Formal certification of ElGamal encryption
A gentle introduction to CertiCrypt *

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

[1] IMDEA Software, Madrid, Spain, `Gilles.Barthe@imdea.org`
[2] Microsoft Research - INRIA Joint Centre, France
[3] INRIA Sophia Antipolis - Méditerranée, France,
`{Benjamin.Gregoire,Sylvain.Heraud,Santiago.Zanella}@inria.fr`

Abstract. CertiCrypt [1] is a framework that assists the construction of machine-checked cryptographic proofs that can be automatically verified by third parties. To date, CertiCrypt has been used to prove formally the exact security of widely studied cryptographic systems, such as the OAEP padding scheme and the Full Domain Hash digital signature scheme. The purpose of this article is to provide a gentle introduction to CertiCrypt. For concreteness, we focus on a simple but illustrative example, namely the semantic security of the Hashed ElGamal encryption scheme in both, the standard and the random oracle model.

1 Introduction

CertiCrypt [1] is a framework that assists the construction of machine-checked cryptographic proofs in the style advocated by *provable security* [2, 3]. According to this style, the interplay between the cryptographic system and the adversary must be specified precisely and the proof of security must be established rigorously, making explicit all the assumptions used in the process. CertiCrypt concentrates on the game-playing approach to cryptographic proofs [4–6]. This approach uses techniques that help reduce the complexity of cryptographic proofs by structuring them in steps of manageable size. To date, CertiCrypt has been used to prove formally the exact security of widely studied cryptographic systems, such as the OAEP padding scheme and the Full Domain Hash digital signature scheme, and to establish results of wide applicability to cryptographic proofs, such as the PRP/PRF Switching Lemma and the Fundamental Lemma of game-playing.

CertiCrypt is built on top of the general purpose proof assistant Coq [7], from which it inherits a high level of trustworthiness and the ability to provide independently verifiable evidence that proofs are correct. One long-term ambition of CertiCrypt is to contribute to increase confidence in cryptographic proofs. Indeed, constructing a correct security proof can be such a delicate task that some

* This work has been partially supported by the ANR project SCALP.

cryptographic systems are notorious for having flawed proofs that stood unchallenged for years. The situation is even worse, as there are concerns about the trustworthiness of cryptographic proofs in general [4, 5]. As a possible solution, Halevi [5] suggested the construction and use of dedicated tools, and singled out some desirable features and functionalities of these tools. In a sense, CertiCrypt provides a first step towards the completion of Halevi's programme, although it focuses more in delivering automation, expressiveness and high assurance, than in providing a user interface to sketch proofs.

The main difficulty in building a tool to certify cryptographic proofs is that they usually involve a broad set of concepts and reasoning methods, drawing on probability, group and complexity theory. In the case of the game-playing approach, proofs additionally rely on programming language semantics and program transformation and verification. While all these aspects are covered in CertiCrypt, this is the first time we address some essential details that arise when using the tool to build a concrete proof.

The purpose of this article is to provide a gentle introduction to CertiCrypt. We give a step-by-step presentation of security proofs for (Hashed) ElGamal encryption, in the hope of helping readers understand some fine-grained details of the framework. Following Shoup's introductory paper on game-based proofs [6], we provide proofs both in the standard model, assuming that the hash function is entropy smoothing, and in the random oracle model, assuming the hash function is indistinguishable from a truly random function. These proofs generalize our earlier proof of ElGamal encryption, which was presented briefly in [1].

2 Provable security and the game-playing technique

The aim of cryptography is to achieve a particular security goal, independently of the behavior of adversaries. However, one cannot just enumerate every way an adversary may behave to break the security goal and design a cryptographic system to counter them all. That methodology is bound to fail because adversaries will behave in unpredicted ways to overcome any anticipated countermeasures. Therefore, valid proofs must establish security against all feasible adversaries. As explained below, not all adversaries are feasible, and some restrictions on their abilities are necessary to construct a security proof. In particular, it must be assumed that the adversary is not omniscient (i.e. it does not know some secrets) nor omnipotent (i.e. it cannot perform arbitrarily expensive computations). Both assumptions will be formalized using access control and resource usage policies on the one hand, and complexity classes on the other hand.

In the flavour of the game-playing technique that is adopted by CertiCrypt, the security goal is expressed through a probabilistic program that captures the interaction between the cryptographic system and an adversary. In the context of this paper, we shall focus on public-key encryption schemes and on their semantic security (equivalently, IND-CPA security), which guarantees ciphertext indistinguishability against chosen plaintext attacks. Informally, a public-key encryption scheme is semantically secure if any feasible adversary that only knows the public key and that chooses a pair of messages (m_0, m_1), cannot distinguish a scenario where it is given an encryption of m_0 from a scenario where it is given an encryption of m_1. Clearly, a necessary condition for an encryption scheme to be semantically secure is to be probabilistic, because otherwise an adversary can just compare the encryption of m_0 with the ciphertext it is given to tell apart both scenarios. In a game-based setting, semantic security is specified by means of the following probabilistic program:

```
Game IND-CPA :
$(sk, pk) \leftarrow \mathsf{KG}(\ )$;
$(m_0, m_1) \leftarrow \mathcal{A}(pk)$;
$b \xleftarrow{\$} \{0, 1\}$;
$\zeta \leftarrow \mathsf{Enc}(pk, m_b)$;
$b' \leftarrow \mathcal{A}'(pk, \zeta)$;
$d \leftarrow b = b'$
```

Here, KG is the key generation algorithm of the scheme and Enc the encryption algorithm, whereas $\mathcal{A}$ and $\mathcal{A}'$ are procedures representing an adversary. In addition to the procedures that appear in the above program, the game may involve oracles that can be called by the adversary; e.g. in Hashed ElGamal the adversary is given access to a public hash oracle. The specification of the IND-CPA game is completed by stating that the adversary belongs to the class of probabilistic polynomial time (PPT) programs, and that has access to a global variable to maintain state, read-only access to pk, but does not have access to sk or b. There are two ways to control access to variables: one can declare a variable as local, in which case it shall only be accessible in the scope of the procedure, or global, in which case its access is restricted by an explicit policy.

The IND-CPA property states that the probability of an adversary guessing which message has been encrypted is not significantly higher than 1/2. The precise definition involves a security parameter η (which determines the scheme parameters) and requires that the probability of $d = 1$ holding at the end of the game, written $\Pr_{\mathsf{IND\text{-}CPA}^\eta}[d = 1]$, is negligibly close to 1/2 as a function of η. Formally, a function $\nu : \mathbb{N} \to \mathbb{R}$ is negligible iff

$$\mathsf{negligible}(\nu) \overset{\text{def}}{=} \forall c.\ \exists n_c.\ \forall n.\ n \geq n_c \Rightarrow |\nu(n)| \leq n^{-c}$$

We say that a function ν is negligibly close to a constant k when the function $\lambda\eta.|\nu(\eta) - k|$ is negligible.

The essence of the game-playing technique is to prove a security property, such as the IND-CPA security of an encryption scheme, through successive transformations of the original attack game. More precisely, proofs that follow the game-playing technique are organized as a sequence of transitions of the form $G, A \to G', A'$ where G and G' are games, and A and A' are events. The goal is to establish for each transition $\Pr_G[A] \leq f(\Pr_{G'}[A'])$, for some monotonic function f. By combining the consecutive inequalities drawn from each transition, one can extract from a game-based proof an inequality $\Pr_{G_0}[A_0] \leq f(\Pr_{G_n}[A_n])$. Thus, if G_0, A_0 denotes the original attack game and event, one can obtain a bound of $\Pr_{G_0}[A_0]$ from a bound of $\Pr_{G_n}[A_n]$.

In many cases, transitions $G, A \to G', A'$ are such that $\Pr_G[A] = \Pr_{G'}[A']$. Such transitions, which are called bridging steps, include semantics-preserving program transformations. Formally, semantics preservation is defined by means of probabilistic non-interference [8], since we are only interested in preserving the observable behavior of games. However, there are many cases in which semantics preservation is context-dependent; to account for such cases, it is necessary to resort to a relational logic that generalizes probabilistic non-interference and that allows to reason modulo pre- and postconditions.

Game-based proofs also rely frequently on failure events, which help bound the probability loss in transitions by the probability of a flag being raised. One essential tool to reason about failure events is the so-called Fundamental Lemma: given two games G_1 and G_2 whose code only differ after a certain bad flag is raised (i.e. after an assignment $\mathsf{bad} \leftarrow \mathsf{true}$, where bad is initially set to false and always remains raised once set), one can conclude that for any event A, $\Pr_{G_1}[A \wedge \neg\mathsf{bad}] = \Pr_{G_2}[A \wedge \neg\mathsf{bad}]$. This implies in turn

$$|\Pr_{G_1}[A] - \Pr_{G_2}[A]| \leq \Pr_{G_1}[\mathsf{bad}] = \Pr_{G_2}[\mathsf{bad}]$$

provided both games terminate with the same probability.

