A Machine-Checked Formalization of the Random Oracle Model

Gilles Barthe and Sabrina Tarento

INRIA Sophia-Antipolis, France
{Gilles.Barthe,Sabrina.Tarento}@sophia.inria.fr

Abstract. Most approaches to the formal analysis of cryptography protocols make the perfect cryptographic assumption, which entails for example that there is no way to obtain knowledge about the plaintext pertaining to a ciphertext without knowing the key. Ideally, one would prefer to abandon the perfect cryptography hypothesis and reason about the computational cost of breaking a cryptographic scheme by achieving such goals as gaining information about the plaintext pertaining to a ciphertext without knowing the key. Such a view is permitted by non-standard computational models such as the Generic Model and the Random Oracle Model. Using the proof assistant Coq, we provide a machine-checked account of the Generic Model and the Random Oracle Model. We exploit this framework to prove the security of the ElGamal cryptosystem against adaptive chosen ciphertexts attacks.

1 Introduction

Cryptographic mechanisms provide a fundamental mechanism to ensure security, and are used pervasively in numerous application domains, including distributed systems and web services. However, designing secure cryptographic mechanisms is extremely difficult to achieve [1]. Therefore, there is an increasing trend to study provable security of cryptographic schemes, whereby one provides a clear specification of the security requirements, and establish with complexity-theoretic arguments that the proposed scheme meets the requirements [18]. Typically, the security of the scheme is established by showing that the attacker has a negligible advantage, i.e. that its chance of succeeding in launching an attack that exploits its capabilities is not significantly higher than its chance of breaking the scheme by brute force. While provable cryptography has become an important tool, it is not unusual to see attacks against cryptographic schemes that were deemed sound using methods from provable security; in most cases, such attacks will exploit an hidden assumption, e.g. that some event occurs with negligible probability.

The objective of our work, initiated in [4], is to machine-check results from provable cryptography. In [4], we use the proof assistant Coq [7] to establish the security of cryptographic schemes, using the Generic Model or GM for short [16, 11], which provides a non-standard computational model for reasoning about the probability and computational cost of breaking a cryptographic scheme. Our

work demonstrates that the benefits of machine-checking results from provable cryptography are two-fold: firstly, we are able to give a precise description of the models that underlie proofs in provable cryptography. Secondly, we are able to provide accurate statements about the security of cryptographic schemes, and to highlight hidden assumptions or approximations of the attacker's advantage in published proofs. However, the formalization of [4] only focuses on non-interactive attacks where the attacker tries to break a cryptographic scheme without any interaction with oracles that perform cryptographic operations. This is an important restriction since in most practical scenarios the attacker is able to interact with oracles that provide useful information for launching an attack. Different forms of oracles include hash oracles, which allow the attacker to hash a message, decryption oracles or decryptors for short, which allow the attacker to retrieve the plaintext from (correct) ciphertexts, and signature oracles or signers for short, which allow the attacker to sign messages.

The main contribution of this paper is to extend our security proofs to an interactive setting, building on a combination of the GM and of the Random Oracle Model or ROM for short [5, 9] that assumes the hash function to be collision resistant (i.e. that collisions of random functions have negligibly small probability). As an application of our results, we prove the security of signed ElGamal encryption against strong adaptive chosen ciphertext attacks. Following [4], the key insight in our formalization is a distinction between the symbolic execution of an attack that specifies the behavior of the attacker, and the concrete execution of the attack that can lead the attacker to gain information about the secrets.

In the case of the Generic Model, the attacker tries to gain knowledge about secrets by trying to find so-called collisions, which establish a correlation between two different outputs produced during the (concrete) execution of the attack. In this setting, the distinction between the symbolic level and concrete level takes the following form:

- at the symbolic level, secrets are treated symbolically and the execution of the attack outputs polynomials $p_1 \dots p_t$ whose indeterminates are the secrets used in the cryptographic scheme. At his level, and the attacker constructs polynomials that will be used at the concrete level for gaining some information about secrets;
- at the concrete level, secrets are interpreted and the attacker checks whether collisions occur, i.e. the secrets are a root of some polynomial $p_i - p_j$ (with $i \neq j$) where p_i and p_j are taken from the polynomials $p_1 \dots p_t$ that were constructed at the symbolic level.

In the setting of the Random Oracle Model, we must also account for interactions with oracles. We do so with the same methodology, i.e. we consider symbolic outputs that are built performing symbolic hash computations, and concrete outputs where the symbolic results of hash computations are interpreted.

Contents of the paper The remainder of the paper is organized as follows. Section 2 provides an account of the Generic Model and of the Random Oracle

Models and of their application to ElGamal. Section 3 discusses our formalization of discrete probabilities and polynomials, which are required to prove our main results. Section 4 reviews our formalization of GM. Section 5 deals with interactive attacks using a hash oracle and a decryptor, and shows an application of our results to ElGamal. We conclude in Section 6.

2 A primer on cryptography

2.1 Public-key cryptography

In public key cryptosystems, each participant gets a pair of keys, a public key and a private key. The public key is published, while the private key is kept secret. All communications involve only public keys, and no private key is ever transmitted or shared. The only requirement is that public keys to be associated with their users in a trusted (authenticated) manner (for instance, in a trusted directory). Anyone can send a confidential message by just using public information, but the message can only be decrypted with the right private key, which is in the sole possession of the intended recipient. Furthermore, public-key cryptography can be used not only for privacy (encryption), but also for authentication (digital signatures) and other various techniques [13].

In a public key cryptosystem, the private key is always linked mathematically to the public key. Therefore, it is always possible to attack a public-key system by deriving the private key from the public key. The typical defense against this is to make the problem of deriving the private key from the public key as difficult as possible. For example, ElGamal cryptosystem assume the intractability of the Decisional Diffie Hellman problem, or DDH-problem [8] i.e., given $g^x\ [p]$ and $g^y\ [p]$, it is hard to tell the difference between $g^{xy}\ [p]$ and $g^r\ [p]$ where r is random and p is a prime number.

