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, Madrid, Spain
[2] INRIA Sophia Antipolis-Méditerranée, France

Abstract. We present EasyCrypt, 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 common reasoning patterns and is significantly easier to use than its predecessors. We argue that EasyCrypt is a plausible candidate for adoption by working cryptographers and illustrate its application to security proofs of the Cramer-Shoup and Hashed ElGamal cryptosystems.

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

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, referred to as games, of the interaction between adversaries and oracle systems. Proofs are organized as sequences of games, starting from a game that represents a security goal (e.g. indistinguishability against chosen-ciphertext attacks), and proceeding to games that represent security assumptions (e.g. Decision Diffie-Hellman) by successive transformations that can be shown to preserve, or alter only slightly the overall security. In a typical step in a game-based proof the goal is to relate the probability of an event A in a game G to the probability of a possibly different event A' in a game G'. For example, the goal may be to establish an inequality of the form $\Pr[G : A] \leq \Pr[G' : A'] + \varDelta$, where $\varDelta$ is an arithmetic expression that depends on the number of oracle queries made by an adversary. The prevailing practice for proving the validity of such proof steps is to use standard mathematical tools, which interleave reasoning about the semantics of games with information-theoretic or arithmetical arguments.

[⋆] Partially funded by European Project FP7-256980 NESSoS, French project ANR SESUR-012 SCALP, Spanish project TIN2009-14599 DESAFIOS 10, and Madrid Regional project S2009TIC-1465 PROMETIDOS.

In the code-based approach to the game-playing technique [8, 17] games are cast as probabilistic algorithms. The adoption of programming idioms allows to give 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 tool for conducting security proofs in a game-based setting in which games are modeled as processes and transitions are justified by means of process-algebraic concepts such as bisimulations. One strength of CryptoVerif, apart from being the first tool to have supported game-based proofs, is that it applies both to protocols and primitives; it has been successfully applied to verify Kerberos [10] and the Full-Domain Hash (FDH) signature scheme [12]. CertiCrypt [6] is another framework that allows for the interactive construction of game-based proofs in the Coq proof assistant [22]. 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 [5], FDH [24], 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 audience among cryptographers. In [5], we contrast the high guarantees given by CertiCrypt 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. 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 encryption 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. Transitions between games 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 claims about the probability of events 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 for its validity, known as verification conditions. 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 about probability. 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 a limited form of specification inference for procedures provides substantial leverage towards making verifiable security practical and makes EasyCrypt a plausible candidate for adoption by working cryptographers.

2 Introductory Example: Hashed ElGamal Encryption

This section illustrates the application of EasyCrypt to a proof of IND-CPA security of Hashed ElGamal encryption 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 [4] 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. Instead, plaintexts are bitstrings of a certain length k and group elements are mapped into bitstrings using a hash function $H : \mathcal{G} \to \{0,1\}^k$. Let $\mathcal{G}$ be a multiplicative cyclic group of order q with generator g. Formally, the scheme is defined by the following triple of algorithms:

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

The security of Hashed ElGamal can be reduced to the Computational Diffie-Hellman (CDH) assumption on the underlying group family. This is the assumption that it is hard to compute g^{xy} given g^x and g^y where x and y are uniformly random elements in $\mathbb{Z}_q$. 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 shows the sequence of games used to justify the security reduction. This is an essential part of the proof sketch that is input to EasyCrypt, and which is composed of five ingredients:[3]