Finally, some transitions are justified by security assumptions. For instance, the proof in Section 4.2 relies on the Decisional Diffie-Hellman assumption or DDH assumption for short. For a family of finite cyclic groups, this assumption states that no efficient algorithm can distinguish between triples of the form (g^x, g^y, g^{xy}) and triples of the form (g^x, g^y, g^z), where x, y, z are uniformly sampled from $\mathbb{Z}_q$, q is the (prime) order of the group, and g a generator. One characteristic of game-based proofs is to formulate these assumptions using games; the DDH assumption is formulated as follows

Definition 1 (DDH assumption). *Consider the games*

```
Game DDH_0 :                          Game DDH_1 :
  $x, y \xleftarrow{\$} \mathbb{Z}_q$;                 $x, y, z \xleftarrow{\$} \mathbb{Z}_q$;
  $d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})$          $d \leftarrow \mathcal{B}(g^x, g^y, g^z)$
```

and define

$$\epsilon_{\mathsf{DDH}}(\eta) \overset{def}{=} |\Pr_{\mathsf{DDH}_0^\eta}[d = 1] - \Pr_{\mathsf{DDH}_1^\eta}[d = 1]|$$

Then, for every PPT *adversary* $\mathcal{B}$, ϵ_{DDH} *is a negligible function. Note that the semantics of the above games (and in particular the order q of the group) depends on the security parameter η.*

3 An introduction to **CertiCrypt**

The goal of this section is to provide a brief overview of the framework. We first present the syntax and semantics of the language used to describe games, and then the tools the framework provides to reason about them.

3.1 Syntax and semantics of games

The lowest layer of CertiCrypt is the formalization of a probabilistic programming language with procedure calls. Given a set $\mathcal{V}$ of variables and a set $\mathcal{P}$ of procedure names, commands can be defined inductively by the clauses:

$$\begin{array}{rll}
\mathcal{I} ::= & \mathcal{V} \leftarrow \mathcal{E} & \text{deterministic assignment} \\
\mid & \mathcal{V} \xleftarrow{\$} \mathcal{D} & \text{random assignment} \\
\mid & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\
\mid & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{while loop} \\
\mid & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \dots, \mathcal{E}) & \text{procedure call} \\
\mathcal{C} ::= & \mathsf{nil} & \text{nop} \\
\mid & \mathcal{I};\ \mathcal{C} & \text{sequence}
\end{array}$$

where $\mathcal{E}$ is the set of expressions and $\mathcal{D}$ is the set of distributions from which values can be sampled in random assignments. Common data types and operators are provided, but in order to adapt to different settings, the syntax is user-extensible: users can define new data types and operations by providing an adequate interpretation in terms of Coq constructions. In addition, the syntax is typed, so that operators and expressions have a total semantics.

Games consist of a main command and an environment that maps a procedure identifier to its declaration, consisting of a list of formal parameters, a body, and a return expression (we use an explicit return when writing games, though),

$$\mathsf{declaration} \stackrel{\text{def}}{=} \{\mathsf{params} : V^\star;\ \mathsf{body} : \mathcal{C};\ \mathsf{re} : \mathcal{E}\}$$

Formally, the type of games is $\mathcal{C} \times (\mathcal{P} \to \mathsf{declaration})$. The semantics of games is defined using the measure monad $M(X)$ of Audebaud and Paulin [9]; its type constructor, unit and binding are defined as:

$$\begin{array}{lcl}
M(X) & \stackrel{\text{def}}{=} & (X \to [0,1]) \to [0,1] \\
\mathsf{unit} & : & X \to M(X) \stackrel{\text{def}}{=} \lambda x.\ \lambda f.\ f\ x \\
\mathsf{bind} & : & M(X) \to (X \to M(Y)) \to M(Y) \\
 & \stackrel{\text{def}}{=} & \lambda\mu.\ \lambda M.\ \lambda f.\ \mu(\lambda\ x.\ M\ x\ f)
\end{array}$$

This monad can be viewed as a specialization of the continuation monad, and allows to provide a continuation-passing style semantics of games. Intuitively, an element in $M(X)$ may be interpreted as the expectation operator of a (sub) probability distribution on X. Thus, the denotation of a game relates an initial memory to the expectation operator of the (sub) probability distribution of final memories that results from its execution. The denotational semantics of games is defined internally by means of a small-step semantics that uses frames to deal with procedure calls. From a user point of view, however, these details can be ignored without hindering understanding; the formal definition of small-step semantics can be found in [1]. The denotation of games is presented in Fig. 1; in the figure we represent a memory m as a pair $(m.\mathsf{loc}, m.\mathsf{glob})$, making explicit its local and global components. Expressions are deterministic and their semantics is given by a function $[\![\cdot]\!]_{\mathcal{E}}$ that evaluates an expression in a given memory and returns a value. The semantics of distributions in $\mathcal{D}$ is given by another function $[\![\cdot]\!]_{\mathcal{D}}$; we give as examples the semantics of the uniform distribution on $\mathbb{B}$ and on integer intervals of the form $[0..n]$. In the figure, we have omitted the procedure environment E for the sake of readability. In the remainder we will frequently make no distinction between a game $G = (c, E)$ and its main command c when the environment where it is evaluated either has no relevance, or is clear from the context.

$$\begin{array}{ll}
[\![\mathsf{nil}]\!]\ m & = \mathsf{unit}\ m \\
[\![i;\ c]\!]\ m & = \mathsf{bind}\ ([\![i]\!]\ m)\ [\![c]\!] \\
[\![x \leftarrow e]\!]\ m & = \mathsf{unit}\ m\{[\![e]\!]_{\mathcal{E}}\ m/x\} \\
[\![x \overset{\$}{\leftarrow} d]\!]\ m & = \mathsf{bind}\ ([\![d]\!]_{\mathcal{D}}\ m)\ (\lambda v.\ \mathsf{unit}\ m\{v/x\}) \\
[\![x \leftarrow f(\boldsymbol{e})]\!]\ m & = \begin{array}{l}\mathsf{bind}\ ([\![E(f).\mathsf{body}]\!]\ (\emptyset\{[\![\boldsymbol{e}]\!]_{\mathcal{D}}\ m/E(f).\mathsf{params}\}, m.\mathsf{glob})) \\ \qquad (\lambda m'.\ (m.\mathsf{loc}, m'.\mathsf{glob})\{[\![E(f).\mathsf{re}]\!]_{\mathcal{E}}\ m'/x\})\end{array} \\
[\![\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\ m & = \begin{cases}[\![c_1]\!]\ \mathsf{if}\ [\![e]\!]_{\mathcal{E}}\ m = \mathsf{true} \\ [\![c_2]\!]\ \mathsf{if}\ [\![e]\!]_{\mathcal{E}}\ m = \mathsf{false}\end{cases} \\
[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\ m & = [\![\mathsf{if}\ e\ \mathsf{then}\ c;\ \mathsf{while}\ e\ \mathsf{do}\ c]\!]\ m \\
 & \\
[\![\{0,1\}]\!]_{\mathcal{D}}\ m & = \lambda f.\ \dfrac{1}{2}\ f(\mathsf{true}) + \dfrac{1}{2}\ f(\mathsf{false}) \\
[\![[0..e]]\!]_{\mathcal{D}}\ m & = \lambda f.\displaystyle\sum_{i=0}^{n} \frac{1}{n+1}\ f(i) \quad \text{where } n = [\![e]\!]_{\mathcal{E}}\ m
\end{array}$$

Fig. 1. Denotational semantics of games

CertiCrypt provides an alternative, more convenient rule for *while* loops:

$$[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\ m\ f = \sup\{[\![[\mathsf{while}\ e\ \mathsf{do}\ c]_n]\!]\ m\ f : n \in \mathbb{N}\}$$

where $[\mathsf{while}\ e\ \mathsf{do}\ c]_n$ is the n-step unrolling of the loop, i.e.

$$\begin{array}{lcl}
[\mathsf{while}\ e\ \mathsf{do}\ c]_0 & = & \mathsf{nil} \\
[\mathsf{while}\ e\ \mathsf{do}\ c]_{n+1} & = & \mathsf{if}\ e\ \mathsf{then}\ c;\ [\mathsf{while}\ e\ \mathsf{do}\ c]_n
\end{array}$$

Note that the function $[\![\cdot]\!]$ maps $\mathcal{M}$ to $M(\mathcal{M})$, but it is trivial to define a semantic function $[\![\cdot]\!]'$ from $M(\mathcal{M})$ to $M(\mathcal{M})$ using the bind operator of the monad: $[\![G]\!]'\ \mu \stackrel{\text{def}}{=} \mathsf{bind}\ \mu\ [\![G]\!]$. One of the major advantages of using the monad $M(\mathcal{M})$ is that the probability of an event A, represented as a Boolean predicate over memories, can be readily defined using the characteristic function $\mathbb{I}_A$ of A:

$$\Pr_{G,m}[A] \stackrel{\text{def}}{=} [\![G]\!]\ m\ \mathbb{I}_A \tag{1}$$

In what follows, we sometimes omit the initial memory m; in that case one may safely assume that the memory initially maps variables to default values of the right type.

3.2 Reasoning about games

In game-based proofs, bridging steps correspond in a sense to semantics preserving transformations; they are used to restate the way certain quantities are computed to prepare the ground for a subsequent transformation. Hence, in a bridging step from G, A to G', A' the goal is to establish $\Pr_{G,m}[A] = \Pr_{G',m}[A']$. If we take a look at definition (1), this amounts to proving that $[\![G]\!]\ m\ \mathbb{I}_A = [\![G']\!]\ m\ \mathbb{I}_{A'}$, or generalizing this to a pair of initial memories m_1, m_2 and arbitrary functions $f, g : \mathcal{M} \to [0,1]$, that $[\![G]\!]\ m_1\ f = [\![G']\!]\ m_2\ g$.

