A Framework for Game-Based Security Proofs [⋆]

David Nowak

Research Center for Information Security, AIST, Tokyo

November 7, 2007[⋆⋆]

Abstract. To be accepted, a cryptographic scheme must come with a proof that it satisfies some standard security properties. However, because cryptographic schemes are based on non-trivial mathematics, proofs are error-prone and difficult to check. The main contributions of this paper are a refinement of the game-based approach to security proofs, and its implementation on top of the proof assistant Coq. The proof assistant checks that the proof is correct and deals with the mundane part of the proof. An interesting feature of our framework is that our proofs are formal enough to be mechanically checked, but still readable enough to be humanly checked. We illustrate the use of our framework by proving in a systematic way the so-called semantic security of the encryption scheme Elgamal and its hashed version.

Keywords: formal verification, game, proof assistant, security

1 Introduction

Information security is nowadays an important issue. Its essential ingredient is cryptography. To be accepted, a cryptographic scheme must come with a proof that it satisfies some standard security properties. However, because cryptographic schemes are based on non-trivial mathematics such as number theory, group theory or probability theory, this makes the proofs error-prone and difficult to check. Bellare and Rogaway even claim that

> "many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor." [5]

In particular, proofs often rely on assumptions that are not clearly stated. This is why they advocate the usage of sequences of games (a.k.a. game-playing technique or game-hopping technique).

This methodology is explicitly presented in [5] and [20] but has been used in various styles before in the literature. It is a way to structure proofs so as to make them less error-prone, more easily verifiable, and, ideally, machine-checkable. A proof starts with the initial game which comes from the definition of the security property to be proved. This can be seen as a challenge involving the attacker and oracles. Attacker and oracles are efficient probabilistic algorithms (usually modeled as probabilistic polynomial-time algorithms). Oracles model services provided by the environment. For example an oracle might provide signed messages in order to model the spying of signed messages circulating on a network. A testing oracle checks whether an attack is successful of not. There are also encryption and decryption oracles. From the initial game, one builds a sequence of games such that the last one is simple enough to reason on directly. The result is then backtracked to the initial game. This is possible because transformations result either in an equivalent game or introduce small enough and quantified changes.

Our contributions. Recently, Halevi [15] has advocated the need for a software which can deal with the mundane part of writing and checking game-based proofs. In order to aim at such goal, we present a refinement of the game-based approach to security proofs, and its implementation[1] on top of the proof

[⋆] A short version of this paper will appear in the proceedings of the 9th International Conference on Information and Communications Security (ICICS 2007) [18].

[⋆⋆] First public version: May 28, 2007

[1] A link to the source code is provided on Cryptology ePrint Archive together with this paper.

assistant Coq[2]. A proof assistant can indeed check that a proof is correct and deal with its mundane part. Of course, human interaction is still needed in order to deal with the creative part of the proof. But, when using a proof assistant, two things are necessary. First, all the intermediate lemmas must be explicited; some of those lemmas are not stated by cryptographers in their proofs because they are considered too obvious in the context of security proofs. Second, a precise mathematical meaning must be given to games; in papers, this is usually either left implicit or informally explained in English. This is why we need to refine the game-based approach. We base our formalization on [20] where games are seen as probability distributions. Our aim is to have a framework in which proofs are formal enough to be mechanically checked, and readable enough to be humanly checked.

The approach to game-based proofs by Shoup [20] differs from the one by Bellare and Rogaway [5]: In the latter, games are seen as syntactic objects. An interest in founding our formalization on this latter approach would be the possibility for more automation because game transformations would be syntactic. But each syntactic transformation should then be proved correct with respect to a precise semantics in terms of probability distributions. However in [5] the semantics is left implicit. They provide arguments for their syntactic transformations, but they cannot be directly formalized in a proof assistant due to the lack of semantics.

We illustrate the use of our framework by proving in a systematic way the so-called semantic security of the encryption scheme ElGamal and its hashed version [12]. It is a widely-used asymmetric key encryption algorithm. It is notably used by GNU Privacy Guard software, recent versions of PGP and other cryptographic software. Under the so-called Decisional Diffie-Hellman (DDH) assumption [10], it can be proved *semantically secure* [21]. To the best of our knowledge, this is the first time a cryptographic scheme is fully machine-checked. This is not the case in related work (see Section 2).

Outline. We start with related work in Section 2. In Section 3, we introduce our mathematical framework. In Section 4, we formalize some security notions. In Section 5, we show how to prove semantic security for the encryption scheme ElGamal and its hashed version. Implementation issues in Coq are addressed in Section 6. Finally, we conclude and give our plan for future work in Section 7.

2 Related work

A lot of work has been done in direction of automatic discovery of proofs. It is essentially based on the Dolev-Yao model [11] which requires a high-level of abstraction, and is thus far from the view usually adopted by cryptographers. In this paper, we are not considering automatic discovery of proofs, but instead we want to facilitate the writing and checking of actual proofs by cryptographers.

The so-called generic model and random oracle model have been formalized in Coq and applied to ElGamal [3]. In contrast to our approach, it is not based on sequences of games which had not yet been popularized by [5] and [20].

CryptoVerif is a software for automated security proofs with sequences of games [6]. It is in particular illustrated with a proof of the Full-Domain Hash (FDH) signature scheme [4]. However this proof relies on certain equivalences that have to be introduced by the user. Those non-trivial equivalences are proved manually in Appendix B of [7]. These are difficult parts of the proof that cannot be handled by CryptoVerif. Moreover this tool consists of 14800 lines of non-certified O'Caml codes. On the other hand, our tool is certified: all our game transformations have been proved correct in the proof assistant Coq.

A probabilistic Hoare-style logic has been proposed (but not implemented) in [9] to formalize game-based proofs. This logic allows for rigorous proofs but those proofs differ from game-based proofs by cryptographers. Indeed, because their language allows for `while` loops and state variables, they are led to use a Hoare-style logic. They illustrate their logic by proving semantic security of the non-hashed version of ElGamal. In our approach, logical reasoning is closer to the one used by cryptographers: we avoid `while` loops and state variables, and thus do not have to use a Hoare-style logic. It is possible because the variables used in [20] are mathematical variables in the sense that they are defined once and only once whereas the value of a state

[2] See `http://coq.inria.fr/`

variable can change in the course of execution. By the way, the property that a variable is defined once and only once is also enforced in CryptoVerif. Moreover, while loops, if used, would have to be restricted because their unrestricted use might break the hypothesis that the attacker and the oracles are efficient algorithms. Our games are probability distributions which are easily defined in our framework. In the case of ElGamal, we finally obtain a more natural proof of semantic security than the one in [9].