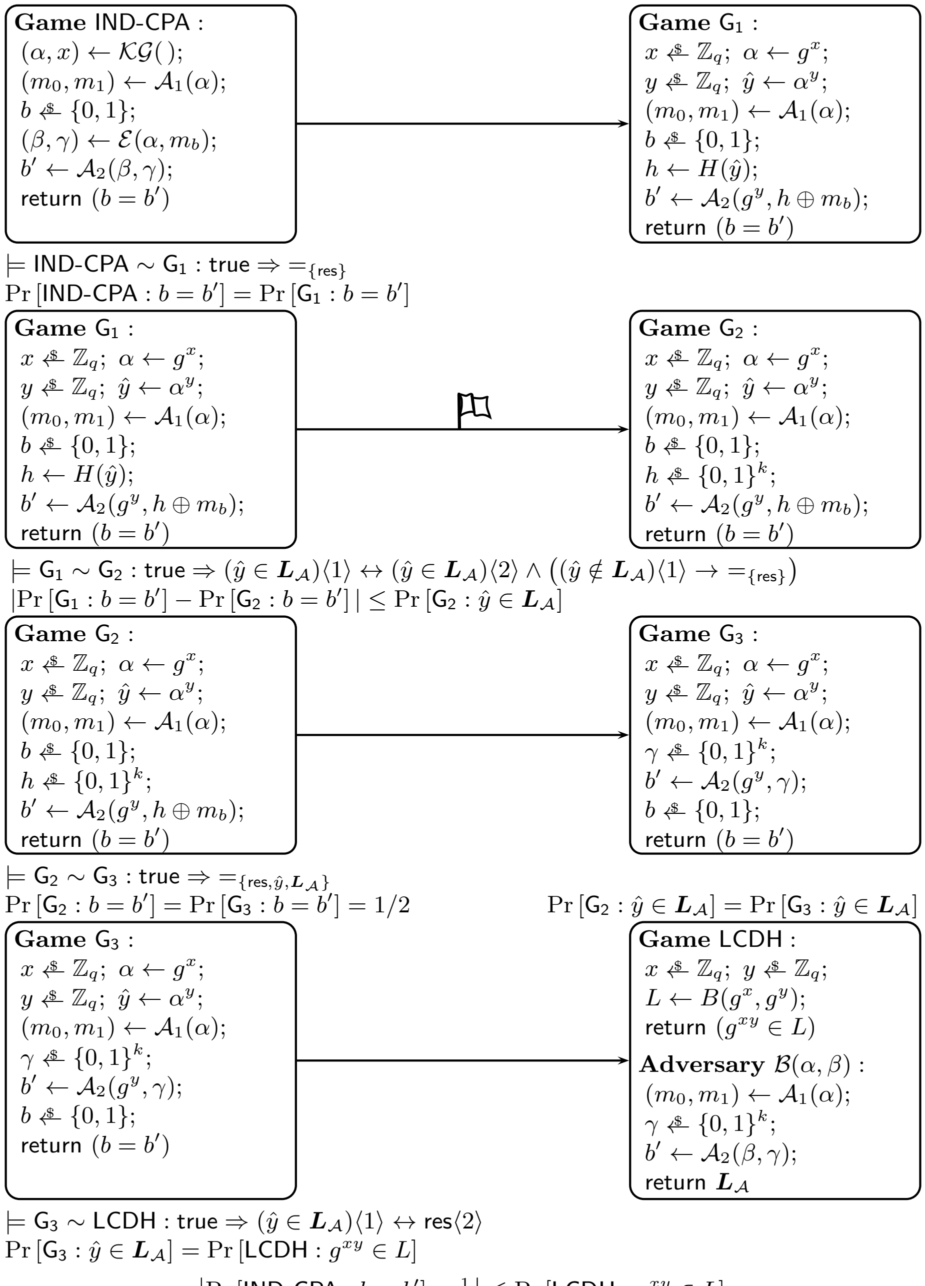
1. Type, constant and operator declarations, which introduce the objects manipulated by the scheme. In this case, they include a type for elements of the cyclic group $\mathcal{G}$, constants representing the length of messages k, the order of the group q and a generator g, and operators denoting the 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. We use axioms to state properties of the group law and exponentiation, and the exclusive or operator;
3. Game definitions, where adversaries are specified as abstract procedures with access to oracles. In all games in the figure the hash function H is modeled as a random oracle and the adversary is represented as two procedures $\mathcal{A}_1$ and $\mathcal{A}_2$ that share state. The procedures representing the adversary are given access to a wrapper $H_{\mathcal{A}}$ for the hash oracle that just stores queries in a list $\boldsymbol{L}_{\mathcal{A}}$ before forwarding them to H:

$$\begin{array}{lcl} H(x) & \overset{\text{def}}{=} & \mathsf{if}\ x \notin \mathsf{dom}(\boldsymbol{L})\ \mathsf{then}\ h \overset{\$}{\leftarrow} \{0,1\}^k; \boldsymbol{L}[x] \leftarrow h\ \mathsf{end\ if}; \mathsf{return}\ \boldsymbol{L}[x] \\ H_{\mathcal{A}}(x) & \overset{\text{def}}{=} & \boldsymbol{L}_{\mathcal{A}} \leftarrow x :: \boldsymbol{L}_{\mathcal{A}};\ m \leftarrow H(x);\ \mathsf{return}\ m \end{array}$$

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 program memories (memories map program variables to values). 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. We often consider equivalence of memories on a set of variables X; we use $=_X$ as a shorthand for the formula $\forall x \in X.\ x\langle 1 \rangle = x\langle 2 \rangle$;
5. Claims about probability, 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 in Figure 1. The first and last games encode the IND-CPA and LCDH experiments, respectively. We obtain G_1 by inlining the key generation and encryption procedures in the initial game and rearranging instructions so that random choices are made upfront. We prove that games IND-CPA and G_1 yield identical distributions on the result of the game (denoted by the keyword res). We deduce from this that the probability of the event $b = b'$ is the same in both games.