The main tool CertiCrypt provides to establish such equalities is the relational logic pRHL, which generalizes Relational Hoare Logic [10] to a probabilistic setting. Judgments in pRHL are of the form $\models G_1 \sim G_2 : \Psi \Rightarrow \Phi$, where G_1 and G_2 are games, and Ψ and Φ are relations over deterministic states. A judgment $\models G_1 \sim G_2 : \Psi \Rightarrow \Phi$ is valid iff for every pair of initial memories m_1, m_2 such that $m_1\ \Psi\ m_2$, $[\![G_1]\!]\ m_1 \sim_\Phi [\![G_2]\!]\ m_2$ holds. The relation $\sim_\Phi$ is a lifting of Φ to measures. If Φ is a PER, the definition of $\sim_\Phi$ is rather intuitive:

$$\mu_1 \sim_\Phi \mu_2 \ \stackrel{\text{def}}{=}\ \forall a.\ \mu_1\ \mathbb{I}_{[a]} = \mu_2\ \mathbb{I}_{[a]}$$

where $\mathbb{I}_{[a]}$ is the characteristic function of the equivalence class of a. The definition of $\sim_\Phi$ for arbitrary relations is less immediate, and involves an existential quantification:

$$\begin{array}{l} \mathsf{range}\ P\ \mu \stackrel{\text{def}}{=} \forall f.\ (\forall a.\ P\ a \Rightarrow f\ a = 0) \Rightarrow \mu\ f = 0 \\ \mu_1 \sim_\Phi \mu_2 \ \stackrel{\text{def}}{=}\ \exists \mu.\ \pi_1(\mu) = \mu_1 \land \pi_2(\mu) = \mu_2 \land \mathsf{range}\ \Phi\ \mu \end{array}$$

where the projections of μ are defined as

$$\pi_1(\mu) \stackrel{\text{def}}{=} \mathsf{bind}\ \mu\ (\lambda p.\mathsf{unit}\ (\mathsf{fst}\ p)) \quad \pi_2(\mu) \stackrel{\text{def}}{=} \mathsf{bind}\ \mu\ (\lambda p.\mathsf{unit}\ (\mathsf{snd}\ p))$$

This definition stems from work on probabilistic bisimulations, and generalizes lifting to arbitrary relations. Both definitions coincide for PERs [11].

In order to reason about pRHL judgments, CertiCrypt provides a set of derived rules and a (partial) weakest precondition calculus. The rules can be found in [1]. An important implication of a pRHL judgment $\models G_1 \sim G_2 : \Psi \Rightarrow \Phi$, is that if two functions f and g are unable to distinguish memories in the Φ relation, i.e.

$$\forall m_1\ m_2.\ m_1\ \Phi\ m_2 \Rightarrow f\ m_1 = g\ m_2$$

then

$$\forall m_1\ m_2.\ m_1\ \Psi\ m_2 \Rightarrow [\![G_1]\!]\ m_1\ f = [\![G_2]\!]\ m_2\ g \qquad (=_{[\![]\!]})$$

In particular, if Φ is the equality on the free variables of a Boolean predicate A, we obtain $\Pr_{G_1,m_1}[A] = \Pr_{G_2,m_2}[A]$. This property extends to the $\leq$ relation.

By specializing pRHL judgments to equality predicates on sets of variables, one recovers probabilistic non-interference: given a set X of variables, define

$$m_1 =_X m_2 \stackrel{\text{def}}{=} \forall x \in X, m_1\ x = m_2\ x$$

Probabilistic non-interference w.r.t. a set I of input variables and a set O of output variables is defined as $\models \cdot \sim \cdot : =_I \Rightarrow =_O$, we use $\models \cdot \simeq^I_O \cdot$ as a shorthand.

CertiCrypt provides several tools to reason about non-interference. In particular, CertiCrypt implements several tactics that help establish non-interference or reduce it to a simpler goal. For example, the tactic `eqobs_in` implements a semi-decision procedure for judgments of the form $\models c, E \simeq^I_O c, E'$. Other tactics, such as `eqobs_hd`, `eqobs_tl`, `eqobs_ctxt`, `deadcode`, and `swap` simplify the goal by using functions that take games c_1, E_1 and c_2, E_2 and sets of variables I,O and return c'_1, c'_2 and I', O' such that

$$\models c'_1, E_1 \simeq^{I'}_{O'} c'_2, E_2 \quad \Rightarrow \quad \models c_1, E_1 \simeq^I_O c_2, E_2$$

The tactics differ in their strategy to compute c'_1, c'_2 and I', O'. Tactic `eqobs_tl` searches for a maximal common prefix c such that $c_1 = c; c'_1$ and $c_2 = c; c'_2$, `eqobs_hd` searches similarly for a maximal suffix, and `eqobs_ctxt` combines both. The tactic `swap` rearranges instructions in programs to generate a largest common suffix while preserving observational equivalence, i.e. $c'_1 = \hat{c}_1; c$ and $c'_2 = \hat{c}_2; c$ are permutations of c_1 and c_2 (and $I' = I$ and $O' = O$). The tactic `deadcode` produces slices of the original commands using the variables in O as slicing criteria.

In addition, CertiCrypt automates other common program transformations: expression propagation (`ep`), variable allocation (`alloc`), and inlining (`inline`). These tactics are shown to preserve non-interference. The tactic `sinline` combines `inline`, `alloc`, `ep`, and `deadcode` in one powerful tactic.

To be able to deal with procedure calls the tactics need information about procedures in the environment of games. This information cannot be computed recursively due to the presence of adversaries whose code is unknown. Given two environments E_1 and E_2, and a procedure f, tactics assume the following information is given:

– For each environment: a set W_i of global variables that f might modify, sets I_i and O_i of global variables, and a subset P_i of its formal parameters such that for every execution of the body of the procedure, the final values of variables in $O_i \cup \mathsf{fv}(E_i(f).\mathsf{re})$ depend only on the initial values of variables in $I_i \cup P_i$. Formally,

$$\begin{array}{l} W_i = \mathsf{globals}(\mathsf{modifies}(E_i(f).\mathsf{body}, E_i)) \wedge \\ P_i \subseteq E_i(f).\mathsf{params} \wedge \\ \models E_i(f).\mathsf{body}, E_i \simeq^{I_i \cup P_i}_{O_i \cup \mathsf{fv}(E_i(f).\mathsf{re})} E_i(f).\mathsf{body}, E_i \end{array}$$

(modifies computes an over approximation of the variables modified by a piece of code.) This information is used by the tactics `swap`, `deadcode`, `ep`, and `inline`;

– Relational information: sets I and O of global variables, and a subset P of the formal parameters of f such that the execution of the body of f in each environment, starting from memories equal on variables in $I \cup P$, results in measures equivalent on $O \cup \mathsf{fv}(E_i(f).\mathsf{re})$. We further require that $E_1(f).\mathsf{re} = E_2(f).\mathsf{re}$. Formally,

$$\begin{array}{l} P \subseteq E_1.(f).\mathsf{params} \ \wedge \ P \subseteq E_2.(f).\mathsf{params} \wedge \\ \models E_1(f).\mathsf{body}, E_1 \simeq^{I \cup P}_{O \cup \mathsf{fv}(E_i(f).\mathsf{re})} E_2(f).\mathsf{body}, E_2 \end{array}$$

This information is used by tactics `eqobs_in`, `eqobs_hd`, and `eqobs_tl`.

CertiCrypt provides several mechanisms to build the above information incrementally and automatically when the bodies of procedures in E_1 and E_2 are observationally equivalent modulo expression propagation and dead code elimination. It is also possible to derive the information for an adversary from the information about the oracles it may call. This is possible provided the adversary is well-formed, since in this case we know that the adversary and any subprocedures it may call respect an access control policy $(\mathcal{O}, \mathcal{RO}, \mathcal{RW})$: they may only call oracles in $\mathcal{O}$, read global variables in $\mathcal{RO}$, and read or modify global variables in $\mathcal{RW}$.

As said before, some transformations performed during proofs are context-dependent. CertiCrypt allows for a rich specification of the context in which a transformation is valid using program invariants. Tactics are thus extended to deal with invariants on global variables; the information they use is specified instead by judgments of the form

$$\models c_1, E_1 \sim c_2, E_2 : =_I \wedge \phi \Rightarrow =_O \wedge \phi$$

4 Semantic security of Hashed ElGamal encryption

Let G be a cyclic group of prime order q and g a generator, and let $(H_k)_{k \in K}$ be a family of keyed hash functions mapping elements in G to bitstrings of a certain length ℓ. Hashed ElGamal is a public-key encryption scheme whose security is believed to be related to the discrete logarithm problem in G. Its key generation, encryption and decryption algorithms are defined as follows:

$$\begin{array}{lcl} \mathsf{KG}(\ ) & \stackrel{\text{def}}{=} & k \stackrel{\$}{\leftarrow} K;\ x \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ \mathsf{return}\ ((k, x), (k, g^x)) \\ \mathsf{Enc}(k, \alpha, m) & \stackrel{\text{def}}{=} & y \stackrel{\$}{\leftarrow} \mathbb{Z}_q;\ h \leftarrow H_k(\alpha^y);\ \mathsf{return}\ (g^y, h \oplus m) \\ \mathsf{Dec}(k, x, \beta, \zeta) & \stackrel{\text{def}}{=} & h \leftarrow H_k(\beta^x);\ \mathsf{return}\ h \oplus \zeta \end{array}$$

The plaintext space of Hashed ElGamal is $\{0,1\}^\ell$, in contrast to the original ElGamal encryption scheme whose plaintext space is simply G.

