between an adversary and an oracle system—such descriptions are referred to as games—and to organize proofs as sequences of games, starting from a game that represents a security goal (e.g. IND-CCA), and proceeding to games that represent security assumptions (e.g. DDH) by successive transformations that can be shown to preserve, or only alter slightly the overall security. A typical transition considers two games G and G' and a claim that relates the probability of an event A in game G to the probability of an event A' in game G'. For instance, the claim may be an inequality of the form $\Pr[G : A] \leq \Pr[G' : A'] + \Delta$, where Δ is an arithmetic expression that depends on the number of adversary calls to oracles. The prevailing practice for proving the validity of such claims is to use standard mathematical tools; in general, proofs interleave reasoning about the semantics of games with information-theoretic or mathematical arguments.

The code-based approach [8, 17] is an instance of the game-playing technique in which games are cast as probabilistic algorithms. The adoption of programming idioms allows giving precise definitions of games, and paves the way for applying programming language methods to justify proof steps rigorously. As anticipated by their proponents, code-based game-playing proofs are amenable to formal verification, and a number of tools provide support for building them. CryptoVerif [11] is a prominent tool for conducting security proofs in a game-based setting. Technically, games in CryptoVerif are modelled as processes, and transitions are justified by means of process-algebraic concepts such as bisimulations. One strength of CryptoVerif,

* This article reports on a tool not yet made publicly available. A copy of the tool and the code of the applications described here may be obtained from the authors upon request.

apart from being the first tool to have supported game-based proofs, is to apply both to protocols and primitives; for instance, it has been successfully applied to verify Kerberos [10] and the FDH signature scheme [12]. CertiCrypt [4] is another framework that allows for the interactive construction of game-based proofs in the Coq proof assistant [24]. One specificity of CertiCrypt is that proofs can be verified independently and automatically by a small trustworthy checker; it has been successfully applied to verify prominent cryptographic constructions, including OAEP [6], FDH [25], and zero-knowledge protocols [7].

While the developments based on CryptoVerif and CertiCrypt make a convincing case that computer-aided cryptographic proofs are indeed plausible, neither tool has reached a wide cryptographic audience. In [6], the authors of CertiCrypt contrast the high guarantees given by their tool with the effort and expertise required to build machine-checked proofs, and conclude that cryptographers are unlikely to adopt verifiable security in its current form. In this sense, it can be considered that CryptoVerif and CertiCrypt only provide a partial realization of Halevi's programme of systematically building computer-aided cryptographic proofs [17].

The thesis of this article is that verifiable security can dramatically benefit from automation using state-of-the-art verification technology, and that verifiable game-based proofs can be constructed with only a moderate effort. The thesis is realized with the presentation of EasyCrypt, an automated tool that builds machine-checked proofs from *proof sketches*, which offer a machine-processable representation of the essence of a security proof. More fundamentally, we argue that EasyCrypt is significantly easier to use than previous tools, making an important step towards the adoption of computer-aided security proofs by working cryptographers and hence towards fulfilling Halevi's programme. To substantiate our claim, we present computer-aided proofs of security of Hashed ElGamal and the Cramer-Shoup cryptosystem.

EasyCrypt adopts the principled approach mandated by CertiCrypt to conduct game-based proofs and imposes a clear separation between program verification and information-theoretic reasoning. Hence, game transitions are justified in two steps: first, one proves logical relations between the games using probabilistic Relational Hoare Logic (pRHL); second, one applies information-theoretic reasoning to derive the claim from pRHL judgments. We provide for each step highly effective mechanisms that build upon a combination of off-the-shelf and purpose-specific tools. Specifically, EasyCrypt implements an automated procedure that computes for any pRHL judgment, a set of sufficient conditions, known as verification conditions, for its validity. The outstanding feature of this procedure, and the key to the effectiveness of EasyCrypt, is that verification conditions are expressed in the language of first-order logic, without any mention of probability, and can be discharged automatically by state-of-the-art tools such as SMT solvers and theorem provers. The verification condition generator is *proof-producing*, in the sense that it generates Coq files that can be machine-checked using the CertiCrypt framework. Moreover, the connection to CertiCrypt makes it possible to benefit from the expressivity and flexibility of a general-purpose proof assistant for advanced verification goals that fall out of the scope of automated techniques. Additionally, EasyCrypt implements an automated mechanism for proving claims. The mechanism combines some elementary rules to compute (bounds on) probabilities of events—e.g. the probability of a uniformly sampled element to belong to a list—with rules to derive (in)equalities between probabilities of events in games from judgments in pRHL. The combination of these tools with other more mundane features such as some limited form of specification inference provides substantial leverage towards making verifiable security practical and makes EasyCrypt a plausible candidate for adoption by the working cryptographer.

2 Introductory Example: Hashed ElGamal Encryption

This section illustrates the application of EasyCrypt to the IND-CPA security of the Hashed ElGamal encryption scheme in the Random Oracle Model. The example serves to introduce the notion of proof sketch and to give the reader an idea of the input that the tool expects. It also allows for a preliminary comparison between EasyCrypt and CertiCrypt. We refer the reader to [5] for a proof of the same result in CertiCrypt.

Hashed ElGamal is a variant of ElGamal encryption that does not require plaintexts to be elements of a group G. Instead, plaintexts are bitstrings of a certain length k and group elements are mapped into bitstrings using a hash function $H : G \to \{0,1\}^k$. Formally, the scheme is defined by the following triple of algorithms:

$$\begin{array}{lcl}
\mathcal{KG}(\eta) & \stackrel{\text{def}}{=} & hk \stackrel{\$}{\leftarrow} K;\ x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ \mathsf{return}\ ((hk, g^x), (hk, x)) \\
\mathcal{E}((hk, \alpha), m) & \stackrel{\text{def}}{=} & y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ h \leftarrow H(hk, \alpha^y);\ \mathsf{return}\ (g^y, h \oplus m) \\
\mathcal{D}((hk, x), (\beta, \zeta)) & \stackrel{\text{def}}{=} & h \leftarrow H(hk, \beta^x);\ \mathsf{return}\ (\zeta \oplus h)
\end{array}$$

The IND-CPA security of Hashed ElGamal can be reduced to the Computational Diffie-Hellman (CDH) assumption on the underlying group family $\{G_\eta\}$; to match the existing proof in CertiCrypt, we exhibit a reduction to the LCDH assumption, the *set* version of the CDH assumption—the reduction from LCDH to CDH is immediate.

Figure 1 provides the sequence of games used to justify the reduction. This is an essential part of the proof sketch that is input to EasyCrypt, and which is composed of five ingredients:[3]

1. Type, constant and operator declarations, which introduce the objects manipulated by the scheme. Here, they include a cyclic group, the group law and exponentiation, and exclusive or on bitstrings;
2. Axioms, which capture mathematical properties of these objects, and are used by automated tools to check the validity of the proof sketch. Here, the axioms state properties of cyclic groups and of exclusive or on bitstrings;
3. Game definitions, where adversaries are specified as abstract procedures with access to oracles. In all the games in the figure the hash function H is modeled as a random oracle (we omit hash keys since they play no role in the proof):

$$H(x) \stackrel{\text{def}}{=} \mathsf{if}\ x \notin \mathsf{dom}(\boldsymbol{L})\ \mathsf{then}\ h \stackrel{\$}{\leftarrow} \{0,1\}^k; \boldsymbol{L}[x] \leftarrow h\ \mathsf{fi}; \mathsf{return}\ \boldsymbol{L}[x]$$

   The adversary is given access to a wrapper $H^{\mathcal{A}}$ of this oracle that just stores queries in a list $\boldsymbol{L}^{\mathcal{A}}$ before forwarding them to H.
4. Judgments in pRHL. The general form of judgments is $\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$, where G_1 and G_2 are games, and the pre-condition Ψ and the post-condition Φ are relations on states. The pre- and post-conditions are first-order formulae built from relational expressions, in which language expressions are tagged with $\langle 1 \rangle$ or $\langle 2 \rangle$ to denote their interpretation in the first or second game. Pre- and post-conditions often consider the equivalence of memories on a set X of variables; we use $=_X$ as a shorthand for the conjunction $\bigwedge_{x \in X} x\langle 1 \rangle = x\langle 2 \rangle$;

[3] The first two are omitted from the figure. We include an extract of the actual input file for reference in Appendix A.