The Diffie-Hellman key exchange algorithm is usually described as an active exchange of keys by two parties A and B, who have a (publicly known) prime number p and a generator g:

- party A selects a random number x, and transmits $g^x\ [p]$ to B, symbolically $A \longrightarrow B:\ g^x\ [p]$;
- party B selects a random number y, and transmits $g^y\ [p]$ to A, symbolically $B \longrightarrow A:\ g^y\ [p]$;
- both parties communicate using $g^{xy}\ [p]$ as their session key.

ElGamal [10] can be considered as a special case of the Diffie-Hellman key exchange algorithm. In ElGamal, to send a message to a party whose public key is $g^y\ [p]$, we send our own public key, $g^x\ [p]$, and in addition the message is enciphered by multiplying it by $g^{xy}\ [p]$ i.e., an ElGamal ciphertext has the form $(g^y\ [p], mg^{xy}\ [p])$ for a plaintext m; the multiplication also being modulo p.

To sign an ElGamal ciphertext, we add a Schnorr signature to the ciphertext $(g^y\ [p], mg^{xy}\ [p])$: pick random s, compute $c = H(g^s, g^y, mg^{xy})$ where H is a random function chosen at randon over all functions of that type with uniform

probability distribution, and compute $z = s + cy$, then (g^y, mg^{xy}, c, z) is the signed ciphertext.

In this paper, we prove the security of ElGamal encryption against strong adaptive chosen ciphertext attacks CCA, as described e.g. by C.Rackoff and D.Simon [12]. CCA security means that indistinguishability against an adversary that has access to a decryption oracle which it can freely use except for the target ciphertext.

2.2 The Generic Model

The generic model, or GM for short, was introduced by Shoup [16], building upon Nechaev [11], and can be used to provide an overall guarantee that a cryptographic scheme is not flawed [14, 15, 18]. For example, GM is useful for establishing the complexity of the discrete logarithm or the decisional Diffie-Hellman problem, which we describe below.

The GM focuses on generic attacks, i.e. attacks that do not exploit any specific weakness in the underlying mathematical structures, which in the case of GM is a cyclic group G of prime order q. More concretely, the GM focuses on attacks that work for all cyclic groups, and that are independent of the encoding of group elements; in practice, this is achieved by leaving the group G unspecified. Furthermore, the GM constrains the behavior of the attacker so that he cannot access oracles, and can only gain information about the secret through testing group equalities (a.k.a. collisions). In order to test group equalities, the attacker performs repeatedly modular exponentiations of the program inputs, using coefficients that are chosen randomly and with uniform distribution over the probability space $\mathbb{Z}_q$.

More precisely, a generic attacker $\mathcal{A}$ over G is given by its list of secrets, say $s_1, \ldots, s_k \in \mathbb{Z}_q$, its list of inputs, say $g^{l_1}, \ldots, g^{l_{t'}} \in \mathbb{Z}_q$, which depends upon secrets, and a run, which is a sequence of t multivariate exponentiation (mex) steps. For the latter, the attacker selects arbitrarily, and independently of the secrets, the coefficients $a_{i,1}, \ldots, a_{i,t'} \in \mathbb{Z}_q$ and computes for $t' < i \leq t$ the group elements $f_i = mex(a_{i,1}, \ldots, a_{i,t'}, (g^{l_1}, \ldots, g^{l_{t'}})) = \prod_{j=1}^{t'} g^{l_j a_{i,j}}$, where $f_j = g^{l_j}$ for $1 \leq j \leq t'$. The output of the run is the list $f_1, \ldots, f_t$, from which the attacker will test for collisions, i.e. equalities $f_j = f_{j'}$ i.e., $f_j - f_{j'} = 0$ with $1 \leq j < j' \leq t$.

Considering $s_1, \ldots, s_k$ as formal variables over $\mathbb{Z}_q$, $f_j - f_{j'}$ is a polynomial in $\mathbb{Z}_q[s_1, \ldots, s_k]$. The random $s_1, \ldots, s_k$ are statistically independent of the coefficients $a_{j,1}, \ldots, a_{j,t'}$ and $a_{j',1}, \ldots, a_{j',t'}$. The attacker can obtain information about the secrets by solving the equation $f_j - f_{j'}$.

The objective of the GM model is to establish upper bounds for the probability of a generic attacker to be successful. To this end, the GM assumes that a generic attacker $\mathcal{A}$ is successful if it finds a non-trivial collision, i.e. a collision that reveals information about secrets (those collisions which do not reveal information are called trivial, and are defined as collisions that hold with probability 1, i.e. for all choices of secret data), or if not, if it guesses the secrets at random. Rather than considering the probability of an attacker to be successful,

it is convenient to consider its advantage, which is the probability to be successful with respect to an attacker who would try to guess secrets at random. Indeed, modeling explicitly the probability of finding secrets requires an implicit assumption about what the attacker wants to find, e.g. that he is only interested in one specific secret or in all secrets. Focusing on the attacker advantage is more general, because we do not need to specify whether the attacker is interested in finding parts or all of the secrets.

Note that the GM also makes the implicit assumption that the advantage of the attacker is reduced to the probability of finding non-trivial collisions. This assumption incurs a loss of precision in the bounds one gives (since finding a non-trivial collision may not be sufficient to reveal all secrets); however, it allows to show that the advantage of the attacker is negligible for a sufficiently large order q of the group G and a reasonable number of steps t of the run.

2.3 The Random Oracle Model

Interactive generic algorithms are an extension of generic algorithms in which the attacker is able to interact with oracles through interactive steps. Such interactive algorithms can be modeled using the Random Oracle Model, or ROM for short, that was introduced by Bellare and Rogaway [5] but its idea originates from earlier work by Fiat and Shamir [9].

For the purpose of our work, we do not need to develop a general framework for interactions; instead we focus on two typical oracles with whom the attacker can interact: queries to hash functions and decryptors. These forms of interaction are used in particular in the signed ElGamal encryption protocol.