In the remainder of this section we present game-based proofs of the semantic security of Hashed ElGamal encryption in two different settings. The first proof is done in the standard model of cryptography; it assumes that the family $(H_k)_{k \in K}$ of hash functions is entropy smoothing and reduces semantic security to the hardness of the DDH problem. The second proof is done in the Random Oracle Model (ROM); it assumes hash functions behave as perfectly random functions and reduces semantic security to the hardness of the (list) CDH problem.

To formalize the proofs in CertiCrypt we first need to extend the syntax and semantics of games to include the types and operators used in the description of the scheme that are not already defined. As explained in Sec. 3, this is done in a modular way. We declare a family of cyclic groups $(G_\eta)_{\eta\in\mathbb{N}}$ indexed by the security parameter and extend the types of the language with user-defined types for elements in G_η and bitstrings of length ℓ. We extend $\mathcal{D}$ with the uniform distribution on bitstrings of length ℓ. We finally extend the language operators with nullary operators q and g to retrieve the order and a generator of G_η respectively, binary operators for the product and power in the group, and $\oplus$ for exclusive or on bitstrings of length ℓ. For the security proof in the standard model, we represent the hash function of the scheme as a binary operator taking a key in K and a value in G_η and returning a bitstring of length ℓ, whereas in the proof in the random oracle model we directly encode the hash function as a procedure and no further extensions are needed.

4.1 Security in the standard model

The proof we present next relies on two assumptions: the assumption that the family of hash functions $(H_k)_{k\in K}$ is entropy smoothing, and the hardness of the DDH problem in G_η. The latter assumption was already formalized in Sec. 2 as Definition 1. The former is formally stated below.

Definition 2 (Entropy Smoothing (ES) assumption). *Consider the games*

```
Game ES_0 :
 $\boldsymbol{k} \xleftarrow{\$} K;\ h \xleftarrow{\$} \{0,1\}^\ell;$
 $d \leftarrow \mathcal{D}(h)$
```

```
Game ES_1 :
 $\boldsymbol{k} \xleftarrow{\$} K;\ z \xleftarrow{\$} \mathbb{Z}_q;$
 $d \leftarrow \mathcal{D}(H(\boldsymbol{k}, g^z))$
```

and define

$$\epsilon_{\mathsf{ES}}(\eta) \overset{def}{=} |\Pr_{\mathsf{ES}_0}[d=1] - \Pr_{\mathsf{ES}_1}[d=1]|$$

Then, for every PPT *adversary $\mathcal{D}$, ϵ_{ES} is a negligible function.*

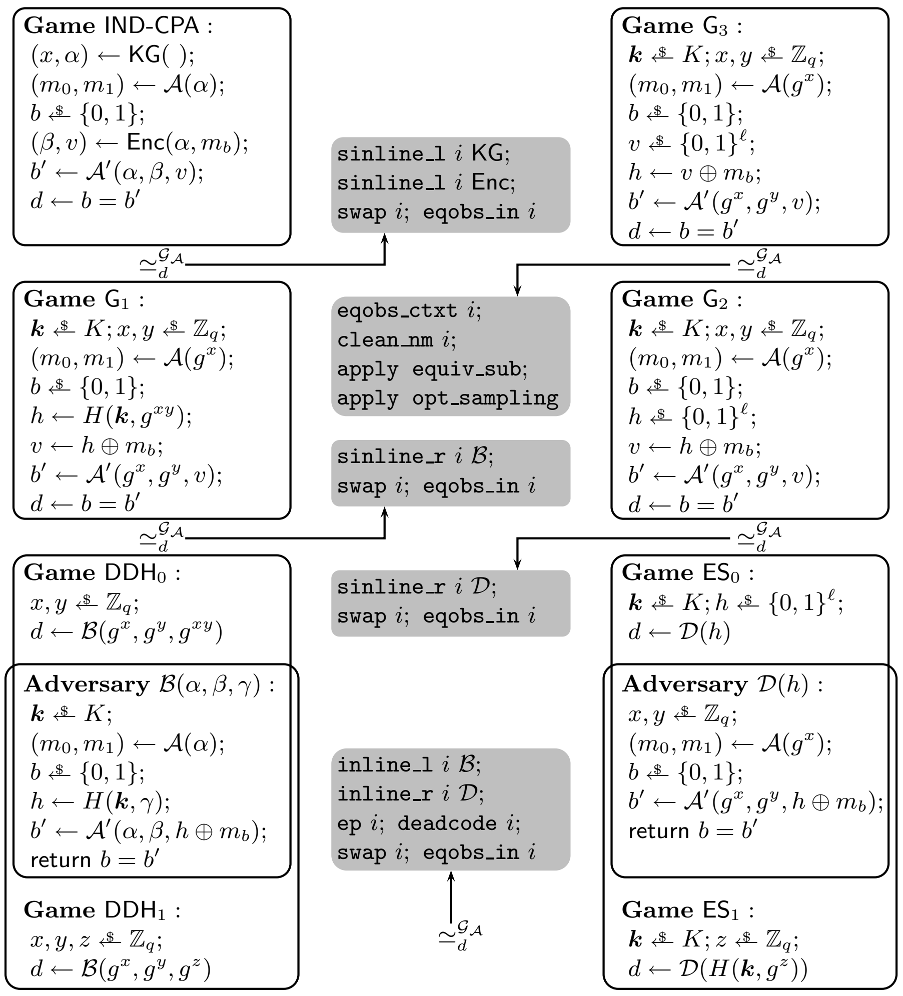
To avoid cluttering the description of games, we slightly modify the presentation of the key generation algorithm: instead of returning the hash key as a component of the secret and public key, we model it as a global variable $\boldsymbol{k}$. This will allow us at the same time to nicely illustrate the use of global variables in CertiCrypt.

Theorem 1 (Security of Hashed ElGamal in the standard model). *For every* PPT *and well-formed adversary $(\mathcal{A}, \mathcal{A}')$,*

$$\left|\Pr_{\mathsf{IND\text{-}CPA}}[d] - \frac{1}{2}\right| \le \epsilon_{\mathsf{DDH}}(\eta) + \epsilon_{\mathsf{ES}}(\eta)$$

Furthermore, under the DDH *and* ES *assumptions, $\Pr_{\mathsf{IND\text{-}CPA}}[d]$ is negligibly close to $\frac{1}{2}$.*

Figure 2 gives an overview of the proof; proof scripts appear inside grey boxes. We model the adversary as two procedures sharing state via global variables in $\mathcal{G}_\mathcal{A}$. The well-formedness condition simply states that the adversary has read-only access to $\boldsymbol{k}$ and that it cannot call procedures named $\mathcal{B}$ or $\mathcal{D}$ as these are the names reserved for adversaries in the reduction. Note that this is without loss of generality, and the adversary is free to define and call any other private procedures of its own as long as they are also well-formed. The information i used by the tactics is thus inferred automatically.



```
Game IND-CPA :
  $(x, \alpha) \leftarrow \mathsf{KG}(\,)$;
  $(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $(\beta, v) \leftarrow \mathsf{Enc}(\alpha, m_b)$;
  $b' \leftarrow \mathcal{A}'(\alpha, \beta, v)$;
  $d \leftarrow b = b'$
```

```
sinline_l $i$ KG;
sinline_l $i$ Enc;
swap $i$; eqobs_in $i$
```

```
Game G$_3$ :
  $\boldsymbol{k} \xleftarrow{\$} K; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $v \xleftarrow{\$} \{0,1\}^\ell$;
  $h \leftarrow v \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$;
  $d \leftarrow b = b'$
```

```
Game G$_1$ :
  $\boldsymbol{k} \xleftarrow{\$} K; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \leftarrow H(\boldsymbol{k}, g^{xy})$;
  $v \leftarrow h \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$;
  $d \leftarrow b = b'$
```

```
eqobs_ctxt $i$;
clean_nm $i$;
apply equiv_sub;
apply opt_sampling
```

```
sinline_r $i$ $\mathcal{B}$;
swap $i$; eqobs_in $i$
```

```
Game G$_2$ :
  $\boldsymbol{k} \xleftarrow{\$} K; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \xleftarrow{\$} \{0,1\}^\ell$;
  $v \leftarrow h \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$;
  $d \leftarrow b = b'$
```

```
Game DDH$_0$ :
  $x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})$
```

```
sinline_r $i$ $\mathcal{D}$;
swap $i$; eqobs_in $i$
```

```
Game ES$_0$ :
  $\boldsymbol{k} \xleftarrow{\$} K; h \xleftarrow{\$} \{0,1\}^\ell$;
  $d \leftarrow \mathcal{D}(h)$
```

```
Adversary $\mathcal{B}(\alpha, \beta, \gamma)$ :
  $\boldsymbol{k} \xleftarrow{\$} K$;
  $(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \leftarrow H(\boldsymbol{k}, \gamma)$;
  $b' \leftarrow \mathcal{A}'(\alpha, \beta, h \oplus m_b)$;
  return $b = b'$
```

```
inline_l $i$ $\mathcal{B}$;
inline_r $i$ $\mathcal{D}$;
ep $i$; deadcode $i$;
swap $i$; eqobs_in $i$
```

```
Adversary $\mathcal{D}(h)$ :
  $x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, h \oplus m_b)$;
  return $b = b'$
```

```
Game DDH$_1$ :
  $x, y, z \xleftarrow{\$} \mathbb{Z}_q$;
  $d \leftarrow \mathcal{B}(g^x, g^y, g^z)$
```

```
Game ES$_1$ :
  $\boldsymbol{k} \xleftarrow{\$} K; z \xleftarrow{\$} \mathbb{Z}_q$;
  $d \leftarrow \mathcal{D}(H(\boldsymbol{k}, g^z))$
```