5. Relations between probabilities, built from probability quantities (the probability of an event in a game), arithmetic operators, and mathematical relations (e.g. $=, <, \leq$). The final statement that expresses the overall security guarantee brought by the proof sketch is usually a claim that upper bounds the probability of adversary success in an initial attack game in terms of the probabilities of one or more adversaries breaking security assumptions.

We briefly comment on the sequence of games: the first and last games encode the IND-CPA and LCDH experiments, respectively. The first transition from IND-CPA to G_1 inlines the key generation and encryption procedures and rearranges the resulting code so that random choices are made upfront. The pRHL judgment states that both games yield identical distributions on res and $\boldsymbol{L}^{\mathcal{A}}$. The keyword res denotes the result of a game. In all games in the figure it is defined as the result of the comparison $(b = b')$, except in the last one where the result is $(g^{xy} \in L)$. The transition to G_2 substitutes the call $H(\hat{y})$ by a random sampling. This only makes a difference if $\mathcal{A}_1$ made this call, and this happens with the same probability in either game. Thus, the difference in the probability of Success in the games is bounded by the probability of Bad in G_2 (these two events are defined on the top of the figure). This can be seen as a semantic variant of the Fundamental Lemma; the logic allows to dispense with the code instrumentation needed in the syntactic counterpart. The transition from G_2 to G_3 uses a code transformation known as *optimistic sampling*: instead of sampling h and defining a value γ as $h \oplus m_b$, one can sample γ and define $h = \gamma \oplus m_b$. This transformation is proven admissible within the logic and removes the dependency of the adversary's output from b. The final transition performs the reduction to LCDH by exhibiting an adversary $\mathcal{B}$ that uses $\mathcal{A}$ as a sub-procedure and for which the semantics of the LCDH game and G_3 coincide. Finally, from the preceding claims, the advantage of $\mathcal{A}$ can be bounded by the probability of $\mathcal{B}$ in solving LCDH. The resulting proof sketch is about 250 lines long, about 6 times shorter than the proof in CertiCrypt reported in [5]—and arguably much simpler and close to a pen-and-paper proof.

3 An Overview of EasyCrypt

Programming Language Games are modelled as programs in a typed, probabilistic, procedural, imperative language. Adversaries are modelled as abstract procedures, for which one can provide an interface that specifies the variables it can read and write, and the procedures it may call. Games can be given a semantics as distribution transformers, in the style of [4]. Formally, the semantics of a game G is a map, denoted $[\![\mathsf{G}]\!]$, that returns for an initial memory m the (sub-)distribution on final memories resulting from executing G in m. Given an initial memory m and an event A, we let $\Pr[\mathsf{G}, m : A]$ denote the probability of A w.r.t. the distribution $[\![\mathsf{G}]\!]\ m$; we simply write $\Pr[\mathsf{G} : A]$ when the initial memory is not relevant.

A Mechanized Probabilistic Relational Hoare Logic We start by reviewing the essentials of pRHL. The validity of a judgment $\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$ is formally cast as a max-flow min-cut problem. For our purposes, it will be sufficient to know that, if $\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$, then $\Pr[\mathsf{G}_1, m_1 : A] = \Pr[\mathsf{G}_2, m_2 : B]$ for all memories m_1 and m_2 such that $m_1\ \Psi\ m_2$, and events A and B such that $\Phi \to (A\langle 1\rangle \leftrightarrow B\langle 2\rangle)$.

$\mathsf{Success} \overset{\text{def}}{=} b = b' \wedge |\boldsymbol{L}^{\mathcal{A}}| \leq q_{\mathsf{H}}$ $\qquad$ $\mathsf{Bad} \overset{\text{def}}{=} \hat{y} \in \boldsymbol{L}^{\mathcal{A}}$

```
Game IND-CPA :
  $(\alpha, x) \leftarrow \mathcal{KG}(\,)$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $(\beta, \gamma) \leftarrow \mathcal{E}(\alpha, m_b)$;
  $b' \leftarrow \mathcal{A}_2(\beta, \gamma)$
```

$\longrightarrow$

```
Game G_1 :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \leftarrow H(\hat{y})$;
  $b' \leftarrow \mathcal{A}_2(g^y, h \oplus m_b)$
```

$\models \mathsf{IND\text{-}CPA} \sim \mathsf{G}_1 : \mathsf{true} \Rightarrow =_{\{\mathsf{res}, \boldsymbol{L}^{\mathcal{A}}\}}$
$\Pr[\mathsf{IND\text{-}CPA} : \mathsf{Success}] = \Pr[\mathsf{G}_1 : \mathsf{Success}]$

```
Game G_1 :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \leftarrow H(\hat{y})$;
  $b' \leftarrow \mathcal{A}_2(g^y, h \oplus m_b)$
```

$\longrightarrow$

```
Game G_2 :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \xleftarrow{\$} \{0,1\}^k$;
  $b' \leftarrow \mathcal{A}_2(g^y, h \oplus m_b)$
```

$\models \mathsf{G}_1 \sim \mathsf{G}_2 : \mathsf{true} \Rightarrow \mathsf{Bad}\langle 1\rangle \leftrightarrow \mathsf{Bad}\langle 2\rangle \wedge \left(\neg \mathsf{Bad}\langle 1\rangle \rightarrow =_{\{\mathsf{res}, \boldsymbol{L}^{\mathcal{A}}\}}\right)$
$|\Pr[\mathsf{G}_1 : \mathsf{Success}] - \Pr[\mathsf{G}_2 : \mathsf{Success}]| \leq \Pr[\mathsf{G}_2 : \mathsf{Bad}]$

```
Game G_2 :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \xleftarrow{\$} \{0,1\}^k$;
  $b' \leftarrow \mathcal{A}_2(g^y, h \oplus m_b)$
```

$\longrightarrow$

```
Game G_3 :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $\gamma \xleftarrow{\$} \{0,1\}^k$;
  $b' \leftarrow \mathcal{A}_2(g^y, \gamma)$;
  $b \xleftarrow{\$} \{0,1\}$
```

$\models \mathsf{G}_2 \sim \mathsf{G}_3 : \mathsf{true} \Rightarrow =_{\{\mathsf{res}, \boldsymbol{L}^{\mathcal{A}}, \hat{y}\}}$
$\Pr[\mathsf{G}_2 : \mathsf{Success}] = \Pr[\mathsf{G}_3 : \mathsf{Success}] = 1/2$
$\Pr[\mathsf{G}_2 : \mathsf{Bad}] = \Pr[\mathsf{G}_3 : \mathsf{Bad}]$

```
Game G_3 :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $\gamma \xleftarrow{\$} \{0,1\}^k$;
  $b' \leftarrow \mathcal{A}_2(g^y, \gamma)$;
  $b \xleftarrow{\$} \{0,1\}$
```

$\longrightarrow$

```
Game LCDH :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ y \xleftarrow{\$} \mathbb{Z}_q$;
  $L \leftarrow B(g^x, g^y)$
Adversary $\mathcal{B}(\alpha, \beta)$ :
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $\gamma \xleftarrow{\$} \{0,1\}^k$;
  $b' \leftarrow \mathcal{A}_2(\beta, \gamma)$;
  return $\boldsymbol{L}^{\mathcal{A}}$
```

$\models \mathsf{G}_3 \sim \mathsf{LCDH} : \mathsf{true} \Rightarrow \boldsymbol{L}^{\mathcal{A}}\langle 1\rangle = L\langle 2\rangle \ \wedge\ \mathsf{Bad}\langle 1\rangle \leftrightarrow \mathsf{res}\langle 2\rangle$
$\Pr\left[\mathsf{G}_3 : \hat{y} \in \boldsymbol{L}^{\mathcal{A}} \wedge |\boldsymbol{L}^{\mathcal{A}}| \leq q_{\mathsf{H}}\right] = \Pr[\mathsf{LCDH} : g^{xy} \in L \wedge |L| \leq q_{\mathsf{H}}]$
$\left|\Pr[\mathsf{IND\text{-}CPA} : \mathsf{Success}] - \frac{1}{2}\right| \leq \Pr[\mathsf{LCDH} : g^{xy} \in L \wedge |L| \leq q_{\mathsf{H}}]$

Fig. 1. Proof sketch of Hashed ElGamal security

The rules of pRHL are either one-sided, or two-sided. One-sided rules operate on a single command, and do not impose any restriction on the other command of the concluding judgment. In contrast, two-sided rules require that the two commands of the concluding judgment are of the same shape. For instance, the respective forms of the conclusions for the one-sided and two-sided rules for assignments are $\models x \leftarrow e_1 \sim c_2 : \Psi \Rightarrow \Phi$ and $\models x \leftarrow e_1 \sim x \leftarrow e_2 : \Psi \Rightarrow \Phi$ where c_2 is an arbitrary command. Thus, one-sided rules are more flexible, and allow to reason about programs that do not have the same shape. All language constructs admit both one-sided and two-sided rules, except for random assignments and adversary calls, for which only two-sided rules exist.