To sign a message, we do an interaction with a hash oracle i.e., a hash function $H : G \to M \to \mathbb{Z}_q$ where M is the set of messages. Cryptographic hash functions are used in various contexts, for example, to compute the message digest when making a digital signature. A hash function compresses the bits of a message to a fixed-size hash value in a way that distributes the possible messages evenly among the possible hash values. A cryptographic hash function does this in a way that makes it extremely difficult to come up with a message that would hash to a particular hash value. The ROM assumes a random hash function and is a stronger assumption that assuming the hash function to be collision resistant; the fundamental assumption of ROM is that the hash function $H : G \to M \to \mathbb{Z}_q$ is chosen at random with uniform probability distribution over all functions of that type. Note that interactions provide the algorithm with values, and that, in this setting, mex-steps perform computations of the form $f_i = \prod_{1 \leq j \leq t'} g^{l_j a_{i,j}}$, where for $1 \leq j \leq t'$, g^{l_j} is an input of the algorithm, and where $a_{i,1}, \ldots, a_{i,t'}$ are arbitrary but may depend on values that the algorithm received through interactions; for $1 \leq i \leq t''$, f_i is a group output of the algorithm so we assume it to do t'' mex-steps. Like in the non interactive case, we consider that the interactive generic adversary takes a list of secrets $s_1, \ldots, s_k$ and a list of inputs $g^{l_1}, \ldots, g^{l_{t'}}$.

Example 1. Let $x \in \mathbb{Z}_q$ and $h = g^x$ be the private and public keys for encryption, $m \in G$ the message to be encrypted. For encryption, pick random $r \in \mathbb{Z}_q$, (g^r, mh^r) is the ElGamal ciphertext. To add Schnorr signatures, pick random $s \in \mathbb{Z}_q$, compute $c = H(g^s, g^r, mh^r)$ and $z = s + cr$, then (g^r, mh^r, c, z) is the signed ciphertext. A decryptor *Dec* takes a claimed ciphertext $(\bar{h}, \bar{f}, c, z)$ and computes

$$F \;=\; (\mathit{if}\ H(g^z \bar{h}^{-c}, \bar{h}, \bar{f}) = c\ \mathit{then}\ \bar{h}^x\ \mathit{else}\ ?)$$

where ? is a random value, and then returns $\frac{\bar{f}}{F}$ which is the original message, if $(\bar{h}, \bar{f}, c, z)$ is a valid ciphertext.

A decryptor should not decrypt the target ciphertext because if the attacker sends to the decryptor the target ciphertext, the equality $H(g^z \bar{h}^{-c}, \bar{h}, \bar{f}) = c$ is always verified and so the attacker obtains immediately the original message.

As in the non-interactive model, an attacker is a generic algorithm that seeks to gain knowledge about secrets through testing equalities between the group elements it outputs, possibly through interactions. However, the attacker has now access to oracles for computing hash values and for decryption. Note that each operation performed by the attacker, i.e. reading an input, performing an interaction, or taking a mex-step, counts as a step in the run. However, as in the non-interactive case, testing equality is free. The adversary's advantage is the probability that the adversary finds non-trivial collisions among computed group elements plus the probability that the adversary obtains information on secrets through an interaction with the decryptor.

In the remaining of this subsection, we explain more precisely how the attacker can get information about the secrets by an interaction with the decryptor. Each interaction with the decryptor yields a polynomial and we can obtain information on the secrets if we find the zero of this polynomial.

Recall that a Schnorr signature on a message m is a triple $(m, c, z) \in M \times \mathbb{Z}_q^2$ such that $H(g^z h^{-c}, m) = c$ and let (f_i, f_j, c, z) be the claimed ciphertext that $\mathcal{A}$ transmits to the decryptor. In the ROM, the equation $c = H(g^z h^{-c}, f_i, f_j)$ required for a valid signature, necessitates that $\mathcal{A}$ selects c from the given hash values $H(f_\sigma, f_i, f_j)$ for given group elements f_σ, f_i, f_j.The attacker gets $c = H(g^z h^{-c}, f_i, f_j)$ from the hash oracle and must compute z so that $g^z h^{-c} = f_\sigma$, i.e., it must compute $z = log_g(f_\sigma f_j^c)$. The computed z i.e., the element z used for an interaction with the decryptor (recall that a decryptor takes as input a quadruple (f_i, f_j, c, z)), does not depend on the secrets $s_1, \ldots, s_k$ whereas $z' = log_g(f_\sigma f_j^c) = log_g f_\sigma + c\ log_g f_i$, which denotes the value required for a signature, may depend on it.

The group steps of the interative generic algorithm refer to the given group elements $l_1, \ldots, l_{t'}$. The adversary computes $f_i := \prod_{1 \le j \le t'} g^{l_j a_{i,j}}$ for $i = 1, \ldots, t$ using exponents $a_{i,1}, \ldots, a_{i,t'} \in \mathbb{Z}_q$ that arbitrarily depend on values that the algorithm received through interactions but not on the secrets $s_1, \ldots, s_k$. Hence z' is of the form:

$$\begin{aligned} z' &= log_g(f_\sigma f_j^c) \\ &= < a_\sigma + c a_j, (l_1, \ldots, l_{t'}) > \end{aligned} \tag{1}$$

where $< ., . >$ is a scalar product i.e., $< (a_1, \ldots, a_n), (c_1, \ldots, c_n) >= \sum_{j=1}^{n} a_j c_j$.