$\simeq_d^{\mathcal{G}_\mathcal{A}}$
$\simeq_d^{\mathcal{G}_\mathcal{A}}$
$\simeq_d^{\mathcal{G}_\mathcal{A}}$
$\simeq_d^{\mathcal{G}_\mathcal{A}}$
$\simeq_d^{\mathcal{G}_\mathcal{A}}$

Fig. 2. Game-based proof of semantic security of Hashed ElGamal encryption in the standard model

We begin by proving

$$\Pr_{\mathsf{IND\text{-}CPA}}[d] = \Pr_{\mathsf{DDH}_0}[d] \qquad (2)$$

For clarity, we introduce an intermediate game G_1 and show that

$$\models \mathsf{IND\text{-}CPA} \simeq_d^{\mathcal{G}_\mathcal{A}} \mathsf{G}_1 \quad \text{and} \quad \models \mathsf{G}_1 \simeq_d^{\mathcal{G}_\mathcal{A}} \mathsf{DDH}_0$$

Since the $\simeq$ relation is transitive,

$$\frac{\models c \simeq_O^I c' \quad \models c' \simeq_O^I c''}{\models c \simeq_O^I c''}\,[\mathsf{R\text{-}Trans}]$$

we obtain $\models \mathsf{IND\text{-}CPA} \simeq_d^{\mathcal{G}_\mathcal{A}} \mathsf{DDH}_0$. Equation (2) follows then from $(=_{[\![]\!]})$. Next we show that

$$\Pr_{\mathsf{DDH}_1}[d] = \Pr_{\mathsf{ES}_1}[d] \qquad (3)$$

To this end, we prove first $\models \mathsf{DDH}_1 \simeq_d^{\mathcal{G}_\mathcal{A}} \mathsf{ES}_1$. We illustrate this transition in detail, showing the intermediate goals obtained after applying each tactic in the proof script:

$$\boxed{x, y, z \xleftarrow{\$} \mathbb{Z}_q;\ d \leftarrow \mathcal{B}(g^x, g^y, g^z)} \simeq_d^{\mathcal{G}_\mathcal{A}} \boxed{\boldsymbol{k} \xleftarrow{\$} K;\ z \xleftarrow{\$} \mathbb{Z}_q;\ d \leftarrow \mathcal{D}(H(\boldsymbol{k}, g^z))}$$

$\texttt{inline\_l}\ i\ \mathcal{B};\ \texttt{inline\_r}\ i\ \mathcal{D}$

$$\boxed{\begin{array}{l} x, y, z \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x; \beta \leftarrow g^y; \gamma \leftarrow g^z; \\ \boldsymbol{k} \xleftarrow{\$} K;\ (m_0, m_1) \leftarrow \mathcal{A}(\alpha); \\ b \xleftarrow{\$} \{0,1\};\ h \leftarrow H(\boldsymbol{k}, \gamma); \\ b' \leftarrow \mathcal{A}'(\alpha, \beta, h \oplus m_b);\ d \leftarrow b = b' \end{array}} \simeq_d^{\mathcal{G}_\mathcal{A}} \boxed{\begin{array}{l} \boldsymbol{k} \xleftarrow{\$} K;\ z \xleftarrow{\$} \mathbb{Z}_q;\ h \leftarrow H(\boldsymbol{k}, g^z); \\ x, y \xleftarrow{\$} \mathbb{Z}_q;\ (m_0, m_1) \leftarrow \mathcal{A}(g^x); \\ b \xleftarrow{\$} \{0,1\}; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, h \oplus m_b);\ d \leftarrow b = b' \end{array}}$$

$\texttt{ep}\ i$

$$\boxed{\begin{array}{l} x, y, z \xleftarrow{\$} \mathbb{Z}_q;\ \alpha \leftarrow g^x; \beta \leftarrow g^y; \gamma \leftarrow g^z; \\ \boldsymbol{k} \xleftarrow{\$} K;\ (m_0, m_1) \leftarrow \mathcal{A}(g^x); \\ b \xleftarrow{\$} \{0,1\};\ h \leftarrow H(\boldsymbol{k}, g^z); \\ b' \leftarrow \mathcal{A}'(g^x, g^y, H(\boldsymbol{k}, g^z) \oplus m_b); \\ d \leftarrow b = b' \end{array}} \simeq_d^{\mathcal{G}_\mathcal{A}} \boxed{\begin{array}{l} \boldsymbol{k} \xleftarrow{\$} K;\ z \xleftarrow{\$} \mathbb{Z}_q;\ h \leftarrow H(\boldsymbol{k}, g^z); \\ x, y \xleftarrow{\$} \mathbb{Z}_q;\ (m_0, m_1) \leftarrow \mathcal{A}(g^x); \\ b \xleftarrow{\$} \{0,1\}; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, H(\boldsymbol{k}, g^z) \oplus m_b); \\ d \leftarrow b = b' \end{array}}$$

$\texttt{deadcode}\ i$

$$\boxed{\begin{array}{l} x, y, z \xleftarrow{\$} \mathbb{Z}_q; \\ \boldsymbol{k} \xleftarrow{\$} K;\ (m_0, m_1) \leftarrow \mathcal{A}(g^x); \\ b \xleftarrow{\$} \{0,1\}; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, H(\boldsymbol{k}, g^z) \oplus m_b); \\ d \leftarrow b = b' \end{array}} \simeq_d^{\mathcal{G}_\mathcal{A}} \boxed{\begin{array}{l} \boldsymbol{k} \xleftarrow{\$} K;\ z \xleftarrow{\$} \mathbb{Z}_q; \\ x, y \xleftarrow{\$} \mathbb{Z}_q;\ (m_0, m_1) \leftarrow \mathcal{A}(g^x); \\ b \xleftarrow{\$} \{0,1\}; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, H(\boldsymbol{k}, g^z) \oplus m_b); \\ d \leftarrow b = b' \end{array}}$$

$\texttt{swap}\ i$

$$\boxed{\begin{array}{l} x, y, z \xleftarrow{\$} \mathbb{Z}_q; \\ \boldsymbol{k} \xleftarrow{\$} K;\ (m_0, m_1) \leftarrow \mathcal{A}(g^x); \\ b \xleftarrow{\$} \{0,1\}; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, H(\boldsymbol{k}, g^z) \oplus m_b); \\ d \leftarrow b = b' \end{array}} \simeq_d^{\mathcal{G}_\mathcal{A}} \boxed{\begin{array}{l} x, y, z \xleftarrow{\$} \mathbb{Z}_q; \\ \boldsymbol{k} \xleftarrow{\$} K;\ (m_0, m_1) \leftarrow \mathcal{A}(g^x); \\ b \xleftarrow{\$} \{0,1\}; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, H(\boldsymbol{k}, g^z) \oplus m_b); \\ d \leftarrow b = b' \end{array}}$$

$\texttt{eqobs\_in}\ i$

We first inline the calls to $\mathcal{B}$ and $\mathcal{D}$ in each game. When inlining a procedure call, the expressions appearing in the list of actual parameters are assigned to the corresponding formal parameters appearing in its declaration and the return expression is assigned to the return variable. We use `ep` to propagate assignments along the game, and `deadcode` to eliminate instructions that do not affect — either directly or indirectly— the value of d. (The tactic `sinline` would achieve the same result as this combination of tactics.) At this point the resulting programs are the same modulo reordering of instructions; we use `swap` to rearrange the instructions of the program in the right hand side in the same order as in the program in the left hand side. The tactic `eqobs_in` concludes by performing a dependency analysis to show that in the resulting program the value of d depends only on the initial value of variables in $\mathcal{G}_{\mathcal{A}}$.