In [16] a process calculus is defined (but not implemented) which allows to reason about cryptographic protocols using bisimulation techniques. Contrary to our approach it is not game-based and differs from usual proofs by cryptographers. It is illustrated by a proof of semantic security for ElGamal.

An encoding of game-based proofs in a proof assistant has been proposed very recently in [1]. It is dedicated for proofs in the random oracle model while our work focuses on the standard model. Up to now the implementation by [1] has only been used to prove the PRP/PRF switching lemma, but not yet a full-fledged cryptographic scheme. Compared to them, we have been very careful in making our design choices such that our implementation remains light. This is an important design issue in formal verification because formal proofs grow quickly in size when one tackles real-world use-cases. For illustration, one can compare the size of our implementation with theirs: their complete implementation consists of 7032 lines of code (compare with our 3381 lines) and their proof of the switching lemma consists of 535 lines (compare with our 160 lines for proving both correctness and semantic security of ElGamal).

3 Mathematical framework

In this section we recall a few mathematical bases on which rely security proofs: probabilities, cyclic groups and properties relating them. We formulate them in a way suitable for formalization in the proof assistant Coq. In particular, we use the elegant notion of monad stemming from category theory [17] and functional programming [22][3].

3.1 Probabilities

Oracles and games are probabilistic algorithms. We model them as functions returning finite probability distributions. A probabilistic choice is a side effect. A standard way to model side effects is with a monad [17, 22]. And indeed probability distributions have a monadic structure [2, 19]. In our case we only need to consider the simpler case of finite probability distributions. In their definition we use the notion of multiset (sometimes also called a bag) which is a set where an element may have more than one occurrence. For example, the multisets $\{1, 2, 2\}$ and $\{1, 2\}$ are different; and the union of $\{1, 2, 2, 3\}$ and $\{1, 4, 4\}$ is equal to $\{1, 1, 2, 2, 3, 4, 4\}$.

Definition 3.1 (Finite probability distribution). *A finite probability distribution δ over a set A is a finite multiset of ordered pairs from $A \times \mathbb{R}$ such that $\sum_{(a,p)\in\delta} p = 1$. We write Δ_A for the set of finite probability distributions over a set A.*

From now on, we will use the word *distribution* as an abbreviation for *finite probability distribution.* Games and oracles are distributions defined by using three primitive operations: $[a]$ is the distribution consisting of only one value a with probability 1; $\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x)$ consists of selecting randomly one value x from the distribution δ and passes it to the function φ; and $\bigoplus\{a_1, \ldots, a_n\}$ is the uniform distribution of the values $a_1, \ldots, a_n$. Before giving their formal meaning in the definition below, we need to define the ponderation of a distribution by a real number p:

$$p \cdot \{(a_1, p_1), \ldots, (a_n, p_n)\} \;=_{\text{def}}\; \{(a_1, p \cdot p_1), \ldots, (a_n, p \cdot p_n)\}$$

[3] No knowledge of category theory or functional programming is assumed.

Definition 3.2 (Operations).

$$[a] =_{def} \{(a,1)\} \tag{1}$$

$$\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x) =_{def} \bigcup_{(a,p)\in\delta} p\cdot\varphi(a) \tag{2}$$

$$\bigoplus\{a_1,\dots,a_n\} =_{def} \{(a_1,\frac{1}{n}),\dots,(a_n,\frac{1}{n})\} \tag{3}$$

It is easily seen that those three operations above produce well-defined distributions.

In the rest of this paper, we use the following abbreviations:

(i) $\mathsf{let}\ x \leftarrow a\ \mathsf{in}\ \varphi(x)$ for $\mathsf{let}\ x \Leftarrow [a]\ \mathsf{in}\ \varphi(x)$, and
(ii) $\mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\ \varphi(x)$ for $\mathsf{let}\ x \Leftarrow \bigoplus A\ \mathsf{in}\ \varphi(x)$.

In (i) we choose ramdomly a value from a distribution with only one value: it is a deterministic assignment. (ii) is a notation for choosing a uniformly random value from a list of values.

It might seem surprising that our distributions are multisets instead of sets. If we were to take sets, our definition of let would be more tricky as it would involve a phase of normalization. Let us see why on an example. Consider the distribution defined by $\mathsf{let}\ x \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in}\ [x \overset{?}{=} x]$ where $\overset{?}{=}$ is the function that returns the boolean true if its two arguments are equal, or false otherwise. The above defined distribution is equal to the multiset $\{(\mathsf{true},\frac{1}{2}),(\mathsf{true},\frac{1}{2})\}$. If distributions were sets, we would have to define let in such a way that it returns what might be called the normal form $\{(\mathsf{true},1)\}$.

The following theorem states that we have indeed defined a (strong) monad.

Theorem 3.3 (Monad laws).

$$\mathsf{let}\ x \leftarrow a\ \mathsf{in}\ \varphi(x) = \varphi(a) \tag{4}$$

$$\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ [x] = \delta \tag{5}$$

$$\mathsf{let}\ y \Leftarrow (\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x))\ \mathsf{in}\psi(y) = \mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \mathsf{let}\ y \Leftarrow \varphi(x)\ \mathsf{in}\ \psi(y) \tag{6}$$

In order to ease notations we assume that the operator let ... in is right-associative: this means that, for example, the right-hand side expression of Equation (6) above should be understood as

$$\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ (\mathsf{let}\ y \Leftarrow \varphi(x)\ \mathsf{in}\ \psi(y))$$

Equation (4) allows for propagating constants. Equation (6) states associativity which allows for getting rid of nested let.

Based on our notion of distribution, we can now define the probability that an element chosen randomly from a distribution satisfies a certain predicate.

Definition 3.4 (Probability). *The probability* $\mathbf{Pr}\left(P\,(\delta)\right)$ *that an element chosen randomly in a distribution* δ *satisfies a predicate* P *is given by:*

$$\mathbf{Pr}\left(P\,(\delta)\right) =_{def} \sum_{(a,p)\in\delta\ s.t.\ P(a)} p$$

We write $\mathbf{Pr}_{\mathsf{true}}\left(\delta\right)$ for $\mathbf{Pr}\left((x \mapsto x = \mathsf{true})\,(\delta)\right)$ where $x \mapsto x = \mathsf{true}$ is the predicate that holds iff its argument x is equal to the boolean value true.