Considering $s_1, \ldots, s_k$ as formal variables over $\mathbb{Z}_q$, z' is a polynomial in $\mathbb{Z}_q[s_1, \ldots, s_k]$ (as the inputs $l_1, \ldots, l_{t'}$ are polynomials in $\mathbb{Z}_q[s_1, \ldots, s_k]$). The random $c, s_1, \ldots, s_k$ are statistically independent of the coefficients $a_{\sigma,1}, \ldots, a_{\sigma,t'}$ and $a_{j,1}, \ldots, a_{j,t'}$. $\mathcal{A}$ can obtain information about the secrets by solving the equation $z' = z$ i.e., the polynomial z' must be equal to the computed group element z, so we must have $z' - z = 0$. Let us notice that the value required for a signature i.e., z' depends on the secrets, so we note $z'(s_1, \ldots, s_k)$ instead z' to make the difference with the value computed by the algorithm i.e., z which is a constant in $\mathbb{Z}_q[s_1, \ldots, s_k]$. The equation $z' - z = 0$ is seen as a polynomial equality $z'(s_1, \ldots, s_k) - z \equiv 0$ for the secrets $s_1, \ldots, s_k$. Each interaction with the decryptor provides a polynomial $z'(s_1, \ldots, s_k) - z$, thus after l interactions with the decryptor, we have a list of l polynomials $z'(s_1, \ldots, s_k) - z$, so we can obtain informations about the secrets if we can find a zero of a polynomial that belongs to this list. In fact, an interaction with the decryptor succeeds if the equation $z'(s_1, \ldots, s_k) - z = 0$ holds; and by applying Schwartz lemma (see Section 3.3) to the polynomial $z'(s_1, \ldots, s_k) - z$, we get a bound to the probability of finding the secrets $s_1, \ldots, s_k$. To conclude, we eliminate interactions with the decryptor by computing extractor for each interaction with the decryptor.

2.4 Applications of GM+ROM

We consider the application to the signed ElGamal encryption; let the attacker be given the generator g, the public key $h = g^x$, distinct messages m_0, m_1, a target signed ciphertext $cip_b = (g^r, m_b g^{rx}, c, z)$ corresponding to m_b for a random bit $b \in \{0, 1\}$ and oracles for the hash function H and for decryption. Then an interactive generic adversary using t generic steps including t'' operations on group elements and l interactions with the decryptor, can not predict b with a better probability than $\frac{1}{2} + \frac{2\binom{t}{2}}{q - 2\binom{t}{2}}$. The probability space consists of the random $x,\ b,\ H$ and the key of the encipherer r.

To prove this, we use Schwartz lemma (see Section 3.3), the bound of the probability of finding collisions among the computed group elements i.e., non trivial collisions occur with no better probability than $\frac{2\binom{t''}{2}}{q - 2\binom{t''}{2}}$ if we compute t'' group elements. For this bound, the probability space refers to the random $b,\ r,\ x$; and we use the bound of the probability of finding information on the plaintext by having l interactions with the decryptor: $\frac{3l}{q}$.

3 Remarks on the formalization of algebra

This section provides a brief discussion on some issues with the formalization of the mathematical concepts that underlie the generic and random oracle models. We focus on two particularly important issues, namely the formalization of discrete probabilities and of multivariate polynomials.

3.1 Sets and algebra

Standard presentations of the generic model involve a cyclic group G of order q, and formalizing the generic model directly in Coq would therefore require that we use a formalization of groups. Although several such formalizations are already provided by the Coq contributions, we find it convenient to exploit the canonical isomorphism between the group G and the ring $\mathbb{Z}_q$ (assuming that g is a generator of the group, the function $x \mapsto g^x$ is an isomorphism from $\mathbb{Z}_q$ to G) and work directly with $\mathbb{Z}_q$ instead of G. Thus, instead of considering an element g^a of the group, we will always consider the exponent a, and instead of considering multivariate exponentiation $mex : \mathbb{Z}_q^d \to G^d \to G$ as done in the generic model $(a_1, \ldots, a_d), (g_1, \ldots, g_d) \mapsto \prod_{i=1}^{d} g_i^{a_i}$ we use the function $mex : \mathbb{Z}_q^d \to \mathbb{Z}_q^d \to \mathbb{Z}_q$ defined as $(a_1, \ldots, a_d), (s_1, \ldots, s_d) \mapsto \sum_{j=1}^{d} a_j s_j$ where it is assumed that $g_i = g^{s_i}$.

Dispensing with the formalization of groups does not allow us to dispense with the formalization of sets, which are represented using setoids [3, 2]. For the sake of readability, we avoid in as much as possible mentioning setoids in our presentation, although they are pervasive in our formalizations.

3.2 Probabilities

There are several possible formalizations of probabilities in Coq. Our choice of formalization was influenced by two important factors. The first factor is a simplifying factor, namely for our purposes we only need to consider discrete probabilities, i.e. probabilities over finite sets. The second factor is a complicating factor, namely for our purposes we need to consider probabilities over setoids.

On this basis, we have found convenient to define probabilities w.r.t. an arbitrary type V and a finite subset E of V, given as a (non-repeating and non-empty) V-list; intuitively, E is meant to contain exactly one representative from each equivalence class generated by the setoid underlying equality. Then the probability of an event A, i.e. of a predicate over V, is defined as the ratio between the number of elements in E for which A holds and the total number of elements in E, i.e.

```
Definition Event:=V → Prop
Definition Pr_E(A:Event):=length (filter E A)/(length E).
```

One can check that Pr_E satisfies the properties of a probability measure, and define derived notions such as conditional probabilities. In the sequel, E will often be omitted to adopt the notation $Pr(A)$.

3.3 Polynomials

There have been several formalizations of polynomials in Coq. Our choice of formalization was guided by the proof of Schwartz Lemma (see below), which requires to view a polynomial in $n + 1$ variables as a polynomial in n or 1 variables. Our current formalization (which is different from the formalization

used in [4] and in our opinion more elegant) uses the Horner representation of polynomials for polynomials in one variable, and define polynomials in $n+1$ variables recursively from polynomials in n variables. Formally, we consider a ring R with underlying carrier C and define $R[X]$ as the inductive type:

```
Inductive R[X]:Type:=
 |Pc : C→ R[X]
 |PX : R[X] → C → R[X].
```

where Pc is the constructor for constant polynomials and `PX Q d =Q*X+d`. Once monovariate polynomials have been formalized, polynomials in $n+1$ variables are built recursively from polynomials in n-variables, using the canonical isomorphism between $R[X_1,\ldots,X_{n+1}]$ and $(R[X_1,\ldots,X_n])[X_{n+1}]$.