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



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

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

$\models \mathsf{IND\text{-}CPA} \sim \mathsf{G}_1 : \mathsf{true} \Rightarrow =_{\{\mathsf{res}\}}$
$\Pr[\mathsf{IND\text{-}CPA} : b = b'] = \Pr[\mathsf{G}_1 : b = b']$

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

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

$\models \mathsf{G}_1 \sim \mathsf{G}_2 : \mathsf{true} \Rightarrow (\hat{y} \in \boldsymbol{L}_{\mathcal{A}})\langle 1 \rangle \leftrightarrow (\hat{y} \in \boldsymbol{L}_{\mathcal{A}})\langle 2 \rangle \wedge \big((\hat{y} \notin \boldsymbol{L}_{\mathcal{A}})\langle 1 \rangle \rightarrow =_{\{\mathsf{res}\}}\big)$
$\big|\Pr[\mathsf{G}_1 : b = b'] - \Pr[\mathsf{G}_2 : b = b']\,\big| \leq \Pr[\mathsf{G}_2 : \hat{y} \in \boldsymbol{L}_{\mathcal{A}}]$

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

```
Game $\mathsf{G}_3$ :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $\gamma \xleftarrow{\$} \{0,1\}^k$;
  $b' \leftarrow \mathcal{A}_2(g^y, \gamma)$;
  $b \xleftarrow{\$} \{0,1\}$;
  return $(b = b')$
```

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

```
Game $\mathsf{G}_3$ :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x$;
  $y \xleftarrow{\$} \mathbb{Z}_q;\ \hat{y} \leftarrow \alpha^y$;
  $(m_0, m_1) \leftarrow \mathcal{A}_1(\alpha)$;
  $\gamma \xleftarrow{\$} \{0,1\}^k$;
  $b' \leftarrow \mathcal{A}_2(g^y, \gamma)$;
  $b \xleftarrow{\$} \{0,1\}$;
  return $(b = b')$
```

```
Game LCDH :
  $x \xleftarrow{\$} \mathbb{Z}_q;\ y \xleftarrow{\$} \mathbb{Z}_q$;
  $L \leftarrow B(g^x, g^y)$;
  return $(g^{xy} \in L)$

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 $\boldsymbol{L}_{\mathcal{A}}$
```

$\models \mathsf{G}_3 \sim \mathsf{LCDH} : \mathsf{true} \Rightarrow (\hat{y} \in \boldsymbol{L}_{\mathcal{A}})\langle 1 \rangle \leftrightarrow \mathsf{res}\langle 2 \rangle$
$\Pr[\mathsf{G}_3 : \hat{y} \in \boldsymbol{L}_{\mathcal{A}}] = \Pr[\mathsf{LCDH} : g^{xy} \in L]$

$$\big|\Pr[\mathsf{IND\text{-}CPA} : b = b'] - \tfrac{1}{2}\big| \leq \Pr[\mathsf{LCDH} : g^{xy} \in L]$$

Fig. 1. Proof sketch of Hashed ElGamal security

In game G_2 we substitute the value $H(\hat{y})$ used to compute the challenge ciphertext by a uniformly chosen value. This only makes a difference if $\mathcal{A}_1$ queries $\hat{y}$ to H, and this happens with the same probability in either game. Thus, the difference in the probability of any event in these games is bounded by the probability of $\hat{y} \in \boldsymbol{L}_{\mathcal{A}}$ in G_2. This can be seen as a semantic variant of the Fundamental Lemma of Game-Playing; the logic allows to dispense with the code instrumentation needed to apply the syntactic counterpart of the lemma.

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 $h \oplus m_b$, we sample γ and define $h = \gamma \oplus m_b$; we then remove the definition of h as dead code. This transformation is proven admissible within the logic and removes the dependency of the adversary's output from the challenge bit 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 games LCDH 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 5 times shorter than the proof in CertiCrypt reported in [4]—and arguably much simpler and close to a pen-and-paper proof.

3 An Overview of EasyCrypt

Programming Language Games are modeled as programs in a typed, probabilistic, procedural, imperative language. Types include Booleans, integers, bitstrings, pairs, lists, maps, and user-defined types. Expressions are built from variables and operators in the usual way; for instance, Boolean-valued operators include the usual connectives, equality, list membership, arithmetic comparisons. The commands of the language are defined by the following grammar:

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

where $\mathcal{V}$ is a set of variables, $\mathcal{P}$ is a set of procedures, and $\mathcal{DE}$ is a set of distribution expressions. For the purpose of this article, distribution expressions are restricted to uniform distributions over specific domains, for instance integers in $\mathbb{Z}_q$ or (non-neutral) elements of some group $\mathcal{G}$. Adversaries are modeled as abstract procedures with an interface that specifies the oracles they may query.

Games can be given a semantics as memory distribution transformers, in the style of [6]. Formally, memories are well-typed mappings from variables to values, and the semantics of a game G is a function, 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 (a Boolean expression), 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.