The following proposition tells us how to compute the probability for a distribution defined by a let.

Proposition 3.5. *For all* P, δ *and* φ,

$$\mathbf{Pr}\left(P\,(\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x))\right) = \sum_{(a,p)\in\delta} p\cdot\mathbf{Pr}\left(P\,(\varphi(a))\right)$$

The following corollary shows how to compute the probability of a successful equality test between a random value and a constant.

Corollary 3.6. *For any finite set A, for any $a \in A$,*

$$\mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in} \\ [x \overset{?}{=} a] \end{array} \right) = \frac{1}{|A|}$$

The following corollary allows for rewriting under a let.

Corollary 3.7. *For all sets A and B, for any distribution $\delta \in \Delta_A$, for all functions φ and ψ from A to Δ_B, if* $\forall a \in A \cdot \mathbf{Pr}\left(P\left(\varphi(a)\right)\right) = \mathbf{Pr}\left(P\left(\psi(a)\right)\right)$ *then* $\mathbf{Pr}\left(P\left(\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x)\right)\right) = \mathbf{Pr}\left(P\left(\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \psi(x)\right)\right)$

As another corollary, we obtain a mean to replace a randomly uniform choice in a goal by a universal quantifier[4].

Corollary 3.8. *For all P, A, φ and p,*

$$(\forall x \in A \cdot \mathbf{Pr}\left(P\left(\varphi(x)\right)\right) = p) \Rightarrow \mathbf{Pr}\left(P\left(\mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\ \varphi(x)\right)\right) = p$$

The reverse implication is not true. We can see that on a counterexample: if the reverse implication was true, from Corollary 3.6 we would deduce that $\forall x \in A \cdot \mathbf{Pr}_{\mathsf{true}}\left([x \overset{?}{=} a]\right) = \frac{1}{|A|}$. This is not true. Here x is either equal or not to a: in case of equality the probability is 1; in case of non-equality the probability is 0. It shows us a fundamental difference between universal quantification and random choice.

The following proposition allows for moving around independent random choices in the definitions of games. In the proposition below, *independent* means that the variable x is not used in the expression δ_2 and the variable y is not used in the expression δ_1.

Proposition 3.9. *For all finite sets A, B and C, for any $\delta_1 \in \Delta_A$, for any $\delta_2 \in \Delta_B$, for any $\varphi : A \times B \to \Delta_C$, if δ_1 and δ_2 are independent, then:*

$$\mathbf{Pr}\left(P\left(\begin{array}{l} \mathsf{let}\ x \Leftarrow \delta_1\ \mathsf{in} \\ \mathsf{let}\ y \Leftarrow \delta_2\ \mathsf{in} \\ \varphi(x,y) \end{array}\right)\right) = \mathbf{Pr}\left(P\left(\begin{array}{l} \mathsf{let}\ y \Leftarrow \delta_2\ \mathsf{in} \\ \mathsf{let}\ x \Leftarrow \delta_1\ \mathsf{in} \\ \varphi(x,y) \end{array}\right)\right)$$

Additionally we define a necessity modality stating that a certain predicate is satisfied by all those elements of a distribution that have a probability strictly greater than 0.

Definition 3.10 (Necessity). $\Box P\ (\delta)$ *states that a predicate P holds necessarily for a distribution δ:* $\Box P\ (\delta) \ \Leftrightarrow_{def}\ \forall (a,p) \in \delta \cdot p > 0 \Rightarrow P(a)$

If P is a predicate on a set A, then $\Box P$ is a predicate on Δ_A. Because distributions are finite, $\Box P\ (\delta)$ is equivalent to $\mathbf{Pr}\left(P\left(\delta\right)\right) = 1$.

The following proposition, when applied recursively, will remove the necessity modality from the goal.

Proposition 3.11.

$$P(a) \Rightarrow \Box P\ ([a]) \tag{7}$$

$$\forall a \in A \cdot P(a) \Rightarrow \Box P\ (\bigoplus A) \tag{8}$$

$$\Box(x \mapsto \Box P\ (\varphi(x)))\ (\delta) \Rightarrow \Box P\ (\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x)) \tag{9}$$

[4] We assume here a backward reasoning as in the proof assistant Coq where we start from the goal and go backward to the hypothesis. For example, if our goal is Q and we have a theorem stating that $P \Rightarrow Q$, applying this theorem leaves us with P as a new goal.

3.2 Cyclic groups

A group $(G, *)$ consists in a set G with an associative operation $*$ satisfying certain axioms. We write a^{-1} for the inverse of a. We write a^i for $\underbrace{a * \cdots * a}_{i \text{ times}}$. A group $(G, *)$ is finite if the set G is finite. In a finite group G, the number of elements is called the order of G. A group is cyclic if there is an element $\gamma \in G$ such that for each $a \in G$ there is an integer i with $a = \gamma^i$. Such γ is called a generator of G. The following permutation properties of cyclic groups will allow us below to connect probabilities with cyclic groups. Let G be a finite cyclic group.

Proposition 3.12. *If the order of G is q, then* $\{\gamma^i \mid 0 \leq i < q\} \ = \ G$

Proposition 3.13. *For any* $b \in G$, $\{a * b \mid a \in G\} \ = \ G$

The set of bit strings of length l equipped the the bitwise exclusive disjunction $\oplus$ forms a commutative group (not cyclic) where the following proposition holds:

Proposition 3.14. *For any* $s' \in \{0,1\}^l$, $\left\{s \oplus s' \mid s \in \{0,1\}^l\right\} \ = \ \{0,1\}^l$

3.3 Probabilities over cyclic groups

The following theorem and its corollaries make explicit a fundamental relation between probabilities and cyclic groups. They are important properties used implicitly by cryptographers but never explicitly stated because they are considered too obvious in the context of security proofs. However it is necessary to explicit them when using a proof assistant.

Let G be a finite cyclic group of order q and $\gamma \in G$ be a generator. We write $\mathbb{Z}_q$ for the set of integers $\{0, \ldots, q-1\}$.

