A Framework for Game-Based Security Proofs *

David Nowak

Research Center for Information Security, AIST, Tokyo

May 28, 2007

Abstract. Information security is nowadays an important issue. Its essential ingredient is cryptography. A common way to present security proofs is to structure them as sequences of games. The main contribution of this paper is a framework which refines this approach. We make explicit important theorems used implicitly by cryptographers but never explicitly stated. Our aim is to have a framework in which proofs are precise enough to be mechanically checked, and 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. All proofs have been mechanically checked in the proof assistant Coq.
Keywords: cryptography, monad, probability, proof assistant, games

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 has become essentially unverifiable" [BR04]. 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 [BR04] and [Sho04] 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 (often 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

* Full version

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.

We propose a formalization of games which is based on [Sho04] where games are seen as probability distributions. The main contribution of this paper is a framework which refines this approach. We make explicit important theorems used implicitly by cryptographers but never explicitly stated. This is in particular the case of theorems connecting group theory and probability theory. Our aim is to have a framework in which proofs are precise enough to be mechanically checked, and 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 [Elg85]. 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 [DH76], it can be proved semantically secure. Our framework is implemented as a library of theorems for the proof assistant Coq [Coq07] and its usability is shown by its application to the proof of semantic security for the ElGamal encryption scheme[1].

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. 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

The approach to game-based proofs by Shoup [Sho04] differs from the one by Bellare and Rogaway [BR04] where 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 [BR04] 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.

The so-called generic model and random oracle model have been formalized in Coq for making proofs on signature schemes [BCT04]. In particular, it was applied in [BT04] to the ElGamal signature scheme (not to be confused with ElGamal encryption scheme we are dealing with is Section 5). In contrast to our approach, it is not based on sequences of games which had not yet been popularized by [Sho04] and [BR04].

[1] The source code is available on request.

CryptoVerif is a software for automated security proofs with sequences of games [BP06a]. It is illustrated with a proof of the Full-Domain Hash (FDH) signature scheme [BR96]. However this proof relies on certain equivalences that have to be introduced by the user. Those equivalences are proved manually in Appendix B of [BP06b]. These are difficult parts of the proof that cannot be handled by CryptoVerif.

The closest work to ours is the probabilistic Hoare-style logic which has been proposed (but not implemented) in [CdH06] to formalize game-based proofs. In their and our framework we do not need to define precisely the terms *efficient* and *negligible*. However those terms can be given precise definitions in terms of polynomials. The logic in [CdH06] allows for rigorous proofs but differs from game-based proofs by cryptographers. Indeed, because they model games as imperative programs, they are led to use Hoare-style logic. They illustrate their logic by proving semantic security of the ElGamal encryption scheme. However, in this example they do not use while loops. It is thus not clear in which way such imperative language for games and its associated Hoare-style logic are beneficial. In our approach, logical reasoning is closer to the one used by cryptographers: we avoid imperative features such as assignments and while loops, which are at the crux of Hoare logic. It is possible because the variables used in [Sho04] are mathematical variables in the sense that they are defined once and only once whereas the value of a variable in an imperative program can change in the course of execution. By the way, the property that a variable is defined once and only once is enforced by 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. We finally obtain a simpler proof of semantic security of ElGamal.

3 Mathematical framework

In this section we give the mathematical bases on which rely our framework: probabilities, cyclic groups and properties relating them.

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. And indeed probability distributions have a monadic structure [RP02,APM06]. In our case we only need to consider the simpler case of finite probability distributions.

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*.

We define the basic operations which will be used to define games and oracles.

Definition 3.2 (Operations).

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

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

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

where, in (2), $\varphi(x)$ denotes a finite probability distribution parameterized by a value x, $\bigcup$ is the union of multisets, and:

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

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

In the rest of this paper, we use the following syntactic sugar:

(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 $\mathsf{let}\ x \Leftarrow \bigoplus A\ \mathsf{in}\ \varphi(x)$.

The intuitive meaning of $\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x)$ is that it selects randomly one value x from the distribution δ and passes it to the function φ. 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).

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

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

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

Equation (4) allows for propagating constants. Equation (6) states associativity which will help for simplifying presentation of games.

The following proposition allows for rewriting under a let.