Then one can define equality $\equiv$ between polynomials using an appropriate inductive relation, and endow $R[X]$ and $R[X_1,\ldots,X_{n+1}]$ with a ring structure under the usual operations of polynomial addition, substraction, multiplication, etc. Using this formalization, we have proved the following lemma, which provides an upper bound on the probability of a vector to be a root of a polynomial of degree d over $\mathbb{Z}_q[X_1,\ldots,X_n]$, and which is the key result that enables security proofs in the Generic Model.

Lemma 1 (Schwartz Lemma).

$\forall$ `(p :` $\mathbb{Z}_q[X_1,\ldots,X_n]$`), q` $\neq$ `0` $\rightarrow$ `p` $\not\equiv$ `0` $\rightarrow$
$Pr_{x_1,\ldots,x_n\in\mathbb{Z}_q^n}$`(`$p(x_1,\ldots,x_n)$`= 0)` $\leq$ `(degree p)/q.`

The lemma requires that q is not null and that p is not identically null, and establishes that the probability that an element $x \in \mathbb{Z}_q^n$ is a zero of a polynomial p is smaller than the degree of the polynomial divided by q. The proof proceeds by induction on the number n of variables. Note that we slightly abuse notation and write $Pr_{x_1,\ldots,x_n\in\mathbb{Z}_q^n}$`(` $p(x_1,\ldots,x_n)$`= 0)` for $Pr_{Var\rightarrow\mathbb{Z}_q}$`(`$\lambda$`f:Var`$\rightarrow\mathbb{Z}_q$`.(p f) = 0)`, where Var is the finite set with inhabitants $X_1,\ldots,X_n$.

We now state corollaries of Schwartz lemma that are used in Section 5. The first corollary follows rather directly from Schwartz lemma, while the second corollary is proved by induction.

Lemma 2. $\forall$ `(d:`$\mathbb{N}$`)(` $p_1,\ldots,p_n$ `:` $\mathbb{Z}_q[X_1,\ldots,X_n]$`), q` $\neq$ `0` $\rightarrow$
`(`$\forall$ `1`$\leq$`j`$\leq$`n, (degree` p_j`)`$\leq$`d` $\wedge$ $p_j \not\equiv$ `0)` $\rightarrow$
$Pr_{x_1,\ldots,x_n\in\mathbb{Z}_q^n}$`(`$p_n(x_1,\ldots,x_n)$`=0|` $\forall$ `j<n,` $p_j(x_1,\ldots,x_n)\neq$`0)`$\leq$`d/(q-nd).`

Lemma 3. $\forall$ `(d:`$\mathbb{N}$`)(` $p_1,\ldots,p_n$ `:` $\mathbb{Z}_q[X_1,\ldots,X_n]$`), q` $\neq$ `0` $\rightarrow$
`(`$\forall$ `1`$\leq$`j`$\leq$`n, (degree` p_j`)`$\leq$`d` $\wedge$ $p_j \not\equiv$ `0)` $\rightarrow$ `nd<q` $\rightarrow$
$Pr_{x_1,\ldots,x_n\in\mathbb{Z}_q^n}$`(`$p_1(x_1,\ldots,x_n)$ `= 0` $\vee\ldots\vee$ $p_n(x_1,\ldots,x_n)$ `= 0)`$\leq$`nd/(q-nd).`

4 A review of the formalization of the Generic Model

The main difficulty in formalizing generic algorithms is to pinpoint the notion of secret. As mentioned in the introduction, we take advantage of the expressiveness of our framework and introduce an abstract type *Sec* of secrets together with an interpretation function from *Sec* to $\mathbb{Z}_q$. In order to fix terminology, we refer to elements of *Sec* as symbolic secrets and to their interpretation in $\mathbb{Z}_q$ as (concrete values of) secrets. Then, we define generic algorithms which describe the behavior of the attacker at an abstract level; being defined at an abstract level, the behavior of the attacker is independent of the concrete values of the secret, as required by the GM.

Generic algorithms may be executed symbolically, producing symbolic outputs in the form of polynomials. Further, abstract runs are interpreted into concrete runs; the latter correspond to executing an attack and may or not be successful, depending on whether a non-trivial collision is found.

```
1  Parameter Sec:Set.
2  Parameter input:list Z_q[Sec].
3
4  Inductive GA:Type:=
5    nostep:GA
6   |step:GA→(list Z_q)→GA.
7
8  Fixpoint SymbOutput(A:GA):(list Z_q[Sec]):=
9    match A with nostep ⇒ nil
10       | (step A' e) ⇒(mex e input)::(SymbOutput A')
11   end.
12
13 Definition ConcrOutput(A:GA)(σ:Sec→Z_q):list Z_q:=
14 map λx:Z_q[Sec].[|x|]_σ  (SymbolicOuput A).
15
16 Definition CO(A:GA)(σ:Sec →Z_q):=
17 ∀ e e':Z_q[Sec], e ∈ (SymbOutput A) ∧
18         e' ∈ (SymbOutput A) ∧ e-e'≠0 ∧ [|e − e|]_σ=0.
```

Fig. 1. Formalization of the GM

The formal definition of a generic algorithm is given in Figure 1. In order to model the notion of secrets, we introduce a type *Sec* of formal secret parameters (see line 1) and model inputs as a list of non-repeating polynomial expressions over secrets (see line 2); note that inputs are determined by the cryptographic system under consideration, and are known to the attacker. In the above we implicitly assume that the set `Sec` is modeled as a finite type with k secrets $s_1, \ldots, s_k$ and we use $\mathbb{Z}_q[Sec]$ as a shorthand for $\mathbb{Z}_q[s_1, \ldots, s_k]$.