Theorem 3.15. *for all sets A, B and C, for any bijective function* $f : A \to B$, *for any function* $g : B \to C$, *for any predicate P on C,*

$$\mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\\ {[g(f(x))]}\end{array}\right)\right] \ = \ \mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ y \overset{R}{\leftarrow} B\ \mathsf{in}\\ {[g(y)]}\end{array}\right)\right]$$

Corollary 3.16. *for any set A, for any function f from G to A, for any predicate P on A,*

$$\mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in}\\ {[f(\gamma^x)]}\end{array}\right)\right] \ = \ \mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ m \overset{R}{\leftarrow} G\ \mathsf{in}\\ {[f(m)]}\end{array}\right)\right]$$

Proof. By Proposition 3.12 and Theorem 3.15. □

Corollary 3.17. *for any set A, for any function f from G to A, for any predicate P on A, for any* $m' \in G$,

$$\mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ m \overset{R}{\leftarrow} G\ \mathsf{in}\\ {[f(m * m')]}\end{array}\right)\right] \ = \ \mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ m \overset{R}{\leftarrow} G\ \mathsf{in}\\ {[f(m)]}\end{array}\right)\right]$$

Proof. By Proposition 3.13 and Theorem 3.15. □

Corollary 3.18. *for any set A, for any function f from* $\{0,1\}^l$ *to A, for any predicate P on A, for any* $s' \in \{0,1\}^l$,

$$\mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ s \overset{R}{\leftarrow} \{0,1\}^l\ \mathsf{in}\\ {[f(s \oplus s')]}\end{array}\right)\right] \ = \ \mathbf{Pr}\left[P\left(\begin{array}{l}\mathsf{let}\ s \overset{R}{\leftarrow} \{0,1\}^l\ \mathsf{in}\\ {[f(s)]}\end{array}\right)\right]$$

Proof. By Proposition 3.14 and Theorem 3.15. □

In Section 3.3 of [20] the proof of semantic security for the encryption scheme ElGamal uses implicitly such corollaries. Shoup writes: "*by independence, the conditional distribution of δ is the uniform distribution on G, and hence from this, one sees that the conditional distribution of $\zeta = \delta \cdot m_b$ is the uniform distribution on G*". The "*by independence*" part corresponds to our corollary 3.16, while the "*one sees that*" part corresponds to our corollary 3.17. It is perfectly legitimate not to state precisely things that are anyway obvious to the reader. But for our implementation on top of the proof assistant Coq it was necessary to state such theorems explicitly and formally.

4 Formal security

In this section we formalize in our framework some security notions which are fundamental in cryptography: the Decisional Diffie-Hellman assumption (DDH), entropy smoothing and semantic security. We also formalize what it means for an encryption scheme to be correct.

4.1 The Decisional Diffie-Hellman assumption

Let G be a finite cyclic group of order q and $\gamma \in G$ be a generator[5].

The DDH assumption [10] for G states that, roughly speaking, no efficient algorithm can distinguish between triples of the form $(\gamma^x, \gamma^y, \gamma^{xy})$ and $(\gamma^x, \gamma^y, \gamma^z)$ where x, y and z are chosen randomly in the set $\mathbb{Z}_q$. More formally, there exists a negligible upper-bound ϵ_{DDH} such that for any efficient algorithm φ from $G \times G \times G$ to $\varDelta_{\{\mathsf{false},\,\mathsf{true}\}}$:

$$\left| \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \varphi(\gamma^x, \gamma^y, \gamma^{xy}) \end{array} \right) - \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ z \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \varphi(\gamma^x, \gamma^y, \gamma^z) \end{array} \right) \right| \leq \epsilon_{\text{DDH}}$$

As will be seen in Section 5, security proofs in our framework mainly consist in game transformations. Thus, as in [9], we do not need to define precisely the terms *efficient* and *negligible*. However they can be given precise definitions in terms of polynomials.

4.2 Entropy smoothing

A family $(H_k)_{k \in K}$, where each H_k is a hash function from G to $\{0,1\}^l$, is entropy smoothing iff there exists a negligible upper-bound ϵ_{ES} such that for any efficient algorithm φ from $K \times \{0,1\}^l$ to $\varDelta_{\{\mathsf{false},\,\mathsf{true}\}}$:

$$\left| \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ m \overset{R}{\leftarrow} G\ \mathsf{in} \\ \varphi(k, H_k(m)) \end{array} \right) - \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ h \overset{R}{\leftarrow} \{0,1\}^l\ \mathsf{in} \\ \varphi(k, h) \end{array} \right) \right| \leq \epsilon_{\text{ES}}$$

Roughly speaking, it means that no efficient algorithm can distinguish between $(k, H_k(m))$ and (k, h) where k, m and h are chosen randomly.

4.3 Semantic security

The notion of semantic security was introduced by Goldwasser and Micali [13]. They later showed that it is equivalent to indistinguishability under Chosen Plaintext Attack (IND-CPA) [14]. We use this latter formulation which is nowadays the most commonly used.

[5] We do not assume that q is prime. However most groups in which DDH is believed to be true have prime order [8].

We assume two oracles: a key generation oracle keygen which generates a pair of public and private keys; and an encryption oracle encrypt which encrypts a given plaintext with a given public key. Because oracles are probabilistic algorithms, they are modeled as functions returning distributions. The attacker is modeled as two deterministic efficient algorithms A_1 and A_2 that take among other input a random seed r taken for some non-empty set R.

The semantic security game $SSG(\mathsf{keygen}, \mathsf{encrypt}, A_1, A_2)$ consists in calling the oracle keygen, then passing the generated public key and a random seed to A_1 which returns a pair of messages m_1 and m_2. One of the messages is chosen randomly and encrypted by the oracle encrypt which returns the corresponding ciphertext. This ciphertext is passed to A_2 which tries to guess which of the two messages was encrypted. In our framework, it is defined by:

```
let $(kp, ks) \Leftarrow \mathsf{keygen}()$ in
let $r \overset{R}{\leftarrow} R$ in let $(m_1, m_2) \leftarrow A_1(r, kp)$ in
let $b \overset{R}{\leftarrow} \{1, 2\}$ in let $c \Leftarrow \mathsf{encrypt}(kp, m_b)$ in
let $\hat{b} \leftarrow A_2(r, kp, c)$ in
$[\hat{b} \overset{?}{=} b]$
```