Proposition 3.4 (extensionality of let). *for all sets A and B, for any finite distribution $\delta \in \Delta_A$, for all functions φ and ψ from A to Δ_B, if* $\forall (a,p) \in \delta \cdot \varphi(a) = \psi(a)$ *then* $\mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \varphi(x) = \mathsf{let}\ x \Leftarrow \delta\ \mathsf{in}\ \psi(x)$

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 3.5 (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 $\mathbf{Pr}\left((\lambda x \cdot x = \mathsf{true})(\delta)\right)$ where $\lambda x \cdot x = \mathsf{true}$ is the predicate that holds iff its argument 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 3.6. *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 3.7. *For any finite set A, for any $a \in A$,*

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

As another corollary, we obtain a mean to replace a randomly uniform choice in a goal by a universal quantifier[2].

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

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

[2] 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.

The reverse implication is not true. We can see that on a counterexample: if the reverse implication was true, from Corollary 3.7 we could deduce that $\forall x \in A \cdot \mathbf{Pr}_{=\mathsf{true}}\left([x \stackrel{?}{=} a]\right) = \frac{1}{|A|}$. 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. This is particularly interesting as it shows us a fundamental difference between universal quantification and random choice.

The following proposition allows for moving around randomly uniform choices in the definitions of games.

Proposition 3.9. *For all finite sets A, B and C, for any $\delta \in \Delta_B$, for any $f : A \times B \to C$, if x does not occur freely in δ then:*

$$\mathbf{Pr}\left(P\begin{pmatrix}\mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\\ \mathsf{let}\ y \Leftarrow \delta\ \mathsf{in}\\ \varphi(x,y)\end{pmatrix}\right) = \mathbf{Pr}\left(P\begin{pmatrix}\mathsf{let}\ y \Leftarrow \delta\ \mathsf{in}\\ \mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\\ \varphi(x,y)\end{pmatrix}\right)$$

In practice, we will use the following corollaries.

Corollary 3.10. *For all finite sets A, B and C, for any $b \in B$, for any $f : A \times B \to C$,*

$$\mathbf{Pr}\left(P\begin{pmatrix}\mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\\ \mathsf{let}\ y \leftarrow b\ \mathsf{in}\\ \varphi(x,y)\end{pmatrix}\right) = \mathbf{Pr}\left(P\begin{pmatrix}\mathsf{let}\ y \leftarrow b\ \mathsf{in}\\ \mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\\ \varphi(x,y)\end{pmatrix}\right)$$

Corollary 3.11. *For all finite sets A, B and C, for any $f : A \times B \to C$,*

$$\mathbf{Pr}\left(P\begin{pmatrix}\mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\\ \mathsf{let}\ y \overset{R}{\leftarrow} B\ \mathsf{in}\\ \varphi(x,y)\end{pmatrix}\right) = \mathbf{Pr}\left(P\begin{pmatrix}\mathsf{let}\ y \overset{R}{\leftarrow} B\ \mathsf{in}\\ \mathsf{let}\ x \overset{R}{\leftarrow} A\ \mathsf{in}\\ \varphi(x,y)\end{pmatrix}\right)$$

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 3.12 (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 $\mathbf{Pr}\left(P(\delta)\right) = 1$.

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

Proposition 3.13.

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

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

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

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

Proposition 3.15. *For any $b \in G$, $\{a * b \mid a \in G\} = G$*

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.

We write $\mathbb{Z}_q$ for the set of integers $\{0, \ldots, q-1\}$.

Theorem 3.16. *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,*

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

Corollary 3.17. *for any set A, for any function f from G to A, for any predicate P on A,*

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

Proof. By Proposition 3.14, $\gamma^{\text{-}}$ is bijective. We can thus apply Theorem 3.16. □

Corollary 3.18. *for any set A, for any function f from G to A, for any predicate P on A, for any $m' \in G$,*

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

Proof. By Proposition 3.15, $\_ * m'$ is bijective. We can thus apply Theorem 3.16. □

In Section 3.3 of [Sho04] the proof of semantic security for the encryption scheme ElGamal uses implicitly those two 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 3.17, while the "*one sees that*" part corresponds to our corollary 3.18.

4 Formal security

In this section we formalize in our framework two important notions which are fundamental in cryptography: the Decisional Diffie-Hellman assumption (DDH) and the notion of 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[3].