The lack of one-sided rules for random assignments and adversary calls limits the applicability of the logic: e.g., it cannot relate the programs $x \xleftarrow{\$} X; y \leftarrow \mathcal{A}(z)$ and $y \leftarrow \mathcal{A}(z); x \xleftarrow{\$} X$, because instructions are executed in a different order. To mitigate this limitation, EasyCrypt implements program transformations for code motion. For instance, the judgment

$$\models x \xleftarrow{\$} X; y \leftarrow \mathcal{A}(z) \sim y \leftarrow \mathcal{A}(z); x \xleftarrow{\$} X : \mathsf{true} \Rightarrow x\langle 1\rangle = x\langle 2\rangle$$

can be shown valid by first applying code motion to swap the statements in one of the programs, and then the two-sided rules of the logic for random assignments and adversary calls.

EasyCrypt implements a tactic language to prove the validity of a pRHL judgment using rules of pRHL and program transformations. The language includes: i. basic tactics for each rule of pRHL; ii. tactics for inlining, code movement, and eager/lazy sampling; iii. tacticals to build more powerful tactics by composition of basic ones. The tactic language provides the necessary infrastructure for making most components of EasyCrypt proof-producing, as discussed below.

Generating Verification Conditions Most practical verification tools adopt a similar methodology: first, one computes from a program and its specification a set of sufficient conditions, known as verification conditions. Second, the conditions are discharged by automated tools. Extending the methodology to pRHL is a significant challenge, for two reasons: i. generating verification conditions for a relational program logic is an open topic of research; ii. there is no prior application of the methodology to probabilistic programs.

There are at least two possible strategies for generating verification conditions in a relational setting. One can either define a relational weakest precondition (wp) calculus that operates on both games in lockstep, in the style of two-sided rules, or else one can apply a weakest precondition calculus on each game separately, in the style of one-sided rules and self-composition [3]. Both strategies are incomplete: the relational wp calculus inherits the restrictions of pRHL, whereas self-composition simply fails to handle random assignments and adversary calls. In order to circumvent these limitations, EasyCrypt implements an alternative strategy to prove pRHL judgments:

1. Calls to non-adversary procedures are eliminated from the games by successive applications of inlining. Whenever the transformation terminates successfully, only adversary calls remain;
2. Random assignments are moved upfront, i.e. to the beginning of the code of the main procedure, or of oracles. If the transformation succeeds, the resulting code consists of a

sequence of random assignments followed by deterministic code, possibly with adversary calls;
3. A relational wp calculus is applied to the deterministic code in the game, using relational specifications of adversaries to deal with calls. Each adversary specification induces a proof obligation, expressed as a pRHL judgment, on the oracles it calls. Self-composition is applied to verify the deterministic code of the oracles with respect to these pRHL judgments.

This approach is clearly incomplete, and would fail on programs with recursive procedures or random assignments in loops, or where calls to adversaries appear in a different order. Pleasingly, the strategy is extremely effective in practice—so that we have found no need to implement alternatives for dealing with programs not handled by our approach.

At this point, all judgments are of the form $\models S_1 \sim S_2 : \Psi \Rightarrow \Phi$, where S_1 and S_2 are sequences of random assignments. The final step for generating proof obligations is turning these judgments into logical formulae. Take

$$\begin{array}{lcl} S_1 & \stackrel{\text{def}}{=} & x_1 \stackrel{\$}{\leftarrow} T_1; \ldots x_l \stackrel{\$}{\leftarrow} T_l \\ S_2 & \stackrel{\text{def}}{=} & y_1 \stackrel{\$}{\leftarrow} U_1; \ldots y_n \stackrel{\$}{\leftarrow} U_n \end{array}$$

Then, $\models S_1 \sim S_2 : \Psi \Rightarrow \Phi$, provided there exists a 1-1 mapping

$$f : T_1 \times \cdots \times T_l \to U_1 \times \cdots \times U_n$$

such that the verification condition $\Phi \Rightarrow_f \Psi$, defined as

$$\forall m_1\ m_2\ \vec{t}\ .\ m_1\ \Psi\ m_2 \implies m_1 \left\{\vec{t}/\vec{x}\right\}\ \Phi\ m_2 \left\{f(\vec{t})/\vec{y}\right\}$$

is valid.[4] In practice, reordering the sequences of statements S_1 and S_2 lexicographically according to the names of the variables and taking f as the identity often works. However, other mappings must sometimes be used, for instance to justify transitions based on one-time paddings or that use different variable names. Hence, EasyCrypt offers a tactic to provide the mapping f that justifies the equivalence of a sequence of random assignments. Note that, for any f explicitly given, $\Phi \Rightarrow_f \Psi$ is a first-order formula whose validity can be established by off-the-shelf tools. In order to target multiple tools, EasyCrypt generates its verification conditions in the intermediate format of the Why tool [16]. We then use the Simplify prover [15], the alt-ergo SMT solver [13], and the Coq proof assistant to discharge the conditions (although many others provers are supported).

Reasoning about Failure Events Game-based proofs often include transitions in which it is argued that two games G_1 and G_2 behave identicaly unless a designated failure event F occurs. Such transitions are justified using the so-called Fundamental Lemma [8, 20], which allows to bound the difference between the probability of an event A in game G_1 and a possibly different event B in game G_2 by the probability of F in either game. Although a syntactical characterization of this lemma is often used, in which the failure event is represented by a Boolean flag in the code of the games, we can pleasingly state a more general version of the lemma using relational logic.

[4] There exist more general conditions not requiring $T_1 \times \cdots \times T_l$ and $U_1 \times \cdots \times U_n$ to be isomorphic. See Appendix B for a justification of the method.

Lemma 1 (Fundamental Lemma). *Let* G_1*,* G_2 *be two games and* A, B*, and* F *be events such that*

$$\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow (F\langle 1\rangle \leftrightarrow F\langle 2\rangle) \wedge (\neg F\langle 1\rangle \rightarrow (A\langle 1\rangle \leftrightarrow B\langle 2\rangle))$$

Then, if $m_1\ \Psi\ m_2$*,*

1. $\Pr[\mathsf{G}_1, m_1 : A \wedge \neg F] = \Pr[G_2, m_2 : B \wedge \neg F]$,
2. $|\Pr[\mathsf{G}_1, m_1 : A] - \Pr[\mathsf{G}_2, m_2 : B]| \leq \Pr[\mathsf{G}_1, m_1 : F] = \Pr[\mathsf{G}_2, m_2 : F]$

The hypotheses of the lemma can be checked using the pRHL prover. The key to proving the validity of the judgment is finding an appropriate specification for adversaries. EasyCrypt infers for each adversary call $x \leftarrow \mathcal{A}(\vec{t})$ a relation Θ and checks the validity of the judgment

$$\begin{aligned}\models \mathcal{A} \sim \mathcal{A} : (\neg F\langle 1\rangle \wedge \neg F\langle 2\rangle \wedge =_{\mathsf{args}(\mathcal{A})} \wedge \Theta) \Rightarrow \\ (F\langle 1\rangle \leftrightarrow F\langle 2\rangle) \wedge \left(\neg F\langle 1\rangle \rightarrow =_{\{\mathsf{res}\}} \wedge \Theta\right)\end{aligned}$$

where $\mathsf{args}(\mathcal{A})$ denotes the set of parameters of $\mathcal{A}$. This in turn, requires inferring and checking similar specifications for oracles. Although these heuristically inferred specifications suffice in most cases, the user can choose to provide their own specifications for one or more oracles or adversaries when needed, leaving the tool to infer the rest.