Definition 4.1 (Semantic security). *An encryption scheme with key generation algorithm* keygen *and encryption algoritm* encrypt *is semantically secure iff for all deterministic efficient algorithms A_1 and A_2,*

$$\left| \mathbf{Pr}_{\mathsf{true}} \Big(SSG(\mathsf{keygen}, \mathsf{encrypt}, A_1, A_2) \Big) - \frac{1}{2} \right| \quad \textit{is negligible.}$$

4.4 Correctness of a cryptographic scheme

In a similar manner we define a correctness game $CG(\mathsf{keygen}, \mathsf{encrypt}, \mathsf{decrypt}, m)$ by:

```
let $(kp, ks) \Leftarrow \mathsf{keygen}()$ in
let $c \Leftarrow \mathsf{encrypt}(kp, m)$ in
let $m' \Leftarrow \mathsf{decrypt}(ks, c)$ in
$[m \overset{?}{=} m']$.
```

where decrypt is a decryption oracle which decrypts a given ciphertext with a given secret key. The purpose of this game is to show that encrypting any message $m \in G$ with a public key, and then decrypting the obtained ciphertext with the corresponding secret key gives back the same message m. Formally, an encryption scheme given as a triple $(\mathsf{keygen}, \mathsf{encrypt}, \mathsf{decrypt})$ is correct iff for any message m the correctness game necessarily returns true:

$$\forall m \quad \cdot \quad \Box(x \mapsto x = true)\ CG(\mathsf{keygen}, \mathsf{encrypt}, \mathsf{decrypt}, m)$$

5 Application to the ElGamal encryption scheme

In our implementation, we illustrate the use of our framework by proving in a systematic way the so-called semantic security of the encryption scheme ElGamal [12] and its hashed version.

5.1 The ElGamal encryption scheme

Let G be a finite cyclic group of order q and $\gamma \in G$ be a generator. The ElGamal encryption scheme consists in the following probabilistic algorithms:

– The key generation algorithm keygen():
  let $x \overset{R}{\leftarrow} \mathbb{Z}_q$ in $[(\gamma^x, x)]$

– The encryption algorithm $\mathsf{encrypt}(kp, m)$:
$\mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in}\ [(\gamma^y, kp^y * m)]$
– The decryption algorithm $\mathsf{decrypt}(ks, c)$:
$[\pi_2(c) \ * \ \left(\pi_1(c)^{ks}\right)^{-1}]$
where π_1 and π_2 denote the first and second projections of an ordered pair.

Messages and public keys are elements of G; secret keys are elements of $\mathbb{Z}_q$; ciphertexts are elements of $G \times G$.

To prove that ElGamal is a correct encryption scheme (as defined in Section 4.4), we apply backward and recursively Proposition 3.11 until we are left with the following equation in G to prove: $m = \gamma^{xy} * m * (\gamma^{yx})^{-1}$. It is obvious from the laws of a group. This backward and recursive application of Proposition 3.11 is dealt with automatically in our implementation (See Section 6).

Theorem 5.1. *The ElGamal encryption scheme is semantically secure.*

Proof. In this proof we implicitly apply Corollaries 3.7 and 3.8, and Proposition 3.9. In particular the reader will notice that the order of variable definitions varies along the game transformations as allowed by Proposition 3.9.

Let us fix A_1 and A_2. We proceed by successive game transformations.

G0. By definition of semantic security, we must prove that:

$$\left| \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ (kp, ks) \ \Leftarrow \ \mathsf{keygen}()\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in}\ \mathsf{let}\ (m_1, m_2) \ \leftarrow \ A_1(r, kp)\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in}\ \mathsf{let}\ c \ \Leftarrow \ \mathsf{encrypt}(kp, m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b} \ \leftarrow \ A_2(r, kp, c)\ \mathsf{in} \\ [\hat{b} \overset{?}{=} b] \end{array} \right) - \frac{1}{2} \right| \text{ is negligible}$$

G1. Knowing that ϵ_{DDH} is negligible, we are led to prove that:

$$\left| \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ (kp, ks) \ \Leftarrow \ \mathsf{keygen}()\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in}\ \mathsf{let}\ (m_1, m_2) \ \leftarrow \ A_1(r, kp)\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in}\ \mathsf{let}\ c \ \Leftarrow \ \mathsf{encrypt}(kp, m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b} \ \leftarrow \ A_2(r, kp, c)\ \mathsf{in} \\ [\hat{b} \overset{?}{=} b] \end{array} \right) - \frac{1}{2} \right| \leq \epsilon_{\text{DDH}}$$

G2. We unfold definitions of oracles and apply associativity of let (by Theorem 3.3 (6)).

$$\left| \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}(kp, ks) \ \leftarrow \ (\gamma^x, x)\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ (m_1, m_2) \ \leftarrow \ A_1(r, kp) \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ c \ \leftarrow \ (\gamma^y, kp^y * m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b} \ \leftarrow \ A_2(r, kp, c)\ \mathsf{in} \\ [\hat{b} \overset{?}{=} b] \end{array} \right) - \frac{1}{2} \right| \leq \epsilon_{\text{DDH}}$$

G3. We propagate definitions of kp, ks, m_1, m_2, c and $\hat{b}$ (by Theorem 3.3 (4)).

$$\left| \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ [A_2(r, \gamma^x, (\gamma^y, \gamma^{xy} * \pi_b(A_1(r, \gamma^x)))) \overset{?}{=} b] \end{array} \right) - \frac{1}{2} \right| \leq \epsilon_{\text{DDH}}$$

G4. According to DDH assumption, we have that:

$$\left| \mathbf{Pr}_{\text{true}} \begin{pmatrix} \text{let } x \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } y \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } r \overset{R}{\leftarrow} R \text{ in} \\ \text{let } b \overset{R}{\leftarrow} \{1,2\} \text{ in} \\ [A_2(r, \gamma^x, (\gamma^y, \\ \quad \gamma^{xy} * \\ \quad\quad \pi_b(A_1(r, \gamma^x)))) \overset{?}{=} b] \end{pmatrix} - \mathbf{Pr}_{\text{true}} \begin{pmatrix} \text{let } x \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } y \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } z \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } r \overset{R}{\leftarrow} R \text{ in} \\ \text{let } b \overset{R}{\leftarrow} \{1,2\} \text{ in} \\ [A_2(r, \gamma^x, (\gamma^y, \\ \quad \gamma^{z} * \\ \quad\quad \pi_b(A_1(r, \gamma^x)))) \overset{?}{=} b] \end{pmatrix} \right| \leq \epsilon_{\text{DDH}}$$

which is **G3** except that $\frac{1}{2}$ is replaced by the probability of another game. We are thus left to prove that this probability is equal to $\frac{1}{2}$:

$$\mathbf{Pr}_{\text{true}} \begin{pmatrix} \text{let } x \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } y \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } r \overset{R}{\leftarrow} R \text{ in} \\ \text{let } b \overset{R}{\leftarrow} \{1,2\} \text{ in} \\ \text{let } z \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ [A_2(r, \gamma^x, (\gamma^y, \gamma^z * \pi_b(A_1(r, \gamma^x)))) \overset{?}{=} b] \end{pmatrix} = \frac{1}{2}$$