The DDH assumption [DH76] 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 $\Delta_{\{\mathsf{false},\mathsf{true}\}}$:

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

As will be seen in Section 5, security proofs in our framework mainly consist in game transformations. Thus, as in [CdH06], 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 Semantic security

The notion of semantic security was introduced by Goldwasser and Micali [GM82]. They later showed that it is equivalent to indistinguishability under Chosen Plaintext Attack (IND-CPA) [GM84]. We use this latter formulation which is nowadays the most commonly used.

We assume two oracles: a key generation oracle keygen which generates a pair of public and private keys; and an encryption oracle $\mathsf{encrypt}$ 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 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 $\mathsf{encrypt}$ which returns the corresponding ciphertext.

[3] We do not assume that q is prime. However most groups in which DDH is believed to be true have prime order [Bon98].

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:

$$\begin{array}{l}\mathsf{let}\ (k_{\mathrm{p}}, k_{\mathrm{s}}) \Leftarrow \mathsf{keygen}()\ \mathsf{in}\\ \mathsf{let}\ r \xleftarrow{R} R\ \mathsf{in}\ \mathsf{let}\ (m_1, m_2) \leftarrow A_1(r, k_{\mathrm{p}})\ \mathsf{in}\\ \mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in}\ \mathsf{let}\ c \Leftarrow \mathsf{encrypt}(k_{\mathrm{p}}, m_b)\ \mathsf{in}\\ \mathsf{let}\ \hat{b} \leftarrow A_2(r, k_{\mathrm{p}}, c)\ \mathsf{in}\\ [\hat{b} \stackrel{?}{=} b]\end{array}$$

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

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

4.3 Correctness of a cryptographic scheme

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

$$\begin{array}{l}\mathsf{let}\ (k_{\mathrm{p}}, k_{\mathrm{s}}) \Leftarrow \mathsf{keygen}()\ \mathsf{in}\\ \mathsf{let}\ c \Leftarrow \mathsf{encrypt}(k_{\mathrm{p}}, m)\ \mathsf{in}\\ \mathsf{let}\ m' \Leftarrow \mathsf{decrypt}(k_{\mathrm{s}}, c)\ \mathsf{in}\\ [m \stackrel{?}{=} m'].\end{array}$$

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 (keygen, encrypt, decrypt) is correct iff for any message m the correctness game necessarily returns true:

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

5 Application to the ElGamal encryption scheme

We illustrate the use of our framework by proving in a systematic way the so-called semantic security of the encryption scheme ElGamal [Elg85].

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(): $\mathsf{let}\ x \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in}\ [(\gamma^x, x)]$

– The encryption algorithm $\mathsf{encrypt}(k_{\mathrm{p}}, m)$: $\quad \mathsf{let}\ y \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in}\ [(\gamma^y, {k_{\mathrm{p}}}^y * m)]$

– The decryption algorithm $\mathsf{decrypt}(k_{\mathrm{s}}, c)$: $\quad [\pi_2(c)\ *\ \left(\pi_1(c)^{k_{\mathrm{s}}}\right)^{-1}]$

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 4.3), we apply backward and recursively Proposition 3.13 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 3.13 is dealt with automatically in our implementation (See Section 6).

5.2 Semantic security

Theorem 5.1 (Semantic security). *The ElGamal encryption scheme is semantically secure.*

Proof. In this proof we implicitly apply Proposition 3.4, and Corollaries 3.10 and 3.11 which respectively allow for rewriting under let and to reorder uniform choices.

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

G0. By definition of semantic security and knowing that ϵ_{DDH} is negligible, we are led to prove that:

$$\left| \mathbf{Pr}_{=\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ (k_{\mathrm{p}}, k_{\mathrm{s}})\ \Leftarrow\ \mathsf{keygen}()\ \mathsf{in} \\ \mathsf{let}\ r\ \xleftarrow{R}\ R\ \mathsf{in}\ \mathsf{let}\ (m_1, m_2)\ \leftarrow\ A_1(r, k_{\mathrm{p}})\ \mathsf{in} \\ \mathsf{let}\ b\ \xleftarrow{R}\ \{1,2\}\ \mathsf{in}\ \mathsf{let}\ c\ \Leftarrow\ \mathsf{encrypt}(k_{\mathrm{p}}, m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b}\ \leftarrow\ A_2(r, k_{\mathrm{p}}, \pi_1(c), \pi_2(c))\ \mathsf{in} \\ [\hat{b}\ \stackrel{?}{=}\ b] \end{array} \right) - \frac{1}{2} \right| \ \leq\ \epsilon_{\mathrm{DDH}}$$