Relational Judgments Pre- and post-conditions in pRHL judgments are first-order formulae built from relational expressions. Relational expressions are arbitrary Boolean expressions over logical variables and program variables tagged with $\langle 1\rangle, \langle 2\rangle$; the only restriction is that logical variables may only appear quantified. By abuse of notation, we write $e\langle i\rangle$ for the expression e in which all variables have been tagged with $\langle i\rangle$. Let b stand for an arbitrary Boolean expression over tagged and logical variables, then logical formulae are defined by the following grammar:

$$\Psi, \Phi ::= b \mid \neg\Phi \mid \Psi \wedge \Phi \mid \Psi \vee \Phi \mid \Psi \rightarrow \Phi \mid \Psi \leftrightarrow \Phi \mid (\Phi) \mid \forall x.\ \Phi \mid \exists x.\ \Phi$$

A logical formula is interpreted as a relation on program memories. For example, the formula $x\langle 1\rangle + y\langle 2\rangle \leq z\langle 1\rangle$ is interpreted as the relation

$$R = \{(m_1, m_2) \mid m_1(x) + m_2(y) \leq m_1(z)\}$$

A pRHL judgment $\models \mathsf{G}_1 \sim \mathsf{G}_2 : \Psi \Rightarrow \Phi$ is valid iff for any pair of initial memories m_1, m_2 satisfying the pre-condition Ψ, the distributions $[\![\mathsf{G}_1]\!]\ m_1$ and $[\![\mathsf{G}_2]\!]\ m_2$ satisfy the lifting of post-condition Φ, $([\![\mathsf{G}_1]\!]\ m_1)\, \mathcal{L}(\Phi)\, ([\![\mathsf{G}_2]\!]\ m_2)$. The lifting of a relation to a distribution is defined as a max-cut min-flow problem, in the style of [18]. Formally, let μ_1 be a probability distribution on a set A and μ_2 a probability distribution on a set B. We define the lifting $\mu_1\, \mathcal{L}(R)\, \mu_2$ of a relation $R \subseteq A \times B$ to μ_1 and μ_2 as follows:[4]

$$\exists \mu : \mathcal{D}(A \times B).\ \pi_1(\mu) = \mu_1 \wedge \pi_2(\mu) = \mu_2 \wedge \forall (a, b) : A \times B.\ \mu(a, b) > 0 \implies a\ R\ b$$

where the projections $\pi_1(\mu)$ and $\pi_2(\mu)$ of μ are defined as

$$\pi_1(\mu)(a) \overset{\text{def}}{=} \sum_{b \in B} \mu(a, b) \qquad \pi_2(\mu)(b) \overset{\text{def}}{=} \sum_{a \in A} \mu(a, b)$$

Claims about probability can be derived from valid relational judgments by means of the following rules:

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

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

Automated Proofs of Relational Judgments Most practical verification tools adopt a similar methodology: a weakest precondition (wp) calculus is used to compute from a program and its specification a set of sufficient conditions, known as verification conditions, and these conditions are discharged by automated

[4] For the clarity of presentation, we assume that A and B are discrete and cast our definitions using the usual representation of distributions. However, the tool builds on a monadic representation of distributions, as in [6].

tools. Extending the methodology to the logic pRHL is a significant challenge, for two reasons: first, generating verification conditions for a relational program logic is an open topic of research, and second, there is no prior application of the methodology to procedural nor probabilistic programs.

There are at least two natural strategies for defining a wp calculus in a relational setting. The calculus can either operate on both games in lockstep, or else it can operate on each game separately, in the style of self-composition [2]. Both strategies are incomplete: the lockstep wp calculus fails on programs that are not structurally equivalent, whereas self-composition fails to handle random assignments and adversary calls. In order to circumvent these limitations, EasyCrypt implements an alternative approach that mixes both strategies:

1. Calls to non-adversary procedures are eliminated from the games by successive inlining their definitions. In the absence of recursion, the transformation terminates successfully and only adversary calls remain;
2. Random assignments are moved upfront. The resulting code consists of a sequence of random assignments followed by deterministic code, possibly with adversary calls;
3. A relational weakest precondition calculus is applied to the deterministic fragment of the game, using relational specifications to deal with adversary calls. Each adversary specification induces a proof obligation, expressed as a pRHL judgment, on the oracles in its interface. Self-composition is applied to verify the code of oracles with respect to these pRHL judgments. This results in a judgment of the form

$$\models x_1 \overset{\$}{\leftarrow} T_1; \ldots x_l \overset{\$}{\leftarrow} T_l \sim y_1 \overset{\$}{\leftarrow} U_1; \ldots y_n \overset{\$}{\leftarrow} U_n : \Psi \Rightarrow \Phi$$