G5. We replace the randomly uniform choice of z and the computation γ^z with a random choice of an element of G (by Corollary 3.16).

$$\mathbf{Pr}_{\text{true}} \begin{pmatrix} \text{let } x \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } y \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } r \overset{R}{\leftarrow} R \text{ in} \\ \text{let } b \overset{R}{\leftarrow} \{1,2\} \text{ in} \\ \text{let } m_z \overset{R}{\leftarrow} G \text{ in} \\ [A_2(r, \gamma^x, (\gamma^y, m_z * \pi_b(A_1(r, \gamma^x)))) \overset{?}{=} b] \end{pmatrix} = \frac{1}{2}$$

G6. We delete the right operand of $*$ (by Corollary 3.17):

$$\mathbf{Pr}_{\text{true}} \begin{pmatrix} \text{let } x \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } y \overset{R}{\leftarrow} \mathbb{Z}_q \text{ in} \\ \text{let } r \overset{R}{\leftarrow} R \text{ in} \\ \text{let } m_z \overset{R}{\leftarrow} G \text{ in} \\ \text{let } b \overset{R}{\leftarrow} \{1,2\} \text{ in} \\ [A_2(r, \gamma^x, (\gamma^y, m_z)) \overset{?}{=} b] \end{pmatrix} = \frac{1}{2}$$

This is true by Corollary 3.6. □

5.2 The hashed ElGamal encryption scheme

The simplest version of ElGamal does not use hash functions. However, in practice, it is more convenient to consider messages which are bit strings (say of length l) instead of elements of a cyclic group. The hashed version of the ElGamal encryption scheme allows for this. We assume that we are given an entropy-smoothing family of hash functions $(H_k)_{k\in K}$, each H_k being a function from G to $\{0,1\}^l$. The ElGamal encryption scheme consists in the following probabilistic algorithms:

- The key generation algorithm $\mathsf{keygen}()$:
  $\mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in}\ \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in}\ [((\gamma^x, k), (x, k))]$
- The encryption algorithm $\mathsf{encrypt}((\alpha, k), m)$:
  $\mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in}\ [(\gamma^y, H_k(\alpha^y) \oplus m)]$
- The decryption algorithm $\mathsf{decrypt}((x, k), c)$:
  $[H_k(\pi_1(c)^x) \oplus \pi_2(c)]$

Messages are elements of $\{0,1\}^l$; public keys are elements of $G \times K$; secret keys are elements of $\mathbb{Z}_q \times K$; ciphertexts are elements of $G \times \{0,1\}^l$.

To prove that hashed ElGamal is a correct encryption scheme (as defined in Section 4.4), we apply backward and recursively Proposition 3.11 until we are left with the following equation which obviously holds: $m = H_k(\gamma^{xy}) \oplus (H_k(\gamma^{yx}) \oplus m)$.

Theorem 5.2. *The hashed ElGamal encryption scheme is semantically secure.*

Proof. Again, in this proof we implicitly apply Corollaries 3.7 and 3.8, and Proposition 3.9.

Let us fix A_1 and A_2. We proceed by successive game transformations.

G0. By definition of semantic security, we must prove that:

$$\left| \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ (kp, ks) \Leftarrow \mathsf{keygen}()\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in}\ \mathsf{let}\ (m_1, m_2) \leftarrow A_1(r, kp)\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in}\ \mathsf{let}\ c \Leftarrow \mathsf{encrypt}(kp, m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b} \leftarrow A_2(r, kp, c)\ \mathsf{in} \\ [\hat{b} \overset{?}{=} b] \end{pmatrix} - \frac{1}{2} \right| \text{ is negligible}$$

G1. Knowing that ϵ_{DDH} and ϵ_{ES} are negligible, we are led to prove that:

$$\left| \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ (kp, ks) \Leftarrow \mathsf{keygen}()\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in}\ \mathsf{let}\ (m_1, m_2) \leftarrow A_1(r, kp)\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in}\ \mathsf{let}\ c \Leftarrow \mathsf{encrypt}(kp, m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b} \leftarrow A_2(r, kp, c)\ \mathsf{in} \\ [\hat{b} \overset{?}{=} b] \end{pmatrix} - \frac{1}{2} \right| \leq \epsilon_{\mathrm{DDH}} + \epsilon_{\mathrm{ES}}$$

G2. We unfold definitions of oracles and apply associativity of let (by Theorem 3.3 (6)).

$$\left| \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}(kp, ks) \leftarrow ((\gamma^x, k), (x, k))\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ (m_1, m_2) \leftarrow A_1(r, kp) \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ c \leftarrow (\gamma^y, H_{\pi_2(kp)}(\pi_1(kp)^y) \oplus m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b} \leftarrow A_2(r, kp, c)\ \mathsf{in} \\ [\hat{b} \overset{?}{=} b] \end{pmatrix} - \frac{1}{2} \right| \leq \epsilon_{\mathrm{DDH}} + \epsilon_{\mathrm{ES}}$$

G3. We propagate definitions of kp, ks, m_1, m_2, c and $\hat{b}$ (by Theorem 3.3 (4)).

$$\left| \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ [A_2(r, (\gamma^x, k), (\gamma^y, H_k(\gamma^{xy}) \oplus \pi_b(A_1(r, (\gamma^x, k))))) \overset{?}{=} b] \end{pmatrix} - \frac{1}{2} \right| \leq \epsilon_{\text{DDH}} + \epsilon_{\text{ES}}$$