G1. We unfold definitions of oracles and apply associativity of let.

$$\left| \mathbf{Pr}_{=\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x\ \xleftarrow{R}\ \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}(k_{\mathrm{p}}, k_{\mathrm{s}})\ \leftarrow\ (\gamma^x, x)\ \mathsf{in} \\ \mathsf{let}\ r\ \xleftarrow{R}\ R\ \mathsf{in} \\ \mathsf{let}\ (m_0, m_1)\ \leftarrow\ A_1(r, k_{\mathrm{p}}) \\ \mathsf{let}\ b\ \xleftarrow{R}\ \{1,2\}\ \mathsf{in} \\ \mathsf{let}\ y\ \xleftarrow{R}\ \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ c\ \leftarrow\ (\gamma^y, {k_{\mathrm{p}}}^y \cdot m_b)\ \mathsf{in} \\ \mathsf{let}\ \hat{b}\ \leftarrow\ A2(r, k_{\mathrm{p}}, \pi_1(c), \pi_2(c))\ \mathsf{in} \\ [\hat{b}\ \stackrel{?}{=}\ b] \end{array} \right) - \frac{1}{2} \right| \ \leq\ \epsilon_{\mathrm{DDH}}$$

G2. We propagate definitions of k_{p}, k_{s}, m_0, m_1, c and $\hat{b}$ (by Theorem 3.3 (4)).

$$\left| \mathbf{Pr}_{=\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \xleftarrow{R} R\ \mathsf{in} \\ \mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in} \\ \mathsf{let}\ y \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ [A2(r, \gamma^x, \gamma^y, \gamma^{xy} \cdot \pi_b(A_1(r, \gamma^x)) \stackrel{?}{=} b] \end{array} \right) - \frac{1}{2} \right| \leq \epsilon_{\mathrm{DDH}}$$

G3. According to DDH assumption, we have that:

$$\left| \mathbf{Pr}_{=\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \xleftarrow{R} R\ \mathsf{in} \\ \mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in} \\ [A2(r, \gamma^x, \gamma^y, \\ \quad \gamma^{xy} \cdot \pi_b(A_1(r, \gamma^x))) \stackrel{?}{=} b] \end{array} \right) - \mathbf{Pr}_{=\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ z \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \xleftarrow{R} R\ \mathsf{in} \\ \mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in} \\ [A2(r, \gamma^x, \gamma^y, \\ \quad \gamma^z \cdot \pi_b(A_1(r, \gamma^x))) \stackrel{?}{=} b] \end{array} \right) \right| \leq \epsilon_{\mathrm{DDH}}$$

We are thus left to prove that:

$$\mathbf{Pr}_{=\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ z \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \xleftarrow{R} R\ \mathsf{in} \\ \mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in} \\ [A2(r, \gamma^x, \gamma^y, \gamma^z \cdot \pi_b(A_1(r, \gamma^x)) \stackrel{?}{=} b] \end{array} \right) = \frac{1}{2}$$

G4. We replace the randomly uniform choice of z and the computation γ^z with a random choice of an element of G (by Corollary 3.17).

$$\mathbf{Pr}_{=\mathsf{true}} \left(\begin{array}{l} \mathsf{let}\ x \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ y \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in} \\ \mathsf{let}\ r \xleftarrow{R} R\ \mathsf{in} \\ \mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in} \\ \mathsf{let}\ m_z \xleftarrow{R} G\ \mathsf{in} \\ [A2(r, \gamma^x, \gamma^y, m_z \cdot \pi_b(A_1(r, \gamma^x)) \stackrel{?}{=} b] \end{array} \right) = \frac{1}{2}$$

G5. We apply Corollary 3.18.

$$\mathbf{Pr}_{=\mathsf{true}}\left[\begin{array}{l}\mathsf{let}\ x \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in}\\ \mathsf{let}\ y \xleftarrow{R} \mathbb{Z}_q\ \mathsf{in}\\ \mathsf{let}\ r \xleftarrow{R} R\ \mathsf{in}\\ \mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in}\\ \mathsf{let}\ m_z \xleftarrow{R} G\ \mathsf{in}\\ [A2(r,\gamma^x,\gamma^y,m_z) \stackrel{?}{=} b]\end{array}\right] = \frac{1}{2}$$