4. A mapping $f : T_1 \times \cdots \times T_l \to U_1 \times \cdots \times U_n$ is selected, and used to generate the verification condition $\Phi \Rightarrow_f \Psi$, defined as[5]

$$\forall m_1\ m_2\ t_1 \ldots t_l\ .\ m_1\ \Psi\ m_2 \implies m_1 \left\{\vec{t}/\vec{x}\right\}\ \Phi\ m_2 \left\{f(t_1, \ldots, t_l)/\vec{y}\right\}$$

   Under specific conditions on f, see [23], the validity of $\Phi \Rightarrow_f \Psi$ entails the validity of the corresponding pRHL judgment. In practice, it is generally sufficient to require that f is a 1-1 mapping, and taking f as the identity function works most of the time. However, in some cases other mappings must be used. For example, to prove the equivalence between games G_2 and G_3 in the proof of Hashed ElGamal described in the previous section, it is necessary to prove a judgment like the following:

$$\models h \overset{\$}{\leftarrow} \{0,1\}^k;\ \gamma \leftarrow h \oplus m_b \sim \gamma \overset{\$}{\leftarrow} \{0,1\}^k;\ h \leftarrow \gamma \oplus m_b : =_{\{m_b\}} \Rightarrow =_{\{h,\gamma\}}$$

   The wp will stop after computing the weakest precondition for the deterministic fragment of the two programs, yielding

$$\models h \overset{\$}{\leftarrow} \{0,1\}^k \sim \gamma \overset{\$}{\leftarrow} \{0,1\}^k : =_{\{m_b\}} \Rightarrow (h\langle 1\rangle = \gamma\langle 2\rangle \oplus m_b\langle 2\rangle)$$

[5] The memory $m_1 \left\{\vec{t}/\vec{x}\right\}$ maps x_i to t_i for $i = 1 \ldots l$ and y to $m_1(z)$ for $z \notin \{x_1 \ldots x_l\}$. Likewise, $m_2 \left\{f(t_1, \ldots, t_l)/\vec{y}\right\}$ is the memory that maps y_i to $\pi_i(f(t_1, \ldots, t_l))$ for $i = 1 \ldots n$ and z to $m_2(z)$ for $z \notin \{y_1 \ldots y_n\}$.

This equivalence is proved in EasyCrypt by providing the bijective function $f(x) = x \oplus m_b$ as a witness. The fact that f is bijective is established automatically since f is idempotent. In the general case this is proved by providing also the inverse mapping.

5. Since $\Phi \Rightarrow_f \Psi$ is a first-order formula, its 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] and the alt-ergo SMT solver [13] to discharge the conditions (although many others provers are supported, including interactive theorem provers such as Coq).

Verification condition generation is incomplete (in the logical sense), and would fail on pRHL judgments where games perform calls to adversaries 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.

A Mechanized Probabilistic Relational Hoare Logic EasyCrypt implements a simple tactic language to prove the validity of judgments using rules of the logic and program transformations. The tactics allow the application of two-sided rules, which require that the two commands of a judgment have the same shape, and one-sided rules, which operate on only one of the games in a judgment. 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 $x \stackrel{\$}{\leftarrow} X; y \leftarrow \mathcal{A}(z)$ and $y \leftarrow \mathcal{A}(z); x \stackrel{\$}{\leftarrow} X$, because instructions are executed in a different order. To mitigate this limitation, EasyCrypt implements program transformations for code motion, allowing to swap instructions that are independent. Moreover, EasyCrypt implements tactics for inlining procedure calls and eagerly/lazily sample random values. Basic tactics can be combined using tacticals to increase automation. The tactic language provides the necessary infrastructure for making most components of EasyCrypt proof-producing, as discussed below.

Reasoning about Failure Events Game-based proofs often include steps in which it is argued that two games G_1 and G_2 behave identically 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 state a more general version of the lemma using relational logic.

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 hypothesis 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 $x \leftarrow \mathcal{A}(\vec{e})$ 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 formal 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 prove their own specifications for one or more oracles or adversaries when needed, leaving the tool to infer the rest.

Computing Probabilities EasyCrypt can prove claims about the probability of events in games using properties of probability (e.g. inclusion-exclusion principle), arithmetic laws, and the rules [PrEq] and [PrLe] above, which allow deriving probability claims from valid relational judgments. We also implement a simple mechanism for computing probability bounds. This mechanism can establish, for instance, 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 significantly increases confidence in proof sketches by producing independently verifiable proofs, and 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 judgments of a proof sketch are compiled into pRHL derivations in CertiCrypt. Finally, the compiler generates for each claim in a proof sketch a Coq lemma that may need to be completed manually with justifications of the probability reasoning performed by EasyCrypt.