Finally, we show that

$$\Pr_{\mathsf{ES}_0}[d] = \Pr_{\mathsf{G}_3}[d] \tag{4}$$

As before, we introduce an intermediate game G_2 and prove $\models \mathsf{ES}_0 \simeq_d^{\mathcal{G}_{\mathcal{A}}} \mathsf{G}_2$, and $\models \mathsf{G}_2 \simeq_d^{\mathcal{G}_{\mathcal{A}}} \mathsf{G}_3$. By [R-Trans] we get $\models \mathsf{ES}_0 \simeq_d^{\mathcal{G}_{\mathcal{A}}} \mathsf{G}_3$ which together with ($=_{[\![]\!]}$) gives (4). The transition from ES_0 to G_2 is similar to the one detailed above. However, the transition from G_2 to G_3 is more interesting since we use an algebraic property of $\oplus$ which we call *optimistic sampling*:

$$\models x \xleftarrow{\$} \{0,1\}^\ell; y \leftarrow x \oplus z \simeq_{\{x,y,z\}}^{\{z\}} y \xleftarrow{\$} \{0,1\}^\ell; x \leftarrow y \oplus z \tag{5}$$

Let us give a step-by-step trace of the interaction with CertiCrypt:

$$\boxed{\begin{array}{l} \boldsymbol{k} \xleftarrow{\$} K;\ x, y \xleftarrow{\$} \mathbb{Z}_q; \\ (m_0, m_1) \leftarrow \mathcal{A}(g^x);\ b \xleftarrow{\$} \{0,1\}; \\ h \xleftarrow{\$} \{0,1\}^\ell;\ v \leftarrow h \oplus m_b; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, v); \\ d \leftarrow b = b' \end{array}} \quad \simeq_d^{\mathcal{G}_{\mathcal{A}}} \quad \boxed{\begin{array}{l} \boldsymbol{k} \xleftarrow{\$} K;\ x, y \xleftarrow{\$} \mathbb{Z}_q; \\ (m_0, m_1) \leftarrow \mathcal{A}(g^x);\ b \xleftarrow{\$} \{0,1\}; \\ v \xleftarrow{\$} \{0,1\}^\ell;\ h \leftarrow v \oplus m_b; \\ b' \leftarrow \mathcal{A}'(g^x, g^y, v); \\ d \leftarrow b = b' \end{array}}$$

$$\texttt{eqobs\_ctxt}\ i$$

$$\boxed{h \xleftarrow{\$} \{0,1\}^\ell;\ v \leftarrow h \oplus m_b} \simeq_{\{k,x,y,b,v\} \cup \mathcal{G}_{\mathcal{A}}}^{\{k,x,y,m_0,m_1,b\} \cup \mathcal{G}_{\mathcal{A}}} \boxed{v \xleftarrow{\$} \{0,1\}^\ell;\ h \leftarrow v \oplus m_b}$$

$$\texttt{clean\_nm}$$

$$\boxed{h \xleftarrow{\$} \{0,1\}^\ell; v \leftarrow h \oplus m_b} \simeq_{\{v\}}^{\{k,x,y,m_0,m_1,b\} \cup \mathcal{G}_{\mathcal{A}}} \boxed{v \xleftarrow{\$} \{0,1\}^\ell;\ h \leftarrow v \oplus m_b}$$

$$\texttt{apply equiv\_sub}$$

$$\boxed{h \xleftarrow{\$} \{0,1\}^\ell;\ v \leftarrow h \oplus m_b} \simeq_{\{h,v,m_b\}}^{\{m_b\}} \boxed{v \xleftarrow{\$} \{0,1\}^\ell;\ h \leftarrow v \oplus m_b}$$

$$\texttt{apply opt\_sampling}$$

First, `eqobs_ctxt` is used to remove the common prefix and suffix in the programs. Then, `clean_nm` removes from the output set the variables appearing in the input set that are not modified throughout the programs. This is justified by the rule:

$$\frac{X \cap \mathsf{modifies}(c_1) = \emptyset \quad X \cap \mathsf{modifies}(c_2) = \emptyset \qquad X \subseteq I \quad \models c_1 \simeq_O^I c_2}{\models c_1 \simeq_{O \cup X}^I c_2}$$

Finally, we apply the subsumption rule

$$\frac{I' \subseteq I \qquad \models c_1 \simeq_O^I c_2 \quad O \subseteq O'}{\models c_1 \simeq_{O'}^{I'} c_2}\ [\text{R-Sub}]$$

to strengthen the postcondition and weaken the precondition so as to obtain a goal that suits the statement of optimistic sampling (5), which allows us to conclude the proof.

The last transition effectively removed the dependency of v on b, and thus the dependency of b' on b. It is then easy to prove that

$$\Pr_{\mathsf{G}_3}[d] = \frac{1}{2} \tag{6}$$

Indeed, in G_3 the variable b is only used to compute h, which is not used anymore. We can use tactics `swap` and `deadcode` to prove that the game is equivalent to a game where b is sampled after calling $\mathcal{A}'$. Since $\forall G\ e\ d,\ \Pr_{G;d\leftarrow e}[d] = \Pr_G[e]$, we have $\Pr_{\mathsf{G}_3}[d] = \Pr_{\mathsf{G}_3}[b = b']$. To conclude, we use the fact that for any game G and Boolean variables b, b', $\Pr_{G;b \overset{\$}{\leftarrow} \{0,1\}}[b = b'] = \frac{1}{2}$, provided G is absolutely terminating. $\mathsf{CertiCrypt}$ provides a semi-decision procedure for absolute termination which can automatically discharge this condition for G_3 based on the assumption that $\mathcal{A}$ and $\mathcal{A}'$ are PPT procedures and thus absolutely terminating.

To summarize, from Equations (2), (3), (4), and (6) we obtain

$$\begin{array}{rcl}
|\Pr_{\mathsf{IND\text{-}CPA}}[d] - \frac{1}{2}| & = & |\Pr_{\mathsf{DDH}_0}[d] - \frac{1}{2}| \\
& = & |\Pr_{\mathsf{DDH}_0}[d] - \Pr_{\mathsf{DDH}_1}[d] + \Pr_{\mathsf{DDH}_1}[d] - \frac{1}{2}| \\
& \le & |\Pr_{\mathsf{DDH}_0}[d] - \Pr_{\mathsf{DDH}_1}[d]| + |\Pr_{\mathsf{DDH}_1}[d] - \frac{1}{2}| \\
& = & |\Pr_{\mathsf{DDH}_0}[d] - \Pr_{\mathsf{DDH}_1}[d]| + |\Pr_{\mathsf{ES}_1}[d] - \frac{1}{2}| \\
& = & |\Pr_{\mathsf{DDH}_0}[d] - \Pr_{\mathsf{DDH}_1}[d]| + |\Pr_{\mathsf{ES}_1}[d] - \Pr_{\mathsf{G}_3}[d]| \\
& = & |\Pr_{\mathsf{DDH}_0}[d] - \Pr_{\mathsf{DDH}_1}[d]| + |\Pr_{\mathsf{ES}_1}[d] - \Pr_{\mathsf{ES}_0}[d]| \\
& = & \epsilon_{\mathsf{DDH}}(\eta) + \epsilon_{\mathsf{ES}}(\eta)
\end{array}$$

From the above equation, $\Pr_{\mathsf{IND\text{-}CPA}}[d]$ is negligibly close to $\frac{1}{2}$ under the DDH and ES assumptions. It suffices to check that the adversaries $\mathcal{B}$ and $\mathcal{D}$ used in the reduction are PPT procedures. This is indeed the case because we assumed that both $\mathcal{A}$ and $\mathcal{A}'$ are PPT procedures. In $\mathsf{CertiCrypt}$, the tactic `PPT_proc` proves this automatically.

4.2 Security in the Random Oracle Model

Hashed ElGamal encryption is semantically secure in the random oracle model under the Computational Diffie-Hellman (CDH) assumption on the underlying group family $(G_\eta)_{\eta\in\mathbb{N}}$. 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 $\mathbb{Z}_q$. If the DDH assumption holds for the group family, then it is computationally unfeasible to test the success of an adversary against CDH (knowing only g^x and g^y). For this reason, we consider the following slightly different formulation that is equivalent in an asymptotic setting.

Definition 3 (List CDH assumption). *Consider the game*

```
Game LCDH :
  $x, y \overset{\$}{\leftarrow} \mathbb{Z}_q$;
  $L \leftarrow \mathcal{C}(g^x, g^y)$
```

and define

$$\epsilon_{\mathsf{LCDH}}(\eta) \overset{def}{=} \Pr_{\mathsf{LCDH}}[g^{xy} \in L]$$

Then, for every PPT *adversary* $\mathcal{C}$, ϵ_{LCDH} *is a negligible function.*

The DDH assumption implies the CDH assumption which in turns is equivalent to the list CDH assumption. To see this, note that an adversary against list CDH whit a non-negligible advantage can be converted into an adversary against CDH by returning a random element in the result list L; since L is necessarily of polynomial size, the list CDH advantage of the resulting adversary is still non-negligible.

Theorem 2 (Security of Hashed ElGamal in the ROM). *For every* PPT *and well-formed adversary* $(\mathcal{A}, \mathcal{A}')$,

$$\left| \Pr_{\mathsf{IND\text{-}CPA}}[d] - \frac{1}{2} \right| \leq \epsilon_{\mathsf{LCDH}}(\eta)$$

Furthermore, under the CDH *assumption,* $\Pr_{\mathsf{IND\text{-}CPA}}[d]$ *is negligibly close to* $\frac{1}{2}$.

What allows us to achieve semantic security under a (possibly) weaker assumption on the group family is a stronger assumption about the underlying family of hash functions. In the random oracle model, we model hash functions as truly random functions represented as stateful procedures. Queries are answered consistently: if some value is queried twice, the same response is given. In this model, there is no reason to continue viewing hash functions as keyed, so in the following we drop hash keys in the formalization.