Then, we consider generic algorithms (see line 4) in which the attacker selects arbitrarily and independently of the secrets a list of coefficients $a_{i,1}, \ldots, a_{i,t'} \in \mathbb{Z}_q$. Generic algorithms can be executed to produce symbolic outputs (see line 8), and concrete outputs (see line 13) are obtained from the symbolic outputs by using the extension of an interpretation function σ from polynomial expressions to elements in $\mathbb{Z}_q$, more precisely, $[\![\ ]\!]_\sigma : \mathbb{Z}_q[Sec] \to \mathbb{Z}_q$ returns the evaluation of a polynomial in $\mathbb{Z}_q[Sec]$ by using an interpretation function σ. An interpretation function $\sigma : Sec \to \mathbb{Z}_q$ maps formal secret parameters to actual secrets in $\mathbb{Z}_q$. Now we can define a non-trivial collision (see line 16) as a pair of polynomials `e` and `e'` (found in `(SymbOutput r)`) that are non identically equal and such that the interpretation of the polynomial `e-e'` under σ is 0. By considering only polynomials non identically equal, we eliminate trivial collisions.

In order to give an upper bound for the probability of finding non-trivial collisions, one relies on Schwartz Lemma (see Section 3.3). In the sequel, we let d be the maximal degree of the inputs i.e., the polynomials l_j for $1 \leq j \leq t'$, let t be the number of steps $\mathcal{A}$ performs.

Proposition 1. $\forall \mathcal{A} : \mathtt{GA},\ Advantage(\mathcal{A}) = Pr(\mathcal{CO}(\mathcal{A})) \leq \dfrac{\binom{t}{2} d}{q - \binom{t}{2} d}$

In the non-interactive setting, we consider that the advantage of the attacker is bounded by the probability of finding non-trivial collisions. Such an over-approximation is quite coarse since we consider the attacker to be successful whenever he gains some informations about the interpretation function σ. In principle, one could try to be more precise and estimate the probability of the attacker to find the function σ (i.e. its value for all inputs). However, we only want to show that the advantage is negligible if the number of steps performed by the attacker is reasonable, and hence the over-approximation is justified.

In [4], we instantiate the proposition to specific cryptographic schemes.

5 Formalization of the Random Oracle Model

5.1 Formalization

The main difficulty in formalizing interactive algorithms is to capture the idea of random hash function. Following the idea of the generic model, we consider a symbolic representation of the interactions with the hash oracle by introducing a type *Val* of random variables that will represent the results of the interactions with the hash oracle. In addition, we define an interpretation function from *Val* to $\mathbb{Z}_q$. In order to fix terminology, we will refer to elements of *Val* as symbolic hash values and to their interpretation as hash results.

The formal definition of interactive generic algorithms is given in Figure3. As explained above, we introduce a type *Val* of symbolic hash results and a type *Sec* of symbolic secrets (see line 1). Then, we model inputs as a non-repeating list of polynomial expressions over secrets (see line 2).

In order to model ciphertexts, we introduce a type of symbolic group elements *SymbG*, defined as the type of $\mathbb{Z}_q$-lists (see line 4). Symbolic group elements are

intuitively assumed to have a length that matches the length of the list of inputs $(l_1, \ldots, l_{t'})$, and the list $(a_1, \ldots, a_{t'})$ represents the polynomial $\sum_{j=1}^{t'} a_j l_j$. In other words, symbolic group elements correspond to linear combinations of the inputs. Then we define symbolic ciphertexts as pairs of symbolic group elements, by analogy with ElGamal ciphertexts that have the form (g^r, mg^{rx}) where g^r and mg^{rx} are group elements (see line 5). Finally, symbolic hash queries are defined as triples of the form (g, m, v) where g is a group element, m is a message, and v is the symbolic hash result, by analogy with hash queries that have the form $H(a, (b, d))$.

Interactive generic algorithms are defined inductively (see line 8) and may consist of an empty step, or a mexstep i.e., a computation of group elements using the function mex, or an hashstep i.e., a query to the hash oracle, or a decstep i.e., an interaction with the decryptor. A few words are in order to explain the type of the constructor `decstep`: first of all, observe that in analogy with ElGamal decryptor that takes a claimed ciphertext $(\bar{h}, \bar{f}, c, z)$, our formalization considers that an interaction with the decryptor requires a symbolic hash query $I = (f_j, \bar{h}, \bar{f}, c)$ and an element z of $\mathbb{Z}_q$.

Interactive algorithms have two kinds of outputs:

- symbolic hash outputs (see line 14) are just a list of elements of type `SymbH`. Then we can derive the list of concrete hash outputs (see line 22) by applying an interpretation function τ;
- symbolic group outputs (see line 25) are polynomials constructed as linear combinations of inputs in the same way of non-interactive generic algorithms. Concrete group outputs (see line 33) are obtained from the symbolic outputs by using the extension of an interpretation function σ from secrets to elements in $\mathbb{Z}_q$.

In an interactive generic algorithm, the attacker might gain knowledge about secrets either through collisions, or through interactions with the decryptor. Thus its advantage will be bounded by the probability of finding a collision plus the probability of performing a successful interaction. In the latter case, we show that the attacker can only obtain information if the interpretation function is solution to a polynomial equation derived from the equality tested by the decryptor, i.e. $c = H(g^z \bar{h}^{-c}, \bar{h}, \bar{f})$, where $(\bar{h}, \bar{f}, c, z)$ is the claimed ciphertext received by the decryptor. Note that we do not need to formalize the result returned by the decryptor, which is random in case the claimed ciphertext does not verify the above equality, since we are only interested in the probability to learn information about secrets.