Computing Probabilities The tool can prove claims about the probability of events in games from previously proved relational judgments, arithmetic laws, and properties of probability (e.g. inclusion-exclusion principle) by exploiting the following rules:

$$\frac{m_1\ \Psi\ m_2 \quad \models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi \quad \Phi \rightarrow (A\langle 1\rangle \leftrightarrow B\langle 2\rangle)}{\Pr[\mathsf{G}_1, m_1 : A] = \Pr[\mathsf{G}_2, m_2 : B]}$$

$$\frac{m_1\ \Psi\ m_2 \quad \models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi \quad \Phi \rightarrow (A\langle 1\rangle \rightarrow B\langle 2\rangle)}{\Pr[\mathsf{G}_1, m_1 : A] \leq \Pr[\mathsf{G}_2, m_2 : B]}$$

EasyCrypt also implements a simple mechanism for computing probability bounds. For instance, it can establish that the probability that a value uniformly chosen from a set T is equal to an arbitrary expression is $1/|T|$, or the probability it belongs to a list of n values is at most $n/|T|$.

Generating Verifiable Evidence EasyCrypt implements a compiler that turns proof sketches into Coq files that are compatible with the CertiCrypt framework and can be verified using the type checker of Coq. The compiler serves two purposes: first, it increases confidence in proof sketches significantly, by producing independently verifiable proofs, and by providing means of checking the consistency of the set of axioms used in a proof sketch. Second, it opens the possibility to conduct in a general-purpose proof assistant proof steps that fall out of the scope of automated methods.

We briefly describe the workings of the compiler. The declarations, definitions of games, and axioms of a proof sketch admit an immediate translation into CertiCrypt. The recommended practice is to prove the axioms used by EasyCrypt in CertiCrypt. In most cases, the axioms already exist in CertiCrypt, or are simple consequences of proven facts. Then, using the proof-producing option of the pRHL prover, all pRHL judgments of a proof sketch are compiled into pRHL derivations in CertiCrypt. In order to make the output derivations checkable by the Coq proof assistant, the compiler also generates a prelude which declares as axioms the verification conditions computed by the prover. Although these verification conditions have been proved using SMT solvers, they must be treated as axioms because there is currently no mechanism that generates Coq proofs from successful runs of SMT solvers—making SMT solvers proof-producing is an active subject of research [23], and advances towards this goal shall benefit immediately to EasyCrypt. As for axioms, there is the possibility of proving the verification conditions in Coq.

Finally, the compiler generates for each claim in a proof sketch a Coq lemma that must be completed manually with justifications of the probability reasoning performed by EasyCrypt. The generated file is a proof skeleton rather than a machine-checked proof: not all reasonings are justified—for instance, the computation of $\Pr[\mathsf{G}_3 : \mathsf{Success}]$ in the proof of Hashed ElGamal in Section 2 is not proof-producing. Extending the compiler to certify such steps is entirely feasible and we plan to do so in the near future.

4 Advanced Application: Cramer-Shoup Cryptosystem

The Cramer-Shoup cryptosystem is a public-key encryption scheme based on ElGamal encryption that gained fame for being the first efficient asymmetric encryption scheme to be proven secure against adaptive chosen-ciphertext attacks under standard assumptions—the length of ciphertexts is just twice the length of ElGamal ciphertexts. Given a cyclic group (family) G of order q and a keyed hash function $H : G^3 \to \mathbb{Z}_q$ mapping triples of group elements into integers in $\mathbb{Z}_q$, key generation, encryption, and decryption are defined as follows:

```
$\mathcal{KG}(\eta)$ :
  $g, \hat{g} \xleftarrow{\$} G \setminus \{1\}$;
  $x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q$;  $hk \xleftarrow{\$} K$;
  $e \leftarrow g^{x_1}\hat{g}^{x_2}$;  $f \leftarrow g^{y_1}\hat{g}^{y_2}$;  $h \leftarrow g^{z_1}\hat{g}^{z_2}$;
  $pk \leftarrow (hk, g, \hat{g}, e, f, h)$;
  $sk \leftarrow (hk, g, \hat{g}, x_1, x_2, y_1, y_2, z_1, z_2)$;
  return $(pk, sk)$
```

```
$\mathcal{E}((hk, g, \hat{g}, e, f, h), m)$ :
  $u \xleftarrow{\$} \mathbb{Z}_q$;  $a \leftarrow g^u$;  $\hat{a} \leftarrow \hat{g}^u$;  $c \leftarrow h^u \cdot m$;
  $v \leftarrow H(hk, a, \hat{a}, c)$;  $d \leftarrow e^u \cdot f^{uv}$;
  return $(a, \hat{a}, c, d)$

$\mathcal{D}((hk, g, \hat{g}, x_1, x_2, y_1, y_2, z_1, z_2), (a, \hat{a}, c, d))$ :
  $v \leftarrow H(hk, a, \hat{a}, c)$;
  if $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then
    return $c/(a^{z_1} \cdot \hat{a}^{z_2})$
  else return $\perp$
```

Cramer-Shoup can be proven secure against adaptive chosen-ciphertext attacks (IND-CCA secure) in the standard model assuming the DDH problem is hard in the group family $\{G_\eta\}$ and the hash function H is target-collision-resistant (TCR).

Definition 1 (CCA-advantage). *Let $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme. The CCA-advantage of an adversary $\mathcal{A}$ limited to $q_\mathcal{D}$ decryption queries against the adaptive chosen-ciphertext security of the scheme is defined as*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathit{CCA}}(q_\mathcal{D}) \overset{def}{=} \left|\Pr\left[\mathit{IND\text{-}CCA} : b = b'\right] - 1/2\right|$$

where the experiment IND-CCA is formally defined by means of the following game:

```
Game INDCCA :
  $(pk, \boldsymbol{sk}) \leftarrow \mathcal{KG}(\ )$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $\boldsymbol{\gamma}^* \leftarrow \mathcal{E}(pk, m_b)$;  $\boldsymbol{\gamma}^*_{\mathsf{def}} \leftarrow$ true;
  $b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$
```

```
Oracle $\mathcal{D}^{\mathcal{A}}(\gamma)$ :
  if $|\boldsymbol{L}_\mathcal{D}| < q_\mathcal{D} \wedge \neg(\boldsymbol{\gamma}^*_{\mathsf{def}} \wedge \gamma = \boldsymbol{\gamma}^*)$
  then $\boldsymbol{L}_\mathcal{D} \leftarrow \gamma :: \boldsymbol{L}_\mathcal{D}$;  return $\mathcal{D}(\boldsymbol{sk}, \gamma)$
  else return $\perp$
```

Theorem 1 (Adaptive Security of Cramer-Shoup). *Let $\mathcal{A}$ be an adversary against the $\mathsf{IND\text{-}CCA}$ security of Cramer-Shoup limited to $q_{\mathcal{D}}$ decryption queries. Then, there exists an algorithm $\mathcal{B}$ for solving the DDH problem in G and an adversary $\mathcal{C}$ against the TCR of the hash function H s.t.*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathsf{CCA}}(q_{\mathcal{D}}) \leq \mathbf{Adv}^{\mathcal{B}}_{\mathsf{DDH}} + \mathbf{Succ}^{\mathcal{C}}_{\mathsf{TCR}} + \frac{q_{\mathcal{D}}^4}{q^4} + \frac{q_{\mathcal{D}}+2}{q}$$

Proof. Figure 2 shows a proof sketch of the above theorem in $\mathsf{EasyCrypt}$. The proof follows closely the one presented in [17]; we give only a high-level description here. Game G_1 in the figure is obtained directly from the $\mathsf{IND\text{-}CCA}$ game instantiated for Cramer-Shoup by inlining the definitions of the key generation and encryption procedures, propagating assignments, and replacing expressions by equivalent ones. It is worth noticing that all verification conditions that ensure the validity of this transformation can be discharged automatically using an SMT—this surpasses Halevi's expectations [17], who suggested this transformation be split in three steps so that it could be handled by an automated tool. We then build a DDH distinguisher $\mathcal{B}$ such that the output distribution on the value of $(b = b')$ is identical in games DDH_0 (where $\mathcal{B}$ receives valid DDH triples) and G_1, on the one hand, and in games DDH_1 (where $\mathcal{B}$ receives random triples) and G_2, on the other. In addition, we instrument the decryption oracle in G_2 to raise a flag **bad** whenever $\mathcal{A}$ queries for the decryption of a valid ciphertext with $\log_a \hat{a} \neq \log_g \hat{g}$. We then show using our semantic characterization of the Fundamental Lemma that the difference in the probability of $(b = b')$ in this game and in game G_3, where $\mathcal{D}$ rejects such ciphertexts, is bounded by the probability of **bad** in the latter game—we also change the way e, f and h are computed in a semantics-preserving way. Up to this point, by the triangular inequality we have

$$\left|\Pr\left[\mathsf{IND\text{-}CCA} : b = b'\right] - \Pr\left[\mathsf{G}_3 : b = b'\right]\right| \leq \mathbf{Adv}^{\mathcal{B}}_{\mathsf{DDH}} + \Pr\left[\mathsf{G}_3 : \mathbf{bad}\right]$$