The proof is sketched in Figure 3. The figure shows the sequence of games used to relate the success of the IND-CPA adversary in the original attack game to the success of the list CDH adversary $\mathcal{C}$ in game LCDH; the definition of the hash oracle is shown alongside each game. As in the proof in the standard model, we begin by inlining the calls to KG and Enc in the IND-CPA game to obtain an observationally equivalent game G_1 such that

$$\Pr_{\mathsf{IND\text{-}CPA}}[b = b'] = \Pr_{\mathsf{G}_1}[b = b'] \tag{7}$$

Then, we perform a nonlocal program transformation: at the beginning of the game we sample the value $\boldsymbol{h}^{+}$ that the hash oracle gives in response to g^{xy}. This is an instance of *lazy sampling*, a technique automated in CertiCrypt that is described in greater detail in [1]. We get

$$\Pr_{\mathsf{G}_1}[b = b'] = \Pr_{\mathsf{G}_2}[b = b'] \tag{8}$$

We can then modify the hash oracle so to not store in $\boldsymbol{L}$ the response given to a g^{xy} query; this will later let us remove $\boldsymbol{h}^{+}$ altogether from the hash oracle. To do this, define the following relational invariant

$$\begin{aligned} \phi_{23} \overset{\text{def}}{=} & \ (\boldsymbol{\Lambda} \in \mathsf{dom}(\boldsymbol{L}) \Longrightarrow \boldsymbol{L}[\boldsymbol{\Lambda}] = \boldsymbol{h}^{+})\langle 1 \rangle \ \wedge \\ & \ \forall \lambda, \lambda \neq \boldsymbol{\Lambda}\langle 1 \rangle \Longrightarrow \boldsymbol{L}[\lambda]\langle 1 \rangle = \boldsymbol{L}[\lambda]\langle 2 \rangle \end{aligned}$$

```
**Game** IND-CPA :
  $\boldsymbol{L} \leftarrow [\,]$;
  $(x, \alpha) \leftarrow \mathsf{KG}(\,)$;
  $(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $(\beta, v) \leftarrow \mathsf{Enc}(\alpha, m_b)$;
  $b' \leftarrow \mathcal{A}'(\alpha, \beta, v)$

**Oracle** $H(\lambda)$ :
  if $\lambda \notin \mathsf{dom}(\boldsymbol{L})$ then
    $h \xleftarrow{\$} \{0,1\}^\ell$;
    $\boldsymbol{L} \leftarrow (\lambda, h) :: \boldsymbol{L}$
  else $h \leftarrow \boldsymbol{L}(\lambda)$
  return $h$
```

$$\simeq^{\mathcal{G}_\mathcal{A}}_{\{b,b'\}}$$

```
**Game** $\mathsf{G}_1$ :
  $\boldsymbol{L} \leftarrow [\,]; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \leftarrow H(g^{xy})$;
  $v \leftarrow h \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$

**Oracle** $H(\lambda)$ :
  if $\lambda \notin \mathsf{dom}(\boldsymbol{L})$ then
    $h \xleftarrow{\$} \{0,1\}^\ell$;
    $\boldsymbol{L} \leftarrow (\lambda, h) :: \boldsymbol{L}$
  else $h \leftarrow \boldsymbol{L}(\lambda)$
  return $h$
```

$$\simeq^{\mathcal{G}_\mathcal{A}}_{\{b,b'\}}$$

```
**Game** $\mathsf{G}_2$ :
  $\boldsymbol{h}^+ \xleftarrow{\$} \{0,1\}^\ell$;
  $\boldsymbol{L} \leftarrow [\,]; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $\boldsymbol{\Lambda} \leftarrow g^{xy}$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \leftarrow H(\boldsymbol{\Lambda})$;
  $v \leftarrow h \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$

**Oracle** $H_2(\lambda)$ :
  if $\lambda \notin \mathsf{dom}(\boldsymbol{L})$ then
    if $\lambda = \boldsymbol{\Lambda}$ then
      $h \leftarrow \boldsymbol{h}^+$;
    else $h \xleftarrow{\$} \{0,1\}^\ell$
    $\boldsymbol{L} \leftarrow (\lambda, h) :: \boldsymbol{L}$
  else $h \leftarrow \boldsymbol{L}(\lambda)$
  return $h$
```

$$\sim^{\mathcal{G}_\mathcal{A}}_{\{b,b'\} \wedge \phi_{23}}$$

```
**Game** $\mathsf{G}_3$ :
  $\boldsymbol{h}^+ \xleftarrow{\$} \{0,1\}^\ell$;
  $\boldsymbol{L} \leftarrow [\,]; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $\boldsymbol{\Lambda} \leftarrow g^{xy}$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $h \leftarrow \boldsymbol{h}^+$;
  $v \leftarrow h \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$

**Oracle** $H_3(\lambda)$ :
  if $\lambda = \boldsymbol{\Lambda}$ then
    $h \leftarrow \boldsymbol{h}^+$
  else
    if $\lambda \notin \mathsf{dom}(\boldsymbol{L})$ then
      $h \xleftarrow{\$} \{0,1\}^\ell$
      $\boldsymbol{L} \leftarrow (\lambda, h) :: \boldsymbol{L}$
    else $h \leftarrow \boldsymbol{L}(\lambda)$
  return $h$
```

```
**Game** $\boxed{\mathsf{G}_4}$ $\boxed{\mathsf{G}_5}$ :
  **bad** $\leftarrow$ false;
  $\boldsymbol{h}^+ \xleftarrow{\$} \{0,1\}^\ell$;
  $\boldsymbol{L} \leftarrow [\,]; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $\boldsymbol{\Lambda} \leftarrow g^{xy}$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $v \leftarrow \boldsymbol{h}^+ \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$

**Oracle** $H_{4,5}(\lambda)$ :
  if $\lambda \notin \mathsf{dom}(\boldsymbol{L})$ then
    if $\lambda = \boldsymbol{\Lambda}$ then
      **bad** $\leftarrow$ true;
      $\boxed{h \leftarrow \boldsymbol{h}^+}$
      $\boxed{h \xleftarrow{\$} \{0,1\}^\ell}$
    else $h \xleftarrow{\$} \{0,1\}^\ell$
    $\boldsymbol{L} \leftarrow (\lambda, h) :: \boldsymbol{L}$
  else $h \leftarrow \boldsymbol{L}(\lambda)$
  return $h$
```

$$\sim^{\mathcal{G}_\mathcal{A}}_{\{L, \Lambda, b, b'\} \wedge (\mathsf{bad} \Longrightarrow \Lambda \in \mathsf{dom}(L)) \langle 1 \rangle}$$

```
**Game** $\mathsf{G}_6$ :
  $\boldsymbol{L} \leftarrow [\,]; x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $\boldsymbol{\Lambda} \leftarrow g^{xy}$;
  $(m_0, m_1) \leftarrow \mathcal{A}(g^x)$;
  $b \xleftarrow{\$} \{0,1\}$;
  $v \xleftarrow{\$} \{0,1\}^\ell$;
  $\boldsymbol{h}^+ \leftarrow v \oplus m_b$;
  $b' \leftarrow \mathcal{A}'(g^x, g^y, v)$

**Oracle** $H(\lambda)$ :
  if $\lambda \notin \mathsf{dom}(\boldsymbol{L})$ then
    $h \xleftarrow{\$} \{0,1\}^\ell$;
    $\boldsymbol{L} \leftarrow (\lambda, h) :: \boldsymbol{L}$
  else $h \leftarrow \boldsymbol{L}(\lambda)$
  return $h$
```

$$\simeq^{\mathcal{G}_\mathcal{A}}_{\{L,x,y\}}$$

```
**Game** LCDH :
  $x, y \xleftarrow{\$} \mathbb{Z}_q$;
  $L' \leftarrow \mathcal{C}(g^x, g^y)$
**Adversary** $\mathcal{C}(\alpha, \beta)$ :
  $\boldsymbol{L} \leftarrow [\,]$;
  $(m_0, m_1) \leftarrow \mathcal{A}(\alpha)$;
  $v \xleftarrow{\$} \{0,1\}^\ell$;
  $b' \leftarrow \mathcal{A}'(\alpha, \beta, v)$
  return $\mathsf{dom}(\boldsymbol{L})$

**Oracle** $H(\lambda)$ :
  if $\lambda \notin \mathsf{dom}(\boldsymbol{L})$ then
    $h \xleftarrow{\$} \{0,1\}^\ell$;
    $\boldsymbol{L} \leftarrow (\lambda, h) :: \boldsymbol{L}$
  else $h \leftarrow \boldsymbol{L}(\lambda)$
  return $h$
```