To eliminate interactions with the decryptor, we formalize an extractor (see Figure 2). Let us remenber that we can obtain information on the secrets by an interaction with the decryptor if $z'(s_1, \ldots, s_k) - z = 0$ holds, where $z'(s_1, \ldots, s_k)$ is the value required for a signature and z is the computed group element (see the explications in the section 2.3). More precisely, $\mathcal{A}$ gets the hash value $c = H(g^z h^{-c}, f_i, f_j)$ from the hash oracle and must compute z' so that $g^{z'} h^{-c} = f_\sigma$, i.e., it must compute $z' = log_g(f_\sigma f_j^c)$ (see line 1). For each `decstep`, $\mathcal{A}$ can obtain

```
1 Definition z'(h:SymbH)(τ:Val→ℤ_q):ℤ_q[Sec]:=
2 let h:=(f_σ, f_i, f_j, c) in
3 let  log_g f_σ :=(mk_pol f_σ input) in
4 let  log_g f_i :=(mk_pol f_i input) in
5  log_g f_σ +(τ c)* log_g f_i .
6
7 Definition Extract(h:SymbH)(z:ℤ_q)(τ:Val→ℤ_q):ℤ_q[Sec]:= (z' h τ)-z.
8
9 Fixpoint list_Extract( A :IGA)(τ:Val→ℤ_q):list ℤ_q[Sec]:=
10 match A with nostep ⇒ nil
11            | mexstep A' _ ⇒ list_Extract A' τ
12            | hashstep A' _ ⇒list_Extract A' τ
13            | decstep A' h z⇒(Extract h z τ) ::(list_Extract A' τ)
14   end.
15
16 Definition Extractor( A :IGA)(τ:Val→ℤ_q)(x:Sec→ℤ_q):Prop:=
17 ∀ p:ℤ_q[Sec], p ∈ (list_Extract A τ) ∧ [|p|]_σ=0.
```

Fig. 2. Formalizatiom of an Extractor

informations about the secrets $s_1, \ldots, s_k$ by finding a zero of the polynomial $z'(s_1, \ldots, s_k) - z$ (see line 7). After l interactions with the decryptor, we have a list of l polynomials $z'(s_1, \ldots, s_k) - z$ (see line 9). So we can find informations about the secrets (see line 16) if there exists a polynomial `p` in the list of l polynomials $z'(s_1, \ldots, s_k) - z$ such that its interpretation under the interpretation function σ is 0.

Let us notice that in our model, as we do not formalize the result of the decryptor but we consider an extractor that tries to find informations on secrets by finding the zeros of the polynomial $z'(s_1, \ldots, s_k) - z \equiv 0$, if the attacker sends the target ciphertext to the decryptor, in fact this leads to a trivial polynomial equality $z'(s_1, \ldots, s_k) \equiv z$ and so we do not obtain informations on secrets.

5.2 Properties of interactive generic algorithm

In this section, we prove the security of cryptographic protocols like ElGamal against a strong adaptive chosen ciphertexts attack by giving an upper bound of the probability for an interactive adversary to find information about secrets.

We consider an interactive generic algorithm $\mathcal{A}$ with inputs polynomials $l_1, \ldots, l_{t'}$ with maximal degree d. Furthermore we assume that $\mathcal{A}$ performs t generic steps including t'' mex-steps and l interactions with the decryptor.

Proposition 2. $\forall \mathcal{A} : IGA,\ Pr(\mathcal{CO}(\mathcal{A})) \leq \frac{\binom{t''}{2} d}{q - \binom{t''}{2} d}$

Proof. All outputs are of the form $p_i = \sum_{1 \leq j \leq t'} a_{i,j}\ l_j(s_1, \ldots, s_k)$, where p_i is a polynomial of degree d. Hence there exists a collision $f_i = f_{i'}$ iff $(s_1, \ldots, s_k)$

is a root of $p_i - p_{i'}$. There are $\binom{t''}{2}$ equalities of the form $f_i = f_{i'}$ to test, hence $\binom{t''}{2}$ polynomials of the form $p_i - p_{i'}$, each of which is not identical to 0 (as there are non-trivial collisions), and has degree $\leq d$. So we can apply Lemma 3 (the extension of Schwartz Lemma) to deduce the expected result.

Proposition 3. $\forall \mathcal{A} : IGA,\ Pr(Extractor(\mathcal{A})) \leq \frac{(d+1)l}{q}$

Proof. The proof is by induction of the interactive generic algorithm $\mathcal{A}$. The only interesting case is when the algorithm interacts with the decryptor. In this case, the interaction is successful iff τ is a solution of the extracted polynomial, which is of degree d.

In the interactive setting, we consider that the advantage of the attacker is bounded by the probability of finding non-trivial collisions plus the probability of finding a zero of a polynomial resulting on an interaction with the decryptor.

Proposition 4. $\forall \mathcal{A} : IGA,\ Advantage(\mathcal{A}) = Pr(\mathcal{CO}(\mathcal{A})) + Pr(Extractor(\mathcal{A}))$
$\leq \frac{\binom{t''}{2}d}{q-\binom{t''}{2}d} + \frac{(d+1)l}{q}$

5.3 Application to signed ElGamal encryption

We can instantiate the propositions to specific cryptographic schemes. For example, we prove the security of signed ElGamal encryption against a strong adaptive chosen ciphertexts attack. We consider ElGamal protocol, be given in input: the generator g, the public key h, distinct messages $m_0,\ m_1$, a target ciphertext cip_b corresponding to m_b for a random bit $b \in 0, 1$ i.e., the concrete inputs are the list $(g, g^x, g^r, m_b g^{rx})$, so the formal inputs are the logarithm of the elements of the list of concrete inputs, i.e the list $(1, x, r, log_g m_b + rx)$. Therefore, the secrets are $b \in \{0, 1\}$ and $r, x \in \mathbb{Z}_q$. By consequent, the maximal degree of the inputs is $d := 2$. In this example, the advantage of the attacker is bounded by $\frac{2\binom{t}{2}}{q-2\binom{t}{2}}$ because $Pr(\mathcal{CO}(\mathcal{A})) \leq \frac{2\binom{t''}{2}}{q-2\binom{t''}{2}}$ and $Pr(Extr(\mathcal{A})) \leq \frac{3l}{q}$.

5.4 Remarks on the formalization

The results of this paper have been formalized and proved in Coq. The formalization is available from:

`http://www-sop.inria.everest/proofs/acces/`

The formalizations, proofs and applications represent 17,515 lines of code which are split as follows:

- discrete probabilities: 4,077 lines of code. This includes the lots of useful lemmas e.g. for rewriting probabilities;
- polynomials: 4,196 lines of code. This includes the representation of polynomials and the proof that they form a ring;

– basic libraries: 4,545 lines of code. This includes libraries for lists over type, $\mathbb{Z}_q$, etc.
– GM+ROM: 6,253 lines of code. This includes the formalization of GM and ROM, a proof of Schwartz lemma, its extensions, and its applications to GM and ROM.