The next game in the sequence, G_4, removes the dependency of the adversary's output from bit b by choosing a random r and setting $c = g^r$. This requires to be able to compute z_2 from $\log_g(c) = uz + (u - u')wz_2 + \log_g(m_b)$, which is not possible if $u = u'$, but this happens only with probability $1/q$. We use again the semantic formulation of the Fundamental Lemma to bound the difference in the probability of $(b = b')$ between G_3 and G_4 by $1/q$. After straightforward information-theoretic reasoning we get

$$\left|\Pr\left[\mathsf{IND\text{-}CPA} : b = b'\right] - 1/2\right| \leq \mathbf{Adv}^{\mathcal{B}}_{DDH} + 2/q + \Pr\left[\mathsf{G}_4 : \mathbf{bad} \wedge u \neq u'\right]$$

We can now move most of the code of the game before the call to $\mathcal{A}_1$. This in turn allows to make d random by uniformly choosing $r' = \log_g(d)$ and defining x_2 in terms of it, rather than the other way around. Since now the game computes the challenge ciphertext in advance, we can instrument $\mathcal{D}$ to raise a flag $\mathbf{bad}_1$ when the challenge is queried during the first phase of the game. Note that at this point the challenge ciphertext is a 4-tuple of uniformly random elements, therefore, this probability is bounded by $(q_{\mathcal{D}}/q)^4$—this is achieved by means of an intermediate game, not shown in the figure, that stores the 4 components of queried ciphertexts in different lists, and by independently bounding the probability of each component of the challenge appearing in the corresponding list. Hence, we have

$$\Pr\left[\mathsf{G}_4 : \mathbf{bad} \wedge u \neq u'\right] \leq \Pr\left[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u'\right] + (q_{\mathcal{D}}/q)^4$$

```
Game $\mathsf{G}_1$ :
  $g, \hat{g} \xleftarrow{\$} G \setminus \{1\}; x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q;$
  $hk \xleftarrow{\$} K;$
  $e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^{z_1}\hat{g}^{z_2};$
  $(m_0, m_1) \leftarrow \mathcal{A}_1(hk, g, \hat{g}, e, f, h);\ b \xleftarrow{\$} \{0,1\};$
  $u \xleftarrow{\$} \mathbb{Z}_q;\ a \leftarrow g^u;\ \hat{a} \leftarrow \hat{g}^u;\ c \leftarrow a^{z_1} \cdot \hat{a}^{z_2} \cdot m_b;$
  $v \leftarrow H(hk, a, \hat{a}, c);\ d \leftarrow a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2};$
  $\boldsymbol{\gamma}^* \leftarrow (a, \hat{a}, c, d);\ \boldsymbol{\gamma}^*_{\mathsf{def}} \leftarrow \mathsf{true};\ b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$
```

```
Oracle $\mathcal{D}(a, \hat{a}, c, d)$ :
  if $|\boldsymbol{L}_{\mathcal{D}}| < q_{\mathcal{D}} \wedge \neg(\boldsymbol{\gamma}^*_{\mathsf{def}} \wedge (a, \hat{a}, c, d) = \boldsymbol{\gamma}^*)$ then
    $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};$
    $v \leftarrow H(hk, a, \hat{a}, c);$
    if $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then
      return $c/(a^{z_1} \cdot \hat{a}^{z_2})$
    else return $\perp$
  else return $\perp$
```

$\models \mathsf{G}_1 \sim \mathsf{DDH}_0 : \mathsf{true} \Rightarrow =_{\{\mathsf{res}\}}$ $\qquad \Pr[\mathsf{G}_1 : b = b'] = \Pr[\mathsf{DDH}_0 : b = b']$

```
Game $\boxed{\mathsf{DDH}_0}$ $\boxed{\mathsf{DDH}_1}$ :
  $g \xleftarrow{\$} G \setminus \{1\}; x \xleftarrow{\$} \mathbb{Z}_q^*;\ y \xleftarrow{\$} \mathbb{Z}_q;$
  $\boxed{z \leftarrow xy}$ $\boxed{z \xleftarrow{\$} \mathbb{Z}_q}$;
  return $\mathcal{B}(g, g^x, g^y, g^z)$
Adversary $\mathcal{B}(g, \hat{g}, a, \hat{a})$ :
  $x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q;\ hk \xleftarrow{\$} K;$
  $e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^{z_1}\hat{g}^{z_2};$
  $(m_0, m_1) \leftarrow \mathcal{A}_1(hk, g, \hat{g}, e, f, h);\ b \xleftarrow{\$} \{0,1\};$
  $c \leftarrow a^{z_1} \cdot \hat{a}^{z_2} \cdot m_b;$
  $v \leftarrow H(hk, a, \hat{a}, c);\ d \leftarrow a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2};$
  $\boldsymbol{\gamma}^* \leftarrow (a, \hat{a}, c, d);\ \boldsymbol{\gamma}^*_{\mathsf{def}} \leftarrow \mathsf{true};\ b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$
```

```
Oracle $\mathcal{D}(a, \hat{a}, c, d)$ :
  if $|\boldsymbol{L}_{\mathcal{D}}| < q_{\mathcal{D}} \wedge \neg(\boldsymbol{\gamma}^*_{\mathsf{def}} \wedge (a, \hat{a}, c, d) = \boldsymbol{\gamma}^*)$ then
    $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};$
    $v \leftarrow H(hk, a, \hat{a}, c);$
    if $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then return $c/(a^{z_1} \cdot \hat{a}^{z_2})$
    else return $\perp$
  else return $\perp$
```

$\models \mathsf{DDH}_1 \sim \mathsf{G}_2 : \mathsf{true} \Rightarrow =_{\{\mathsf{res}\}}$ $\qquad \Pr[\mathsf{DDH}_1 : b = b'] = \Pr[\mathsf{G}_2 : b = b']$

```
Game $\mathsf{G}_2$ :
  $g \xleftarrow{\$} G \setminus \{1\};\ w \xleftarrow{\$} \mathbb{Z}_q^*;\ \hat{g} \leftarrow g^w;$
  $u, u' \xleftarrow{\$} \mathbb{Z}_q;\ a \leftarrow g^u;\ \hat{a} \leftarrow \hat{g}^{u'};$
  $x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q;\ hk \xleftarrow{\$} K;$
  $e \leftarrow g^{x_1}\hat{g}^{x_2};\ f \leftarrow g^{y_1}\hat{g}^{y_2};\ h \leftarrow g^{z_1}\hat{g}^{z_2};$
  $(m_0, m_1) \leftarrow \mathcal{A}_1(hk, h, \hat{g}, e, f, h);\ b \xleftarrow{\$} \{0,1\};$
  $c \leftarrow a^{z_1} \cdot \hat{a}^{z_2} \cdot m_b;$
  $v \leftarrow H(hk, a, \hat{a}, c);\ d \leftarrow a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2};$
  $\boldsymbol{\gamma}^* \leftarrow (a, \hat{a}, c, d);\ \boldsymbol{\gamma}^*_{\mathsf{def}} \leftarrow \mathsf{true};\ b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$
```

```
Oracle $\mathcal{D}(a, \hat{a}, c, d)$ :
  if $|\boldsymbol{L}_{\mathcal{D}}| < q_{\mathcal{D}} \wedge \neg(\boldsymbol{\gamma}^*_{\mathsf{def}} \wedge (a, \hat{a}, c, d) = \boldsymbol{\gamma}^*)$ then
    $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};\ v \leftarrow H(hk, a, \hat{a}, c);$
    if $\hat{a} = a^w$ then ;
      if $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then return $c/(a^{z_1} \cdot \hat{a}^{z_2})$
      else return $\perp$
    elsif $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then
      $\mathbf{bad} \leftarrow \mathsf{true};$ return $c/(a^{z_1} \cdot \hat{a}^{z_2})$
    else return $\perp$
  else return $\perp$
```