Fig. 3. Game-based proof of semantic security of Hashed ElGamal encryption in the Random Oracle Model

where by $e\langle 1\rangle$ (resp., $e\langle 2\rangle$) we mean the value that expression e takes in the left hand side (resp., right hand side) program. It is easy to prove that oracles H_2 and H_3 are semantically equivalent under this invariant and preserve it. Since ϕ_{23} is established just before calling $\mathcal{A}$ and is preserved throughout the games, we can prove $\models \mathsf{G}_2 \sim \mathsf{G}_3 : \mathcal{G}_{\mathcal{A}} \Rightarrow \{b, b'\} \wedge \phi_{23}$ by inlining the call to H in game G_2, hence

$$\Pr_{\mathsf{G}_2}[b = b'] = \Pr_{\mathsf{G}_3}[b = b'] \tag{9}$$

Then, we *undo* the modification to the hash oracle to prove that games G_3 and G_4 are observationally equivalent, i.e. $\models \mathsf{G}_3 \simeq^{\mathcal{G}_{\mathcal{A}}}_{\{b,b'\}} \mathsf{G}_4$, from which we obtain

$$\Pr_{\mathsf{G}_3}[b = b'] = \Pr_{\mathsf{G}_4}[b = b'] \tag{10}$$

Games G_4 and G_5 are syntactically equal to up to the point where the flag **bad** is raised. From the Fundamental Lemma described in Sec. 2, it follows that

$$|\Pr_{\mathsf{G}_4}[b = b'] - \Pr_{\mathsf{G}_5}[b = b']| \leq \Pr_{\mathsf{G}_5}[\textbf{bad}] \tag{11}$$

We then prove

$$\models \mathsf{G}_5 \sim \mathsf{G}_6 : =_{\mathcal{G}_{\mathcal{A}}} \Rightarrow =_{\{L,\Lambda,b,b'\}} \wedge\ (\textbf{bad} \Longrightarrow \boldsymbol{\Lambda} \in \mathsf{dom}(\boldsymbol{L}))\langle 1\rangle$$

Using `ep` we coalesce the branches in the innermost conditional statement of H_5 to recover the original hash oracle. We defer the sampling of $\boldsymbol{h}^+$ in G_5 to the point just before computing v using `swap`, and we substitute

$$v \xleftarrow{\$} \{0,1\}^\ell;\ \boldsymbol{h}^+ \leftarrow v \oplus m_b \qquad \text{for} \qquad \boldsymbol{h}^+ \xleftarrow{\$} \{0,1\}^\ell;\ v \leftarrow \boldsymbol{h}^+ \oplus m_b$$

using the equivalence (5) presented in Sec. 4.1. Thus,

$$\Pr_{\mathsf{G}_5}[b = b'] = \Pr_{\mathsf{G}_6}[b = b'] \tag{12}$$

and by $(\leq_{[\![]\!]})$,

$$\Pr_{\mathsf{G}_5}[\textbf{bad}] \leq \Pr_{\mathsf{G}_6}[\boldsymbol{\Lambda} \in \mathsf{dom}(\boldsymbol{L})] \tag{13}$$

Observe that in G_6, b' does not depend anymore on b, so we may as well sample b at the end of the game, thus obtaining

$$\Pr_{\mathsf{G}_6}[b = b'] = \frac{1}{2} \tag{14}$$

We construct an adversary $\mathcal{C}$ against list CDH that interacts with the adversary $(\mathcal{A}, \mathcal{A}')$ playing the role of an $\mathsf{IND\text{-}CPA}$ challenger. It returns the list of queries that the adversary $(\mathcal{A}, \mathcal{A}')$ makes to the hash oracle. Observe that $\mathcal{C}$ does not need to know x or y because it gets g^x and g^y as parameters. The success probability of $\mathcal{C}$ is the same as the probability of $\boldsymbol{\Lambda} = g^{xy}$ being in the domain of $\boldsymbol{L}$ in G_6. Therefore, we finally have that

$$\Pr_{\mathsf{G}_6}[\boldsymbol{\Lambda} \in \mathsf{dom}(\boldsymbol{L})] = \Pr_{\mathsf{G}_6}[g^{xy} \in \mathsf{dom}(\boldsymbol{L})] = \Pr_{\mathsf{LCDH}}[g^{xy} \in L'] \tag{15}$$

To summarize, from Equations (7)—(15) we obtain

$$\begin{aligned}
|\Pr_{\mathsf{IND\text{-}CPA}}[b = b'] - \tfrac{1}{2}| &= |\Pr_{\mathsf{G}_4}[b = b'] - \tfrac{1}{2}| \\
&= |\Pr_{\mathsf{G}_4}[b = b'] - \Pr_{\mathsf{G}_6}[b = b']| \\
&= |\Pr_{\mathsf{G}_4}[b = b'] - \Pr_{\mathsf{G}_5}[b = b']| \\
&\leq \Pr_{\mathsf{G}_5}[\mathbf{bad}] \\
&\leq \Pr_{\mathsf{G}_6}[\boldsymbol{\Lambda} \in \mathsf{dom}(\boldsymbol{L})] \\
&= \Pr_{\mathsf{LCDH}}[g^{xy} \in L'] \\
&= \epsilon_{\mathsf{LCDH}}(\eta)
\end{aligned}$$

From the above equation and under the list CDH assumption (or equivalently, under the plain CDH assumption), the IND-CPA advantage of adversary $(\mathcal{A}, \mathcal{A}')$ results negligibly close to $\frac{1}{2}$. To see this, it suffices to verify that adversary $\mathcal{C}$ runs in probabilistic polynomial time. This is the case because adversary $(\mathcal{A}, \mathcal{A}')$ does, and $\mathcal{C}$ does not perform any costly computations.

5 Related work

ElGamal is a standard example of a game-based cryptographic proof that provides a benchmark against which other works can be compared. We briefly comment on three proofs that are closely related to ours. For a more general account of related work, we refer to [1].

The most recent, and closely related is a formalization in Coq of a game-based proof of ElGamal semantic security by Nowak [12]. While we opt for a deep embedding, Nowak uses a shallow embedding and models adversaries directly as Coq functions. As a consequence, the resulting framework only provides limited support for proof automation. For the same reason, Nowak's formalization cannot deal with random oracles, so that he only presents the proof of Hashed ElGamal in the standard model of cryptography. Finally, it is not clear how to formalize complexity in the context of a shallow embedding, and Nowak's formalization ignores complexity altogether; as a result, security assumptions such as DDH cannot be modelled faithfully.

An earlier work by Barthe, Cederquist and Tarento [13] provides the foundations of a formal proof of security of Signed ElGamal encryption in Coq. In contrast to our work, they consider an idealized model of cryptography that abstracts away many details of the system and the security definition. Thus, the connection between the formalization and the security statement is not as strong as desired.

Corin and den Hartog [14] developed a (non-relational) Hoare logic for reasoning about probabilistic algorithms. They used it to construct a proof of semantic security of ElGamal encryption, but we are not aware of any other system verified using this logic. Being based on a mere probabilistic extension of Hoare logic, their formalism is not sufficiently expressive to model the notion of PPT complexity, and so security goals and hypotheses cannot be expressed precisely. More generally, the logic by itself provides no means to reason about context-dependent program transformations or transformations made in oracles.

6 Conclusion

CertiCrypt is a fully formalized framework that assists the construction of cryptographic game-based proofs. Proofs in CertiCrypt rely on a minimal trusted base and their correctness can be verified automatically by third parties. In this paper, we have illustrated some key aspects of CertiCrypt through the formalization of semantic security proofs of the Hashed ElGamal public-key encryption scheme in the standard and random oracle model, and we have highlighted some essential differences between our proofs and those that appear in the literature.

Acknowledgments We would like to thank Daniel Hedin for his helpful comments on an earlier draft of this work.

References

1. Barthe, G., Grégoire, B., Zanella Béguelin, S.: Formal certification of code-based cryptographic proofs. In: Proceedings of the 36th ACM Symposium on Principles of Programming Languages, ACM Press (2009)
2. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2) (1984) 270–299
3. Stern, J.: Why provable security matters? In: Advances in Cryptology – EUROCRYPT'03. Volume 2656 of Lecture Notes in Computer Science., Springer-Verlag (2003)
4. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Advances in Cryptology – EUROCRYPT'06. Volume 4004 of Lecture Notes in Computer Science. (2006) 409–426
5. Halevi, S.: A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181 (2005)
6. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004)
7. The Coq development team: The Coq Proof Assistant Reference Manual v8.2 (2008) Available at `http://coq.inria.fr`.
8. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher-Order and Symbolic Computation **14**(1) (2001) 59–91
9. Audebaud, P., Paulin-Mohring, C.: Proofs of randomized algorithms in Coq. Science of Computer Programming (2008)
10. Benton, N.: Simple relational correctness proofs for static analyses and program transformations. In: Proceedings of the 31th ACM Symposium on Principles of Programming Languages, ACM Press (2004) 14–25
11. Jonsson, B., Larsen, K.G., Yi, W.: Probabilistic extensions of process algebras. In: Handbook of Process Algebra. Elsevier (2001) 685–711
12. Nowak, D.: A framework for game-based security proofs. In: Information and Communications Security. Volume 4861., Springer-Verlag (2007) 319–333
13. Barthe, G., Cederquist, J., Tarento, S.: A machine-checked formalization of the generic model and the random oracle model. In: 2nd International Joint Conference on Automated Reasoning, Springer-Verlag (2004) 385–399
14. Corin, R., den Hartog, J.: A probabilistic Hoare-style logic for game-based cryptographic proofs. In: Proceedings of the 33rd International Colloquium on Automata, Languages and Programming. Volume 4052 of LNCS. (2006) 252–263