  Our development is modular, and can be instantiated to other cryptosystems based on cyclic groups. To prove the security of such systems, we just have to give the list of inputs, a generator and a set of secrets, as illustrated by the instantiation of our results to ElGamal.

```
1  Parameter Sec Val:Set.
2  Parameter input:list ℤ_q[Sec].
3
4  Definition SymbG:=list ℤ_q.
5  Definition SymbM:=SymbG * SymbG.
6  Definition SymbH:=SymbG * SymbM * Val.
7
8  Inductive IGA : Type :=
9    | nostep : IGA
10   | mexstep :  IGA → (list ℤ_q) →  IGA
11   | hashstep: IGA → SymbH → IGA
12   | decstep: IGA →  SymbH→ℤ_q → IGA.
13
14 Fixpoint SymbHashOutput(𝒜:IGA):(list SymbH):=
15   match 𝒜 with
16   | nostep ⇒nil
17   | mexstep 𝒜' e ⇒ SymbHashOutput 𝒜'
18   | hashstep 𝒜' h ⇒ h::(SymbHashOutput 𝒜')
19   | decstep 𝒜' _ ⇒ SymbHashOutput 𝒜'
20 end.
21
22 Definition ConcrHashOutput(𝒜: IGA)(τ:Val→ℤ_q):(list ℤ_q):=
23 map λ(x,y,z,t).τ t (SymbHashOutput 𝒜).
24
25 Fixpoint SymbMexOutput(𝒜:IGA): list ℤ_q[Sec]:=
26   match 𝒜 with
27   | nostep ⇒nil _
28   | mexstep 𝒜' e ⇒ (mex e input)::(SymbMexOutput 𝒜')
29   | hashstep 𝒜' _ ⇒ SymbMexOutput 𝒜'
30   | decstep 𝒜' _ ⇒ SymbMexOutput 𝒜'
31 end.
32
33 Definition ConcrMexOutput(𝒜: IGA)(σ:Sec→ℤ_q):(list ℤ_q):=
34 map λx:ℤ_q[Sec].[|x|]_σ (SymbMexOutput 𝒜)).
```

Fig. 3. Formalizatiom of ROM

6 Conclusion

Provable cryptography aims at establishing rigorous security proofs for cryptographic schemes and appeals to involved complexity-theoretic arguments to show that the advantage of an attacker (over an attacker that proceeds by brute force) is negligible. Whereas provable cryptography provides an overall guarantee of the correctness of cryptographic schemes, it is not unusual for security proofs to contain glitches or to rely on hidden assumptions which open the way for attacks. In this perspective, it is very important to provide machine-checked proofs of the main results in provable cryptography. Dually, formalizing provable cryptography is interesting from the perspective of machine-checked mathematics because it relies on many mathematical theories of interest, including discrete probabilities, polynomials, and linear algebra.

In this paper, we have extended our previous machine-checked account of the GM to include the ROM and to establish security bounds for interactive algorithms. In another related work [19], we provide a machine-checked treatment of signature forgery attacks, as reported in [14]. These results generalize previous work, and give more rigorous bounds than those present in the literature. Nevertheless, machine-checked proofs of provable cryptography has barely been scratched, and much work remains to be done. For example, we would like to exploit our formalization to prove the security of realistic protocols, following e.g. [6, 17], and eventually perhaps to provide a machine-checked account of a formalism that integrates the computational view of cryptography, and provable cryptography.

Acknowledgments We are grateful to the anonymous referees for their constructive and detailed comments.

References

1. M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *Transactions on Software Engineering*, 22(1):6–15, January 1996.
2. H. Barendregt and H. Geuvers. Proof assistants using dependent type systems. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 18, pages 1149–1238. Elsevier Publishing, 2001.
3. G. Barthe, V. Capretta, and O. Pons. Setoids in type theory. *Journal of Functional Programming*, 13:261–293, March 2003.
4. G. Barthe, J. Cederquist, and S. Tarento. A Machine-Checked Formalization of the Generic Model and the Random Oracle Model. In D. Basin and M. Rusinowitch, editors, *Proceedings of IJCAR'04*, volume 3097 of *Lecture Notes in Computer Science*, pages 385–399, 2004.
5. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, November 1993.
6. D. Brown. Generic Groups, Collision Resistance, and ECDSA, 2002. Available from `http://eprint.iacr.org/2002/026/`.

7. Coq Development Team. *The Coq Proof Assistant User's Guide. Version 8.0*, January 2004.
8. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
9. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proc. CRYPTO'86*, volume 286 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1986.
10. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
11. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.
12. Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 433–444, London, UK, 1992. Springer-Verlag.
13. Bruce Schneier. *Applied Cryptography (Second Edition)*. John Wiley & Sons, 1996.
14. C.-P. Schnorr. Security of Blind Discrete Log Signatures against Interactive Attacks. In S. Qing, T. Okamoto, and J. Zhou, editors, *Proceedings of ICICS'01*, volume 2229 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2001.
15. C.-P. Schnorr and M. Jakobsson. Security of Signed ElGamal Encryption. In T. Okamoto, editor, *Proceedings of ASIACRYPT'00*, volume 1976 of *Lecture Notes in Computer Science*, pages 73–89. Springer-Verlag, 2000.
16. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Proceedings of EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1997.
17. N. Smart. The Exact Security of ECIES in the Generic Group Model. In B. Honary, editor, *Cryptography and Coding*, pages 73–84. Springer-Verlag, 2001.
18. J. Stern. Why provable security matters? In E. Biham, editor, *Proceedings of EUROCRYPT'03*, volume 2656 of *Lecture Notes in Computer Science*, pages 449–461. Springer-Verlag, 2003.
19. S. Tarento. Machine-checked security proofs of cryptographic signature sch emes. In *Proceedings of ESORICS'05*, volume 3xxx of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.