$\models \mathsf{G}_2 \sim \mathsf{G}_3 : \mathsf{true} \Rightarrow =_{\{\mathbf{bad}\}} \wedge \big(\neg\mathbf{bad}\langle 1\rangle \rightarrow =_{\{\mathsf{res}\}}\big)$ $\qquad |\Pr[\mathsf{G}_2 : b = b'] - \Pr[\mathsf{G}_3 : b = b']| \leq \Pr[\mathsf{G}_3 : \mathbf{bad}]$

```
Game $\mathsf{G}_3$ :
  $g \xleftarrow{\$} G \setminus \{1\};\ w \xleftarrow{\$} \mathbb{Z}_q^*;\ \hat{g} \leftarrow g^w;\ hk \xleftarrow{\$} K;$
  $x, x_2 \xleftarrow{\$} \mathbb{Z}_q;\ x_1 \leftarrow x - wx_2;\ e \leftarrow g^x;$
  $y, y_2 \xleftarrow{\$} \mathbb{Z}_q;\ y_1 \leftarrow y - wy_2;\ f \leftarrow g^y;$
  $z, z_2 \xleftarrow{\$} \mathbb{Z}_q;\ z_1 \leftarrow z - wz_2;\ h \leftarrow g^z;$
  $(m_0, m_1) \leftarrow \mathcal{A}_1(hk, h, \hat{g}, e, f, h);\ b \xleftarrow{\$} \{0,1\};$
  $u, u' \xleftarrow{\$} \mathbb{Z}_q;\ a \leftarrow g^u;\ \hat{a} \leftarrow \hat{g}^{u'};\ c \leftarrow a^{z_1} \cdot \hat{a}^{z_2} \cdot m_b;$
  $v \leftarrow H(hk, a, \hat{a}, c);\ d \leftarrow a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2};$
  $\boldsymbol{\gamma}^* \leftarrow (a, \hat{a}, c, d);\ \boldsymbol{\gamma}^*_{\mathsf{def}} \leftarrow \mathsf{true};\ b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$
```

```
Oracle $\mathcal{D}(a, \hat{a}, c, d)$ :
  if $|\boldsymbol{L}_{\mathcal{D}}| < q_{\mathcal{D}} \wedge \neg(\boldsymbol{\gamma}^*_{\mathsf{def}} \wedge (a, \hat{a}, c, d) = \boldsymbol{\gamma}^*)$ then
    $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};\ v \leftarrow H(hk, a, \hat{a}, c);$
    if $\hat{a} = a^w$ then
      if $d = a^{x+vy}$ then return $c/a^z$ else return $\perp$
    elsif $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then
      $\mathbf{bad} \leftarrow \mathsf{true};$ return $\perp$
    else return $\perp$
  else return $\perp$
```

$\models \mathsf{G}_3 \sim \mathsf{G}_4 : \mathsf{true} \Rightarrow (u = u')\langle 1\rangle \leftrightarrow (u = u')\langle 2\rangle \wedge \big((u \neq u')\langle 1\rangle \rightarrow =_{\{\mathsf{res},\mathbf{bad}\}}\big)$

$\Pr[\mathsf{G}_4 : b = b'] = 1/2$ $\qquad |\Pr[\mathsf{G}_3 : b = b'] - \Pr[\mathsf{G}_4 : b = b']| \leq \Pr[\mathsf{G}_3 : u = u'] = 1/q$

Fig. 2. Proof sketch of the IND-CCA security of the Cramer-Shoup cryptosystem

```
**Game** $\mathsf{G}_4$ :
  $g \stackrel{\$}{\leftarrow} G \setminus \{1\};\ w \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*;\ \hat{g} \leftarrow g^w;\ hk \stackrel{\$}{\leftarrow} K;$
  $x, x_2 \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ x_1 \leftarrow x - wx_2;\ e \leftarrow g^x;$
  $y, y_2 \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ y_1 \leftarrow y - wy_2;\ f \leftarrow g^y;$
  $z \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ h \leftarrow g^z;$
  $u, u' \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ a \leftarrow g^u;\ \hat{a} \leftarrow \hat{g}^{u'};\ r \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ c \leftarrow g^r;$
  $v \leftarrow H(hk, a, \hat{a}, c);\ d \leftarrow a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2};$
  $(m_0, m_1) \leftarrow \mathcal{A}_1(hk, h, \hat{g}, e, f, h);\ b \stackrel{\$}{\leftarrow} \{0,1\};$
  $\boldsymbol{\gamma}^* \leftarrow (a, \hat{a}, c, d);\ \boldsymbol{\gamma}^*_{\mathsf{def}} \leftarrow \mathsf{true};\ b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$
```

```
**Oracle** $\mathcal{D}(a, \hat{a}, c, d)$ :
  if $|\boldsymbol{L}_\mathcal{D}| < q_\mathcal{D} \wedge \neg(\boldsymbol{\gamma}^*_{\mathsf{def}} \wedge (a, \hat{a}, c, d) = \boldsymbol{\gamma}^*)$ then
    $\boldsymbol{L}_\mathcal{D} \leftarrow \gamma :: \boldsymbol{L}_\mathcal{D};\ v \leftarrow H(hk, a, \hat{a}, c);$
    if $\hat{a} = a^w$ then
      if $d = a^{x+vy}$ then return $c/a^z$ else return $\bot$
    elsif $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then
      $\mathbf{bad} \leftarrow \mathsf{true}$; return $\bot$
    else return $\bot$
  else return $\bot$
```

$\models \mathsf{G}_4 \sim \mathsf{G}_4' : \mathsf{true} \Rightarrow (u = u')\langle 1\rangle \leftrightarrow (u = u')\langle 2\rangle \wedge \big((u \neq u')\langle 1\rangle \rightarrow =_{\{\mathbf{bad}\}}\big)$

$\models \mathsf{G}_4' \sim \mathsf{G}_5 : \mathsf{true} \Rightarrow =_{\{\mathbf{bad}_1\}} \wedge \big(\neg\mathbf{bad}_1\langle 1\rangle \rightarrow =_{\{\mathbf{bad}, u, u'\}}\big)$

$\Pr[\mathsf{G}_4 : \mathbf{bad} \wedge u \neq u'] \leq \Pr[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u'] + \Pr[\mathsf{G}_5 : \mathbf{bad}_1] \leq \Pr[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u'] + (q_\mathcal{D}/q)^4$

```
**Game** $\boxed{\mathsf{G}_4'}$ $\mathsf{G}_5$ :
$g \stackrel{\$}{\leftarrow} G \setminus \{1\};\ w \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*;\ \hat{g} \leftarrow g^w;\ hk \stackrel{\$}{\leftarrow} K;$
$u, u' \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ a \leftarrow g^u;\ \hat{a} \leftarrow \hat{g}^{u'};$
$y, y_2 \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ y_1 \leftarrow y - wy_2;\ f \leftarrow g^y;$
$x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ e \leftarrow g^x;\ r' \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ d \leftarrow g^{r'};$
$x_2 \leftarrow (r' - u(x+vy))/(w(u'-u)) - vy_2;$
$x_1 \leftarrow x - wx_2;\ z \stackrel{\$}{\leftarrow} \mathbb{Z}_q; h \leftarrow g^z;\ r \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ c \leftarrow g^r;$
$v \leftarrow H(hk, a, \hat{a}, c);\ \boldsymbol{\gamma}^* \leftarrow (a, \hat{a}, c, d);$
$(m_0, m_1) \leftarrow \mathcal{A}_1(hk, h, \hat{g}, e, f, h);$
$\boldsymbol{\gamma}^*_{\mathsf{def}} \leftarrow \mathsf{true};\ b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$
```

```
**Oracle** $\mathcal{D}(a, \hat{a}, c, d)$ :
  if $|\boldsymbol{L}_\mathcal{D}| < q_\mathcal{D} \wedge \neg\boldsymbol{\gamma}^*_{\mathsf{def}} \wedge (a, \hat{a}, c, d) = \boldsymbol{\gamma}^*$ then
    $\mathbf{bad}_1 \leftarrow \mathsf{true}$
  if $|\boldsymbol{L}_\mathcal{D}| < q_\mathcal{D} \wedge (\boxed{\neg\boldsymbol{\gamma}^*_{\mathsf{def}} \vee}(a, \hat{a}, c, d) \neq \boldsymbol{\gamma}^*)$ then
    $\boldsymbol{L}_\mathcal{D} \leftarrow \gamma :: \boldsymbol{L}_\mathcal{D};\ v \leftarrow H(hk, a, \hat{a}, c);$
    if $\hat{a} = a^w$ then
      if $d = a^{x+vy}$ then return $c/a^z$ else return $\bot$
    elsif $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2}$ then $\mathbf{bad} \leftarrow \mathsf{true}$;
      if $v = H(hk, g^u, \hat{g}^{u'}, g^r)$ then $\mathbf{bad}_2 \leftarrow \mathsf{true}$
    else return $\bot$
  else return $\bot$
```