G4. According to DDH assumption, we have that:

$$\left| \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ [A_2(r, (\gamma^x, k), (\gamma^y, \\ \quad H_k(\gamma^{xy}) \oplus \\ \quad \pi_b(A_1(r, (\gamma^x, k))))) \overset{?}{=} b] \end{pmatrix} - \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ z \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ [A_2(r, (\gamma^x, k), (\gamma^y, \\ \quad H_k(\gamma^{z}) \oplus \\ \quad \pi_b(A_1(r, (\gamma^x, k))))) \overset{?}{=} b] \end{pmatrix} \right| \leq \epsilon_{\text{DDH}}$$

where the left-hand side game is the one from **G3**. We are thus left to prove that[6]:

$$\left| \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ \mathsf{let}\ z \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ [A_2(r, (\gamma^x, k), (\gamma^y, H_k(\gamma^{z}) \oplus \pi_b(A_1(r, (\gamma^x, k))))) \overset{?}{=} b] \end{pmatrix} - \frac{1}{2} \right| \leq \epsilon_{\text{ES}}$$

G5. We replace the randomly uniform choice of z and the computation γ^z with a random choice of an element of G (by Corollary 3.16).

$$\left| \mathbf{Pr}_{\mathsf{true}} \begin{pmatrix} \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ m_z \overset{R}{\leftarrow} G\ \mathsf{in} \\ \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ [A_2(r, (\gamma^x, k), (\gamma^y, H_k(m_z) \oplus \pi_b(A_1(r, (\gamma^x, k))))) \overset{?}{=} b] \end{pmatrix} - \frac{1}{2} \right| \leq \epsilon_{\text{ES}}$$

[6] Indeed, for all r_1, r_2, r_3, $r_{1,2}$, $r_{2,3}$, in order to prove that $|r_1 - r_3| \leq r_{1,2} + r_{2,3}$, it is sufficient to prove that $|r_1 - r_2| \leq r_{1,2}$ and $|r_2 - r_3| \leq r_{2,3}$.

G6. According to the entropy-smoothing assumption, we have that:

$$\left| \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ m_z \overset{R}{\leftarrow} G\ \mathsf{in} \\ \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ {[}A_2(r, (\gamma^x, k), (\gamma^y, \\ \quad H_k(m_z) \oplus \\ \quad \pi_b(A_1(r, (\gamma^x, k))))) \overset{?}{=} b] \end{array} \right) - \mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ h \overset{R}{\leftarrow} \{0,1\}^l\ \mathsf{in} \\ \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ {[}A_2(r, (\gamma^x, k), (\gamma^y, \\ \quad h \oplus \\ \quad \pi_b(A_1(r, (\gamma^x, k))))) \overset{?}{=} b] \end{array} \right) \right| \leq \epsilon_{\mathrm{ES}}$$

which is **G5** except that $\frac{1}{2}$ is replaced by the probability of another game. We are thus left to prove that this probability is equal to $\frac{1}{2}$:

$$\mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ \mathsf{let}\ h \overset{R}{\leftarrow} \{0,1\}^l\ \mathsf{in} \\ {[}A_2(r, (\gamma^x, k), \gamma^y, h \oplus \pi_b(A_1(r, (\gamma^x, k))) \overset{?}{=} b] \end{array} \right) = \frac{1}{2}$$

G7. We delete the right operand of $\oplus$ (by Corollary 3.18):

$$\mathbf{Pr}_{\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ k \overset{R}{\leftarrow} K\ \mathsf{in} \\ \mathsf{let}\ x \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \overset{R}{\leftarrow} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \overset{R}{\leftarrow} R\ \mathsf{in} \\ \mathsf{let}\ h \overset{R}{\leftarrow} \{0,1\}^l\ \mathsf{in} \\ \mathsf{let}\ b \overset{R}{\leftarrow} \{1,2\}\ \mathsf{in} \\ {[}A_2(r, (\gamma^x, k), \gamma^y, h) \overset{?}{=} b] \end{array} \right) = \frac{1}{2}$$

This is true by Corollary 3.6. □

6 Implementation in the proof assistant Coq

The proof assistant Coq. Coq is a goal-directed proof assistant. This means that if we are trying to prove that a formula Q (the goal) is true, and we have a theorem stating that P_1 & P_2 implies Q, then we can apply this theorem. Coq will replace the goal Q by two subgoals P_1 and P_2. We proceed this way until we finally reach goals that are either axioms or are true by definition. On the way, Coq builds a so-called proof term. The critical part of Coq is its kernel which takes a proof term as an input and checks whether it is correct or not. On top of that there is a script language which allows users to state theorems and build their proofs interactively. This script language includes predefined tactics to prove automatically some mathematical statements such as tautologies, Presburger arithmetic statements, linear inequations over real numbers… Users can also define their own tactics.

Our framework in Coq. Our current implementation consists of the following Coq files:

CoqLib.v addendum to the Coq standard library
Distrib.v distributions, probabilities and necessity
Equiv.v equivalence modulo a negligible probability
DistribAuto.v automatically generated properties of distributions
Group.v basic group theory, cyclic groups
GroupProba.v probabilities over cyclic groups
BitString.v bit strings
Challenge.v correctness and security games
DDH.v the DDH assumption
Hash.v hash functions, entropy smoothing
Tactic.v support for automation
CryptoGames.v the main file including the full library
ElGamal.v correctness and semantic security for ElGamal
HashedElGamal.v correctness and semantic security for hashed ElGamal

Our library consists of 3381 lines of Coq and O'Caml code. The O'Caml part is a program which generates automatically 5923 other lines of Coq code. By using our library, the proofs of correctness and semantic security for ElGamal and hashed ElGamal consists respectively of only 160 lines and 209 lines of Coq code. This shows that our framework, while allowing for fully formal and readable security proofs, is scalable. Therefore, we believe that it can be further extended and applied to much more involved security proofs.

We write games as Coq functions and reason on them using the full logic of Coq: this is a so-called shallow embedding. We use Coq notations which allow for games and formulas to be written in a syntax close to the one used in this paper. For example, the game **G1** in the proof of Theorem 5.2 appears in Coq as:

```
mlet k <~ keygen in
mlet r <$ seed in
mlet mm <- A1 r (fst k) in
mlet b <$ [true; false] in
mlet c <~ encrypt (fst k) (if b then fst mm else snd mm) in
mlet b' <- A2 r (fst k) c in
[[eqb b' b]]
```

Probabilistic choices occurring in games are modeled with a monad. A similar encoding of randomized algorithms was given in [2]. However our encoding is much simpler due to the fact that it is enough for our purpose to consider distributions which are finite. In order to be able to compute a probability, we need to decide whether a predicate is true or not for each value of a distribution. We thus restrict ourselves to decidable predicates, i.e. predicates that can be encoded as computable functions into booleans.

