A Machine-Checked Formalization of the Generic Model and the Random Oracle Model

Gilles Barthe[1], Jan Cederquist[2]⋆, and Sabrina Tarento[1]⋆⋆

[1] INRIA Sophia-Antipolis, France
{Gilles.Barthe,Sabrina.Tarento}@sophia.inria.fr
[2] CWI, Amsterdam, The Netherlands
Jan.Cederquist@cwi.nl

Abstract. Most approaches to the formal analyses of cryptographic protocols make the perfect cryptography assumption, i.e. the hypothese that there is no way to obtain knowledge about the plaintext pertaining to a ciphertext without knowing the key. Ideally, one would prefer to rely on a weaker hypothesis on the computational cost of gaining information about the plaintext pertaining to a ciphertext without knowing the key. Such a view is permitted by the Generic Model and the Random Oracle Model which provide non-standard computational models in which one may reason about the computational cost of breaking a cryptographic scheme. Using the proof assistant Coq, we provide a machine-checked account of the Generic Model and the Random Oracle Model.

1 Introduction

Background. Cryptographic protocols are widely used in distributed systems as a means to authenticate data and principals and more generally as a fundamental mechanism to guarantee such security goals as the confidentiality and the integrity of sensitive data. Yet their design is particularly error prone [2], and serious flaws have been uncovered, even in relatively simple and carefully designed protocols such as the Needham-Schroeder protocol [20]. In light of the difficulty of achieving correct designs, numerous frameworks have been used for modeling and analyzing cryptographic protocols formally, just to name a few: belief logics [10], type systems [1, 17], model checkers [20], proof assistants [8, 26], and frameworks that integrate several approaches [21]. Many of these frameworks have been used to good effect for discovering subtle flaws or for validating complex cryptographic protocols such as IKE [22] or SET [6]; due to space constraints, we refer to the recent article by Meadows [23] for a more detailed account of the history and applications of formal methods to cryptographic protocol analysis.

⋆ Supported by an ERCIM Fellowship. Most of this work was performed at INRIA Sophia-Antipolis.

⋆⋆ Part of this work was supported by the INRIA Color project "Acces: Algorithmes Cryptographiques Certifiés".

Although these approaches differ in their underlying formalisms, they share the perfect cryptography assumption, i.e. the hypothesis that there is no way to obtain knowledge about the plaintext pertaining to a ciphertext without knowing the key; see [12, 13] for some mild extensions of these models. Ideally, one would prefer to rely on a weaker hypothesis about the probability and computational cost of gaining information about the plaintext pertaining to a ciphertext without knowing the key. Such a view is closer to the prevailing view in cryptographic research, and there have been a number of recent works that advocate bridging the gap between the formal and computational views [3, 23]. Yet in contrast to the formal view of cryptography, there is little work on the machine-checked formalization of the computational view of cryptography.

Generic Model and Random Oracle Model. The Generic Model, or GM for short, and the Random Oracle Model, or ROM for short, provide non-standard computational models in which one may reason about the probability and computational cost of breaking a cryptographic scheme.

ROM was introduced by Bellare and Rogaway [7], but its idea originates from earlier work by Fiat and Shamir [16]. GM was introduced by Shoup [29] and Nechaev [24], and further studied e.g. by Jakobsson and Schnorr [27, 28], who also considered the combination of GM and ROM. The framework of GM and ROM is expressive enough to capture a variety of problems, such as the complexity of the discrete logarithm or the decisional Diffie-Hellman problem. More generally, GM and ROM provide an overall guarantee that a cryptographic scheme is not flawed [27, 28, 31], and have been used for establishing the security of many cryptographic schemes, including interactive schemes for which traditional security proofs do not operate [27, 28].[3]

The basic idea behind GM and ROM is to constrain the behavior of an attacker so that he can only launch attacks that do not exploit any specific weakness from the group or the hash function underlying the cryptographic scheme under scrutiny. More precisely, GM and ROM define a class of generic algorithms that operate on an ideal group G of prime order q, and using an ideal hash function H that behaves as an oracle providing for each query a random answer. Then, one considers the probability of an attacker, taken among generic algorithms, to gain information about some secret, e.g. a cyphertext or a key. As we assume an ideal group, the attacker cannot obtain knowledge from a specific representation of group elements, and information about the secret may only be gained through queries to the oracle, or through testing group equalities. The main results of GM and ROM are upper bounds to the probability for an attacker of finding the secret: these upper bounds, which depend on the order q of the group, and on the number t of computations performed by the attacker, show that the probability is negligible for a sufficiently large q and a reasonable t.