$\models \mathsf{G}_5 \sim \mathsf{TCR} : \mathsf{true} \Rightarrow \mathbf{bad}_2\langle 1\rangle \rightarrow \mathsf{res}\langle 2\rangle$

$\Pr[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u'] \leq \Pr[\mathsf{TCR} : H(hk, m_0) = H(hk, m_1) \wedge m_0 \neq m_1] + \Pr[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u' \wedge \neg\mathbf{bad}_2]$

```
**Game** $\mathsf{TCR}$ :
  $m_0 \leftarrow \mathcal{C}_1(\,);\ hk \stackrel{\$}{\leftarrow} K;\ m_1 \leftarrow \mathcal{C}_2(hk);$
  return $(H(hk, m_0) = H(hk, m_1) \wedge m_0 \neq m_1)$
**Adversary** $\mathcal{C}_1()$ :
  $g \stackrel{\$}{\leftarrow} G \setminus \{1\};\ w \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*;\ \hat{g} \leftarrow g^w;$
  $u, u' \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ a \leftarrow g^u;\ \hat{a} \leftarrow \hat{g}^{u'};\ r \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ c \leftarrow g^r;$
  return $(a, \hat{a}, c)$
**Adversary** $\mathcal{C}_2(hk)$ :
  $r', x, y, z \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ d \leftarrow g^{r'}; e \leftarrow g^x; f \leftarrow g^y;\ h \leftarrow g^z;$
  $y_2 \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ y_1 \leftarrow y - wy_2;\ \hat{hk} \leftarrow hk;$
  $v \leftarrow H(hk, a, \hat{a}, c);$
  $x_2 \leftarrow (r' - u(x+vy))/(w(u'-u)) - vy_2;$
  $x_1 \leftarrow x - wx_2;\ \ (m_0, m_1) \leftarrow \mathcal{A}_1(hk, h, \hat{g}, e, f, h);$
  $\boldsymbol{\gamma}^* \leftarrow (a, \hat{a}, c, d);\ b' \leftarrow \mathcal{A}_2(\boldsymbol{\gamma}^*)$; return $\hat{m}$
```

```
**Oracle** $\mathcal{D}(a, \hat{a}, c, d)$ :
  if $|\boldsymbol{L}_\mathcal{D}| < q_\mathcal{D} \wedge (a, \hat{a}, c, d) \neq \boldsymbol{\gamma}^*)$ then
    $\boldsymbol{L}_\mathcal{D} \leftarrow \gamma :: \boldsymbol{L}_\mathcal{D};\ v \leftarrow H(\hat{hk}, a, \hat{a}, c);$
    if $\hat{a} = a^w$ then
      if $d = a^{x+vy}$ then return $c/a^z$ else return $\bot$
    elsif $d = a^{x_1+vy_1} \cdot \hat{a}^{x_2+vy_2} \wedge v = H(\hat{hk}, g^u, \hat{g}^{u'}, g^r)$
    then $\hat{m} \leftarrow (a, \hat{a}, c)$; return $\bot$
    else return $\bot$
  else return $\bot$
```

Fig. 2. Proof sketch of the $\mathsf{IND\text{-}CCA}$ security of the Cramer-Shoup cryptosystem

The decryption oracle in game G_5 also raises a flag $\mathbf{bad}_2$ when a valid ciphertext with $H(hk, a, \hat{a}, c) = H(hk, g^u, \hat{g}^{u'}, g^r)$ is queried. Since this leads to a collision, we can build an adversary $\mathcal{C}$ against the TCR of H such that its success probability is lower bounded by the probability of $\mathbf{bad}_2$ being raised in G_5. Thus,

$$\Pr\left[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u'\right] \leq \mathbf{Succ}^{\mathcal{C}}_{\mathsf{TCR}} + \Pr\left[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u' \wedge \neg\mathbf{bad}_2\right]$$

The proof concludes by showing that the probability of **bad** being set but not $\mathbf{bad}_2$ in G_5 is bounded by $q_{\mathcal{D}}/q$. This is done by reformulating the test under which $\mathbf{bad}_2$ is set so that it does not depend on x_1, x_2, y_1, y_2, therefore, the probability of this test succeeding in any decryption query (under the condition that $u \neq u'$) is the probability of the adversary guessing a random value in the group, at most $q_{\mathcal{D}}/q$ summing over all queries. The bound in the statement follows.

5 Conclusion

Computer-aided verification of cryptographic protocols in the symbolic model is an established field of research: robust tools are available and have been used successfully to analyze realistic protocols [2, 9, 18]. In contrast, there is little prior work on computer-aided cryptographic proofs in the computational model. The importance of such proofs was suggested independently by Bellare and Rogaway [8] and, more explicitly, by Halevi [17], who convincingly argues that they can be viewed as the "natural next step along the way of viewing cryptographic proofs as a sequence of probabilistic games". To date, there are two main tools for computer-aided cryptographic proofs: CertiCrypt, which favors generality and verifiable proofs, and CryptoVerif, which favors automation. Other frameworks are confined to specific properties [14], or dispense from formalizing the computational model by advocating soundness results [21, 22], or have not yet been applied to significant examples [1], or are not implemented [7, 19].

EasyCrypt provides, the first flexible and automated framework for building machine-checkable cryptographic proofs. Moreover, proofs in EasyCrypt are significantly easier and faster to build than in any of its predecessors, while providing guarantees similar to CertiCrypt. We have substantiated our claim by presenting computer-aided securtiy proofs of Hashed ElGamal in the random oracle model and of the Cramer-Shoup cryptosystem in the standard model. Overall, EasyCrypt makes an important step towards the adoption of computer-aided proofs by working cryptographers. A public release of EasyCrypt is planned for Summer 2011. We intend to incorporate some missing features, for instance automated complexity analyses, and proof-producing computations of probabilities. Moreover, we plan to incorporate a few useful functionalities, such as generating LaTeX code and figures from sequences of games. A longer-term goal will be to build a structured proof editor for writing games, and a user interface that supports rich features like proof-by-pointing or drag-and-drop of pieces of code.

There remain ample opportunities to apply methods from programming languages and formal verification to computer-aided cryptographic proofs. We mention two exciting avenues for improving automation in EasyCrypt. The first avenue is to improve our mechanism for inferring relational specifications of adversaries: there is a large body of knowledge on inferring invariants, and it would be beneficial to transpose them to our setting. More speculatively, program synthesis could be used to discover the sequence of games that must be used to carry a game-based proof. Both specification inference and program synthesis rely on verification condition generation and SMT solving, hence the basic blocks for such an investigation are in place.