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) $\mathcal{G}$ of order q and a keyed hash function $\{H_k : \mathcal{G}^3 \to \mathbb{Z}_q\}_{k \in K}$ mapping triples of group elements into integers in $\mathbb{Z}_q$, key generation, encryption, and decryption are defined as follows:

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

```
$\mathcal{E}((k, g, \hat{g}, e, f, h), m) \overset{\text{def}}{=}$
  $u \xleftarrow{\$} \mathbb{Z}_q;\ a \leftarrow g^u;\ \hat{a} \leftarrow \hat{g}^u;\ c \leftarrow h^u \cdot m;$
  $v \leftarrow H_k(a, \hat{a}, c);\ d \leftarrow e^u \cdot f^{uv};$
  return $(a, \hat{a}, c, d)$
$\mathcal{D}((k, g, \hat{g}, x_1, x_2, y_1, y_2, z_1, z_2), (a, \hat{a}, c, d)) \overset{\text{def}}{=}$
  $v \leftarrow H_k(a, \hat{a}, c);$
  if $d = a^{x_1 + v y_1} \cdot \hat{a}^{x_2 + v y_2}$ then
    return $c / (a^{z_1} \cdot \hat{a}^{z_2})$
  else return $\perp$
```

We prove that the Cramer-Shoup cryptosystem is secure against adaptive chosen-ciphertext attacks (IND-CCA secure) in the standard model assuming the DDH problem is hard in the underlying group family and the hash function H is target collision-resistant (i.e., universal one-way).

Definition 1 (Target Collision-Resistance). *Let $\{H_k : A \to B\}_{k \in K}$ be a keyed family of hash functions. The advantage of an adversary $\mathcal{C}$ against the target collision-resistance of H is defined as*

$$\mathbf{Adv}^{\mathcal{C}}_{\mathit{TCR}} \overset{\mathit{def}}{=} \Pr[\mathit{TCR} : H_k(x) = H_k(y) \land x \neq y]$$

where the experiment TCR is defined by means of the following game:

$$\textbf{Game } \mathsf{TCR} : x \leftarrow \mathcal{C}_1(\,);\ k \xleftarrow{\$} K;\ y \leftarrow \mathcal{C}_2(k)$$

Definition 2 (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*

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

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

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

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

Theorem 1 (Security of Cramer-Shoup). *Let $\mathcal{A}$ be an adversary against the* IND-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 $\mathcal{G}$ and an adversary $\mathcal{C}$ against the target collision-resistance of the hash function H such that*

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

Figure 2 shows a proof sketch of the above theorem in 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 IND-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. We observe that all verification conditions that ensure the validity of this transformation can be discharged automatically using an SMT solver. 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

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

The next game in the sequence, G_4, removes the dependency of the adversary's output from bit b by choosing uniformly 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

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

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,

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

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

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

```
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
    $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};\ v \leftarrow H_k(a, \hat{a}, c)$;
    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
      bad $\leftarrow$ true; return $c/(a^{z_1} \cdot \hat{a}^{z_2})$
    else return $\perp$
  else return $\perp$
```

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

therefore, the probability of $\mathbf{bad}_1$ 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

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

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

```
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
   $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};\ v \leftarrow H_k(a, \hat{a}, c);$
   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 $\perp$
```

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

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

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

```
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
   $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};\ v \leftarrow H_k(a, \hat{a}, c);$
   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 $\perp$
```

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

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

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

Fig. 2. Proof sketch of the IND-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 $H_k(a, \hat{a}, c) = H_k(g^u, \hat{g}^{u'}, g^r)$ 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,

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

The proof concludes by showing that the probability in G_5 of $\mathbf{bad}$ being set while $\mathbf{bad}_2$ is not 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.

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

```
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 $\mathbf{bad}_1 \leftarrow$ 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 $\boldsymbol{L}_{\mathcal{D}} \leftarrow \gamma :: \boldsymbol{L}_{\mathcal{D}};\ v \leftarrow H_k(a, \hat{a}, c);$
   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
     $\mathbf{bad} \leftarrow$ true;
     if $v = H_k(g^u, \hat{g}^{u'}, g^r)$ then
       $\mathbf{bad}_2 \leftarrow$ true
   else return $\perp$
 else return $\perp$
```

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

$\Pr[\mathsf{G}_5 : \mathbf{bad} \wedge u \neq u'] \leq$

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

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

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

5 Limitations and Extensions

EasyCrypt is in its early stages of development; we briefly comment on some of its main limitations and possible extensions:

- Programming language: in comparison with CertiCrypt, the language of EasyCrypt lacks loops, recursive procedures, and drawing from skewed distributions. We do not see the need for extending the current language with recursive procedures. In contrast, we believe that more general forms for sampling and bounded loops are useful and foresee no specific difficulty in adding them to the language (note that annotating loops with invariants may be required for verification condition generation);

- Verifiable evidence: EasyCrypt only generates partial verifiable evidence. As there is currently no SMT solver that generates Coq proofs, the verification conditions are admitted in order to make the output derivations checkable by the Coq proof assistant. Making SMT solvers proof-producing is an active subject of research [21], and advances towards this goal shall benefit immediately to EasyCrypt;
- Computation of probability: EasyCrypt generates proof skeletons for claims about probability rather than fully machine-checked proofs. While it is entirely feasible to extend the compiler for justifying more reasonings, a more principled solution would require a tool that can symbolically compute the probability of an event in a distribution.