We provide automated tactics which can move deterministic assignments, random choices and calls to oracles from one place to another inside the game, and prove automatically that this transformation leads to an equivalent game. Those tactics are defined in the file **Tactic.v**. When proving correctness of an encryption scheme, the backward and recursive application of Proposition 3.11 is dealt with automatically by the following tactic defined in the file **Distrib.v**:

```
Ltac necessarily :=
  repeat (
    match goal with
    | |- Necessarily _ _ (unit _) => apply nec_unit
    | |- Necessarily _ _ (bind _ _) => apply nec_bind; intros
    | |- Necessarily _ _ (uniform _) => apply nec_uniform
    end;
    intros
  ).
```

It parses the current game and, depending on the root of the syntax tree, applies the corresponding implication of Proposition 3.11.

Difficulties. A trouble with Coq and similar proof assistants based on intensional type theory is the way they deal with equality. Equality is important because it allows for replacing a subterm by an equal one. The equality in Coq is intensional: roughly speaking, two expressions are equal iff they have the same definition (modulo β-equivalence). But in mathematics we commonly use extensional equality for functions: roughly speaking, two functions f and g are equal iff for any x, $f(x) = g(x)$. Fortunately, it is safe to assume an axiom stating that the extensional equality for functions implies the intensional one. This axiom is consistent because, whenever two functions are extensionally equal, it is not provable in Coq that those two functions are not intensionally equal. This axiom can be seen in many Coq developments and is one of the recurring questions on the Coq mailing-list. It is important for our formalization because we deal with functions: for example, $\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x)$ is encoded in Coq by `bind` δ (`fun` x `=>` $\varphi(x)$).

Because the sets and multisets we deal with are finite, we can simply encode them as lists. Equality of sets (or multisets) is then defined appropriately. As in the case of extensional equality for functions, because equality of sets is not the intensional equality of Coq, we cannot replace at will a set (or a multiset) with an equal one. And the trick used for functions consisting in adding an axiom permitting this cannot be used as, in this case, it would lead to inconsistency! It might be profitable to use the Coq mechanism for setoids which allows for dealing with such user-defined equalities in a transparent way. However one can only do replacements in contexts which are syntactic compositions of morphisms that have been proved compatible with the user-defined equality. Another way would be to enforce a normal form by using a dependent type so that user-defined equality for terms is exactly Coq's equality for terms in normal form.

Also, with Coq it is not possible to replace a term by an equal one under a function binder. Thus, in order to rewrite inside a let, we need to go through Corollaries 3.7 and 3.8.

Another limitation of Coq is that, When a goal contains a subterm of the form $\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x)$, the tactic language of Coq cannot parse properly the $\varphi(x)$. This is due to the fact that unification in the tactic language of Coq is limited to non-linear first order unification and therefore cannot parse under lambdas.

7 Conclusions and future work

We have proposed a framework for formalizing game-based proofs of cryptographic schemes. It allows for security proofs which are precise enough to be mechanically checked, and readable enough to be humanly checked. We have implemented it as a library of theorems and tactics for the proof assistant Coq. We have illustrated its use by proving semantic security for the encryption scheme ElGamal and its hashed version. Our proofs are close to the ones given by Shoup [20] but more precise. The main advantages of using a proof assistant are that reasoning errors are not possible and that all the assumptions must be stated. On the other hand, all tedious details of the proof must be dealt with: you cannot simply claim that something is obvious. This is why we need to develop libraries which deal once and for all with those details.

A possible research direction is to formalize more security notions in our framework. Another extension is to extend the mathematical framework with more on groups in order to be able to deal with more advanced cryptographic schemes such as the Weil pairing used in elliptic curve cryptography and identity based encryption. It is also possible to extend our framework with a syntax for games, and with an associated semantics in terms of distributions. In order to increase automation and have simpler security proofs, one possible direction is to make such an extension and prove that some syntactic transformations are correct. A syntax would help to overcome the impossibility we have currently to parse under let in the tactic language, and would then allow us to write more powerful tactics by reflection which would for example take care automatically of rewriting under let, or reordering uniform choices.

Acknowledgements. We would like to thank Reynald Affeldt for having directed us to this research area in the first place, and for helpful discussions. We are also grateful to Yang Cui, Nicolas Marti, Kirill Morozov, Miki Tanaka and Rui Zhang for fruitful discussions.

References

1. R. Affeldt, M. Tanaka, and N. Marti. Formal proof of provable security by game-playing in a proof assistant. In *ProvSec 2007*, volume 4784 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2007.
2. P. Audebaud and C. Paulin-Mohring. Proofs of randomized algorithms in Coq. In *MPC*, volume 4014 of *Lecture Notes in Computer Science*, pages 49–68. Springer, 2006.
3. G. Barthe and S. Tarento. A machine-checked formalization of the random oracle model. In *TYPES*, volume 3839 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2004.
4. M. Bellare and P. Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. In *Eurocrypt*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416, 1996.
5. M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. `http://eprint.iacr.org/`.
6. B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.
7. B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. Cryptology ePrint Archive, Report 2006/069, 2006. `http://eprint.iacr.org/`.
8. D. Boneh. The Decision Diffie-Hellman problem. In *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 1998.
9. R. Corin and J. den Hartog. A probabilistic Hoare-style logic for game-based cryptographic proofs. In *ICALP*, volume 4052 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2006.
10. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
11. D. Dolev and A. C.-C. Yao. On the security of public key protocols (extended abstract). In *FOCS*, pages 350–357. IEEE, 1981.
12. T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
13. S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.
14. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
15. S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005. `http://eprint.iacr.org/`.
16. J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006.
17. E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
18. D. Nowak. A framework for game-based security proofs. In *ICICS 2007*, volume 4861 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2007.
19. N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, 2002.
20. V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.
21. Y. Tsiounis and M. Yung. On the security of ElGamal based encryption. In *Public Key Cryptography*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 1998.
22. P. Wadler. Comprehending monads. In *LISP and Functional Programming*, pages 61–78, 1990.