References

1. M. Backes, M. Berg, and D. Unruh. A formal language for cryptographic pseudocode. In *15th International conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2008*, volume 5330 of *Lecture Notes in Computer Science*, pages 353–376. Springer, 2008.
2. Michael Backes, Matteo Maffei, and Dominique Unruh. Computationally sound verification of source code. In *17th ACM conference on Computer and Communications Security, CCS 2010*, pages 387–398, New York, 2010. ACM.
3. G. Barthe, P. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *17th IEEE workshop on Computer Security Foundations, CSFW 2004*, pages 100–114, Washington, 2004. IEEE Computer Society.
4. G. Barthe, B. Grégoire, and S. 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.
5. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago 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*, volume 5491 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, 2009. Springer.
6. Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago Zanella Béguelin. Beyond provable security. Verifiable IND-CCA security of OAEP. In *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196, Berlin, 2011. Springer.
7. Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. A machine-checked formalization of Sigma-protocols. In *23rd IEEE Computer Security Foundations symposium, CSF 2010*, pages 246–260, Los Alamitos, Calif., 2010. IEEE Computer Society.
8. 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, Berlin, 2006. Springer.
9. Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Modular verification of security protocol code by typing. In *37th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL 2010*, pages 445–456. ACM, 2010.
10. B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In *15th ACM conference on Computer and Communications Security, CCS 2008*, pages 87–99, New York, 2008. ACM.
11. Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *27th IEEE symposium on Security and Privacy, S&P 2006*, pages 140–154. IEEE Computer Society, 2006.
12. Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554, Berlin, 2006. Springer.
13. Sylvain Conchon, Evelyne Contejean, Johannes Kanig, and Stéphane Lescuyer. CC(X): Semantic combination of congruence closure with solvable theories. *Electronic Notes in Theoretical Computer Science*, 198(2):51–69, 2008.
14. Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *15th ACM conference on Computer and Communications Security, CCS 2008*, pages 371–380, New York, 2008. ACM.
15. David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Laboratories Palo Alto, 2003.
16. Jean-Christophe Filliâtre. The WHY verification tool: Tutorial and Reference Manual Version 2.28. Online – `http://why.lri.fr`, 2010.
17. S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005.
18. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *J. of Comput. Secur.*, 6(1-2):85–128, 1998.
19. Arnab Roy, Anupam Datta, Ante Derek, and John Mitchell. Inductive proofs of computational secrecy. In *Computer Security – ESORICS 2007, 12th European symposium on Research In Computer Security*, volume 4734 of *Lecture Notes in Computer Science*, pages 219–234, Berlin, 2008. Springer.

20. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.
21. Christoph Sprenger, Michael Backes, David Basin, Birgit Pfitzmann, and Michael Waidner. Cryptographically sound theorem proving. In *19th IEEE workshop on Computer Security Foundations, CSFW 2006*, pages 153–166. IEEE Computer Society, 2006.
22. Christoph Sprenger and David Basin. Cryptographically-sound protocol-model abstractions. In *21st IEEE Computer Security Foundations symposium, CSF 2008*, pages 115–129, Los Alamitos, Calif., 2008. IEEE Computer Society.
23. Aaron Stump. Proof checking technology for satisfiability modulo theories. *Electr. Notes Theor. Comput. Sci.*, 228:121–133, 2009.
24. The Coq development team. The Coq Proof Assistant Reference Manual Version 8.3. Online – `http://coq.inria.fr`, 2010.
25. 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*, pages 237–250, Los Alamitos, Calif., 2009. IEEE Computer Society.

A Input File for the Proof of Security of Hashed ElGamal

The following is an extract of the EasyCrypt input file giving a proof sketch for the IND-CPA security of Hashed ElGamal described in Section 2:

```
type group

cnst k : int

type secret_key = int
type public_key = group
type keys       = secret_key * public_key
type message    = bitstring{k}
type cipher     = group * bitstring{k}

cnst q    : int
cnst qH   : int
cnst g    : group
cnst zero : bitstring{k}

op (*) : group, group -> group                       = group_mult
op (^) : group, int -> group                         = group_pow
op (+) : bitstring{k}, bitstring{k} -> bitstring{k} = xor_bs

axiom group_pow_mult : forall (x:int, y:int). { (g^x)^y == g^(x*y) }

...

adversary A1(pk:public_key)           : message * message { group -> message }
adversary A2(pk:public_key, c:cipher) : bool              { group -> message }

game INDCPA = {
  var L : (group, bitstring{k}) map

  fun H(lam:group) : message = {
    var h : message = {0,1}^k;
    if (! in_dom(lam, L) ) { L[lam] = h; };
    return L[lam];
  }

  fun H_A (lam:group) : message = {
    var m : message;
    LA = lam :: LA;
    m = H(lam);
    return m;
  }

  ...

  abs A1 = A1 {H_A}
  abs A2 = A2 {H_A}

  fun Main () : bool = {
    var sk : secret_key;
    var pk : public_key;
    var m0, m1 : message;
    var c : cipher;
    var b, b' : bool;

    L = empty_map(); LA = [];
    (sk,pk) = KG();
    (m0,m1) = A1(pk);
    b = {0,1};
    c = Enc(pk, b ? m0 : m1);
    b' = A2(pk, c);
    return (b == b');
  }
}

game G1 = INDCPA
  var y' : group
  where Main = {
    var sk : secret_key;
    var pk : public_key;
    var m0, m1 : message;
    var c: cipher;
    var b, b' : bool;
    var y:int;
    var hy : message;

    L = empty_map (); LA = [];
    (sk,pk) = KG();
    y = [0..q-1];
    y' = pk^y;
    (m0,m1) = A1(pk);
    b = {0,1};
    hy = H(y');
    b' = A2(pk, (g ^ y, hy ^^ (b ? m0 : m1)));
    return (b == b');
  }

equiv auto Fact1 : INDCPA.Main \tilde G1.Main : {true} ==> ={res,LA} inv ={L,LA}

claim Pr1 : INDCPA.Main[res && length(LA) <= qH] == G1.Main[res && length(LA) <= qH]
 using Fact1

...
```

B Justifying Verification Conditions

The purpose of this appendix is to justify the last step in the generation of verification conditions. For simplicity, we only deal with the case where Φ is a partial equivalence relation. In this case, a judgment $\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$ is valid iff for every m', m_1, m_2 s.t. $m_1 \; \Psi \; m_2$,

$$\Pr\left[\mathsf{G}_1, m_1 : \lambda m.\ m \; \Phi \; m'\right] = \Pr\left[\mathsf{G}_2, m_2 : \lambda m.\ m \; \Phi \; m'\right]$$

Now let $S_1 \overset{\text{def}}{=} x_1 \overset{\$}{\leftarrow} T_1; \dots x_l \overset{\$}{\leftarrow} T_l$ and $S_2 \overset{\text{def}}{=} y_1 \overset{\$}{\leftarrow} U_1; \dots y_n \overset{\$}{\leftarrow} U_n$. Assume there exists a 1-1 mapping $f : T_1 \times \cdots \times T_l \to U_1 \times \cdots \times U_n$ such that the verification condition $\Phi \Rightarrow_f \Psi$, defined as

$$\forall m_1\ m_2\ \vec{t}\ .\ m_1\ \Psi\ m_2 \implies m_1 \left\{\vec{t}/\vec{x}\right\}\ \Phi\ m_2 \left\{f(\vec{t})/\vec{y}\right\}$$

is valid. We prove that $\models S_1 \sim S_2 : \Psi \Rightarrow \Phi$. First, observe that

$$\Pr\left[S_1, m_1 : \lambda m.\ m\ \Phi\ m'\right] = \frac{\#\{\vec{t} \mid m_1 \left\{\vec{t}/\vec{x}\right\}\ \Phi\ m'\}}{\#(T_1 \times \cdots \times T_l)}$$

This equality follows from the definition of the semantics of S_1 since the set $T_1 \times \cdots \times T_l$ is finite. Likewise,

$$\Pr\left[S_2, m_2 : \lambda m.\ m\ \Phi\ m'\right] = \frac{\#\{\vec{u} \mid m_2 \left\{\vec{u}/\vec{y}\right\}\ \Phi\ m'\}}{\#(U_1 \times \cdots \times U_n)}$$

Now let m_1 and m_2 such that $m_1\ \Psi\ m_2$. We claim that

$$\Pr\left[S_1, m_1 : \lambda m.\ m\ \Phi\ m'\right] = \Pr\left[S_2, m_2 : \lambda m.\ m\ \Phi\ m'\right]$$

from which the result follows.

Since f is a 1-1 mapping, we have $\#(T_1 \times \cdots \times T_l) = \#(U_1 \times \cdots \times U_n)$, so by the above it is sufficient to show that for every memory m', $\#\{\vec{t} \mid m_1 \left\{\vec{t}/\vec{x}\right\}\ \Phi\ m'\} = \#\{\vec{u} \mid m_2 \left\{\vec{u}/\vec{y}\right\}\ \Phi\ m'\}$. Since Φ is a partial equivalence relation and $\Phi \Rightarrow_f \Psi$ is valid, we can easily prove that for every $\vec{t} \in T_1 \times \cdots \times T_l$, we have $m_1 \left\{\vec{t}/\vec{x}\right\}\ \Phi\ m'$ iff $m_2 \left\{\vec{u}/\vec{y}\right\}\ \Phi\ m'$, where $f(\vec{t}) = (\vec{u})$. The claim follows.