Further research into the theory of cryptographic proofs, in the line of [3], is needed to broaden the scope of applications and effectiveness of EasyCrypt. Essential goals include providing a formal account of useful reasoning principles, such as rewinding arguments or coin-fixing, and notions, such as statistical distance, that have not yet been considered in our setting.

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 part of the sequence of games needed to conclude a proof, and to build adversaries that justify reductions to cryptographic assumptions. 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.

Finally, Halevi [17] stresses that "the usefulness of (a) tool will depend crucially on the willingness of the customers (in this case the cryptographic community) to use it", and suggests on this account that an appropriate user interface will be a crucial component of the tool. We fully adhere to his view, and see building such an interface as an important objective for further work.

5.1 Comparison with CertiCrypt

Table 1 compares CertiCrypt and EasyCrypt on various security proofs formalized in both systems. Times are measured on a 2.8GHz Intel Core 2 Duo processor with 4GB of RAM under Mac OS X 10.6.7. For comparison, we show the size and checking time of CertiCrypt proofs extracted from EasyCrypt proof sketches. This is not an altogether fair comparison, because extracted proofs assume as axioms proof obligations checked by automated provers. As an experiment, we completed interactively the extracted proof of security of ElGamal encryption, thus obtaining a full proof verifiable under Coq. The resulting proof is 1173 long (meaning that only 43 lines are needed to prove in Coq the proof obligations checked by automated provers) and takes 25s to check.

Table 1. Comparison of proof size and checking time between CertiCrypt and EasyCrypt.

	CertiCrypt		**EasyCrypt**		**Extracted**	
	Lines	Time	Lines	Time	Lines	Time
ElGamal (IND-CPA)	565	45s	190	12s	1130	23s
Hashed ElGamal (IND-CPA)	1255	1m05s	243	33s	1772	41s
Full-Domain Hash (EF-CMA)	2035	5m46s	509	1m26s	2724	1m11s
Cramer-Shoup (IND-CCA)	n/a	n/a	1637	5m12s	5504	3m14s
OAEP (IND-CPA)	2451	3m27s	n/a	n/a	n/a	n/a
OAEP (IND-CCA)	11162	37m32s	n/a	n/a	n/a	n/a

6 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 (e.g. [1, 9, 14, 19]). 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. We have presented EasyCrypt, a new tool which provides the first flexible and automated framework for building machine-checkable cryptographic proofs, and illustrated its use through computer-aided security proofs of Hashed ElGamal encryption in the Random Oracle Model and the Cramer-Shoup cryptosystem in the standard model. These examples demonstrate that proofs in EasyCrypt are significantly easier and faster to build than in any previous tool, while providing guarantees similar to CertiCrypt. Overall, we believe that EasyCrypt makes an important step towards the adoption of computer-aided proofs by working cryptographers.

Acknowledgments We are grateful to Daniel Hedin and Anne Pacalet for their participation in the initial phases of the project, to Yassine Lakhnech and David Pointcheval for useful discussions, and to the anonymous reviewers for their insightful comments.