G6. We apply Corollary 3.8.

$$\forall x \in \mathbb{Z}_q \cdot \forall y \in \mathbb{Z}_q \cdot \forall r \in R \cdot \forall m_z \in G \cdot \mathbf{Pr}_{=\mathsf{true}}\left[\begin{array}{l}\mathsf{let}\ b \xleftarrow{R} \{1,2\}\ \mathsf{in}\\ [A2(r,\gamma^x,\gamma^y,m_z) \stackrel{?}{=} b]\end{array}\right] = \frac{1}{2}$$

This is true by Corollary 3.7. □

6 Implementation in the proof assistant Coq

The proof assistant Coq. Coq is 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 inequation over real numbers... Users can also define their own tactics.

Our framework in Coq. Our current implementation consists in the 6 following files:

CoqLib.v addendum to the Coq standard library

Distr.v distributions, probabilities and necessity modality

Group.v basic group theory, cyclic groups

GroupProba.v probabilities over cyclic groups

DDH.v the DDH assumption

ElGamal.v semantic security and application to ElGamal

The whole implementation consists in 2000 lines of source code where about 20% is specific to ElGamal. This shows that our framework, while allowing for fully

detailed and readable security proofs, is scalable. Therefore, we believe that it can be further extended and applied to much more involved security proofs.

Because the language we use for games is functional, we can write games as Coq functions and reason on them using the full logic of Coq: this is a so-called shallow embedding. Probabilistic choices occurring in games are modeled with a monad. A similar encoding of randomized algorithms was given in [APM06]. 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 a computable functions into booleans.

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 **G0** appears in Coq as:

```
Rabs
  (probability
    (fun b : bool => b=true) (fun b : bool => bool_dec b true)
    (
     mlet k := keygen in
     mlet r := uniform 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
     [eqb (A2 r (fst k) (fst c) (snd c)) b]
    )
    - 1 / 2) <= epsilon_DDH
```

When proving correctness of an encryption scheme, the backward and recursive application of Proposition 3.13 is dealt with automatically by the following tactic defined in the file **Distr.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 equation of Proposition 3.13. In the file **ElGamal.v**, We define tactics to replace the current game by a new game given as a parameter. The tactic `new_game0` replaces the current game by a β-equivalent new game and thus do not generate any additional subgoal; it fails if the game provided as an

argument is not β-equivalent to the current game. The tactic `new_game1` replaces the current game by a new game and requires the user to prove that those two games are equal. The tactic `new_game2` replaces the current game by a new game and requires the user to prove that this does not change the resulting probability.

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 function 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 (See `http://coq.inria.fr/`). 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 $\texttt{bind}\ \delta\ (\lambda x \cdot \varphi(x))$.

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 Proposition 3.4.

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. Our proof is close to the one given by Shoup [Sho04] 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.

As future work, we plan to formalize more security notions in our framework. Another extension is obviously to extend the mathematical framework with more on groups in order to be able to deal with much more advanced cryptographic schemes such as the Weil pairing used in elliptic curve cryptography and identity based encryption. It is also easy 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, we plan 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 Nicolas Marti, Kirill Morozov, Miki Tanaka and Rui Zhang for fruitful discussions.

References

[APM06] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in Coq. In *MPC*, volume 4014 of *Lecture Notes in Computer Science*, pages 49–68. Springer, 2006.

[BCT04] Gilles Barthe, Jan Cederquist, and Sabrina Tarento. A machine-checked formalization of the generic model and the random oracle model. In *IJCAR*, volume 3097 of *Lecture Notes in Computer Science*, pages 385–399. Springer, 2004.

[Bon98] Dan Boneh. The Decision Diffie-Hellman problem. In *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 1998.

[BP06a] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.

[BP06b] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. Cryptology ePrint Archive, Report 2006/069, 2006. `http://eprint.iacr.org/`.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. In *EUROCRYPT*, pages 399–416, 1996.

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. `http://eprint.iacr.org/`.

[BT04] Gilles Barthe and Sabrina 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.

[CdH06] Ricardo Corin and Jerry 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.

[Coq07] Coq Development Team. *The Coq Proof Assistant Reference Manual – Version 8.1*, 2007. `http://coq.inria.fr/`.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[Elg85] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[RP02] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, 2002.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.