References

1. Backes, M., Maffei, M., Unruh, D.: Computationally sound verification of source code. In: 17th ACM conference on Computer and Communications Security, CCS 2010. pp. 387–398. ACM, New York (2010)
2. Barthe, G., D'Argenio, P., Rezk, T.: Secure information flow by self-composition. In: 17th IEEE workshop on Computer Security Foundations, CSFW 2004. pp. 100–114. IEEE Computer Society, Washington (2004)
3. Barthe, G., Daubignard, M., Kapron, B., Lakhnech, Y.: Computational indistinguishability logic. In: 17th ACM conference on Computer and Communications Security, CCS 2010. pp. 375–386. ACM, New York (2010)
4. Barthe, G., Grégoire, B., Heraud, S., Zanella Béguelin, S.: Formal certification of ElGamal encryption. A gentle introduction to CertiCrypt. In: 5th International workshop on Formal Aspects in Security and Trust, FAST 2008. Lecture Notes in Computer Science, vol. 5491, pp. 1–19. Springer, Berlin (2009)
5. Barthe, G., Grégoire, B., Lakhnech, Y., Zanella Béguelin, S.: Beyond provable security. Verifiable IND-CCA security of OAEP. In: Topics in Cryptology – CT-RSA 2011. Lecture Notes in Computer Science, vol. 6558, pp. 180–196. Springer, Berlin (2011)
6. Barthe, G., Grégoire, B., Zanella Béguelin, S.: Formal certification of code-based cryptographic proofs. In: 36th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2009. pp. 90–101. ACM, New York (2009)
7. Barthe, G., Hedin, D., Zanella Béguelin, S., Grégoire, B., Heraud, S.: A machine-checked formalization of Sigma-protocols. In: 23rd IEEE Computer Security Foundations symposium, CSF 2010. pp. 246–260. IEEE Computer Society, Los Alamitos, Calif. (2010)
8. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Advances in Cryptology – EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 409–426. Springer, Berlin (2006)
9. Bhargavan, K., Fournet, C., Gordon, A.D.: Modular verification of security protocol code by typing. In: 37th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL 2010. pp. 445–456. ACM (2010)
10. Blanchet, B., Jaggard, A.D., Scedrov, A., Tsay, J.K.: Computationally sound mechanized proofs for basic and public-key Kerberos. In: 15th ACM conference on Computer and Communications Security, CCS 2008. pp. 87–99. ACM, New York (2008)
11. Blanchet, B.: A computationally sound mechanized prover for security protocols. In: 27th IEEE symposium on Security and Privacy, S&P 2006. pp. 140–154. IEEE Computer Society (2006)
12. Blanchet, B., Pointcheval, D.: Automated security proofs with sequences of games. In: Advances in Cryptology – CRYPTO 2006. Lecture Notes in Computer Science, vol. 4117, pp. 537–554. Springer, Berlin (2006)
13. Conchon, S., Contejean, E., Kanig, J., Lescuyer, S.: CC(X): Semantic combination of congruence closure with solvable theories. Electronic Notes in Theoretical Computer Science 198(2), 51–69 (2008)
14. Cremers, C.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: 20th International Conference on Computer Aided Verification, CAV 2008. Lecture Notes in Computer Science, vol. 5123, pp. 414–418. Springer, Berlin (2008)

15. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. Tech. Rep. HPL-2003-148, HP Laboratories Palo Alto (2003)
16. Filliâtre, J.C.: The WHY verification tool: Tutorial and Reference Manual Version 2.28. Online – `http://why.lri.fr` (2010)
17. Halevi, S.: A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181 (2005)
18. Jonsson, B., Yi, W., Larsen, K.G.: Probabilistic extensions of process algebras. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 685–710. Elsevier, Amsterdam (2001)
19. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. of Comput. Secur. 6(1-2), 85–128 (1998)
20. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004)
21. Stump, A.: Proof checking technology for satisfiability modulo theories. Electr. Notes Theor. Comput. Sci. 228, 121–133 (2009)
22. The Coq development team: The Coq Proof Assistant Reference Manual Version 8.3. Online – `http://coq.inria.fr` (2010)
23. Zanella Béguelin, S.: Formal Certification of Game-Based Cryptographic Proofs. Ph.D. thesis, Ecole Nationale Supérieure des Mines de Paris – Mines ParisTech (2010)
24. Zanella Béguelin, S., Grégoire, B., Barthe, G., Olmedo, F.: Formally certifying the security of digital signature schemes. In: 30th IEEE symposium on Security and Privacy, S&P 2009. pp. 237–250. IEEE Computer Society, Los Alamitos, Calif. (2009)

A Input File for the Proof of Security of Hashed ElGamal

The following is an extract taken from the EasyCrypt input file corresponding to the proof of IND-CPA security of Hashed ElGamal described in Section 2:

```
type group

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

type skey    = int
type pkey    = group
type key     = skey * pkey
type message = bitstring{k}
type cipher  = group * bitstring{k}

op (*)  : group, group → group                     = mul
op (^)  : group, int → group                       = pow
op (^^) : bitstring{k}, bitstring{k} → bitstring{k} = xor

axiom pow_mul  : ∀(x:int, y:int). { (g^x)^y = g^(x*y) }
axiom xor_comm : ∀(x:bitstring{k}, y:bitstring{k}). { (x^^y) = (y^^x) }

...

adversary A1(pk:pkey) : message * message { group → message}
adversary A2(pk:pkey) : bool              { group → message}

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

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

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

  ...

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

  fun Main() : bool = {
    var sk : skey;
    var pk : pkey;
    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(c);
    return (b = b');
  }
}

game G1 = INDCPA
  var y' : group
  where  Main = {
    var m0, m1 : message;
    var c : cipher;
    var b, b' : bool;
    var x, y : int;
    var hy : message;
    var α : group;

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

equiv Fact1 : INDCPA.Main ~ G1.Main : {true} ⟹ ={res}
 inline KG, Enc; derandomize;
 auto inv ={L,LA};
 pop⟨2⟩ 1; repeat rnd; trivial;;
save;;

claim Pr1 : INDCPA.Main[res] = G1.Main[res] using Fact1
...
```