[3] On the other hand, there is an ongoing debate about the exact significance of results that build upon GM and ROM; part of the debate arises from signatures scheme which can be proven secure under ROM, but which cannot implemented in a secure way, see e.g. [11] and e.g. [15] for a similar result about GM. Yet there is a consensus about the interest of these models.

This paper reports on preliminary work to provide, within the proof assistant Coq [14], a machine-checked formalization of GM and ROM and of their applications. In particular, we use our model of GM to give a machine-checked proof of the intractability of the discrete logarithm [29] (we do not prove anything about interactive algorithms here).

The formalization of GM and ROM within a proof assistant poses a number of interesting challenges. In particular, these models appeal to involved mathematical concepts from probabilities, such as uniform probability distribution, or statistical independence which are notoriously hard to formalize in a proof assistant, and to advanced results on multivariate polynomials, such as Schwartz Lemma, see Lemma 1, which provides an upper bound to the probability for a vector $\boldsymbol{x}$ over a finite field of order q to be the root of a multivariate polynomial of degree d. Furthermore, proofs about GM and ROM are carried out on examples, rather than in the general case. In order to increase the applicability of our formalization, we thus had to generalize the results of [27–29]. In particular, our main results, i.e. Propositions 1 and 2, provide for an arbitrary non-interactive generic algorithm an upper bound to the probability of breaking the cryptographic scheme, whereas previous works only provide such results for specific algorithms, such as the Discrete Logarithm [29].

Finally, proofs about cryptographic schemes are carried out in a very informal style, which makes their formalization intricate. In particular, proofs about GM and ROM often ignore events that occur with a negligible probability, i.e. events whose probability tends to 0 when the size of the group tends to ∞. When formalizing such proofs, we thus had to make statements and proofs about GM and ROM more accurate; an alternative would have been to formalize the notion of negligible probability using limits, but proofs would have become more complex.

Contents of the paper. The remainder of the paper is organised as follows. Section 2 provides a brief account of the Coq proof assistant, and presents our formalization of probabilities and polynomials, which are required to prove our main results. Section 3 describes our formal model of GM, whereas Section 4 provides a brief account of our formal model of ROM. We conclude in Section 5.

Formal proofs. For the sake of readability, we do not use Coq notation and adopt instead an informal style to present our main definitions and results. Formal developments can be retrieved from

`http://www-sop.inria.fr/everest/soft/Acces`

2 Preliminaries in Coq

This section provides a brief overview of the proof assistant Coq, and discusses some of issues with the formalization of algebra. Further, it describes our formalization of probabilities and of multivariate polynomials.

2.1 Coq

Coq [14] is a general purpose proof assistant based the Calculus of Inductive Constructions, which extends the Calculus of Constructions with a hierarchy of universes and mechanisms for (co)inductive definitions.

Coq integrates a very rich specification language. For example, complex algebraic structures can be defined and manipulated as first-class objects; in our formalization we rely on the formalization of basic algebraic structures by L. Pottier, available in the Coq contributions. Likewise, complex data structures are definable as inductive types; for the purpose of this paper however, we make a limited use of inductive definitions and mostly use first-order parameterized datatypes such as the type $list_X$ of X-lists.

In order to reason about specifications, Coq also integrates (through to the Curry-Howard isomorphism) a higher-order predicate logic. One particularity of the underlying logic is to be intuitionistic, hence types need not have a decidable equality, and predicates need not be decidable. For the sake of readability, we gloss over this issue in our presentation, but our formalization addresses decidability by making appropriate assumptions in definitions and results.

Further, logical statements can be used in specifications, e.g. in order to form the "subset" of prime numbers as the type of pairs $\langle n, \phi \rangle$ where n is a natural number and ϕ is a proof that n is prime. There are, however, some limitations to the interaction between specifications and propositions. In particular, dependent type theories such as the Calculus of Inductive Constructions lack intensional constructs that allow the formation of subsets or quotients. In order to circumvent this problem, formalizations rely on *setoids* [5], that is mathematical structures packaging a carrier, the "set"; its equality, the "book equality"; and a proof component ensuring that the book equality is well-behaved. For the sake of readability, we avoid in as much as possible mentioning setoids in our presentation, although they are pervasive in our formalizations.

Finally, let us introduce some Coq notation: **Prop** denotes the universe of propositions, and **Type** denotes the universe of types.

2.2 The field $\mathbb{Z}_q$

Integers modulo q are modeled as a setoid: given $q \in \mathbb{N}$, we formalize $\mathbb{Z}_q$ as a setoid with underlying type $\mathbb{Z}$ (defined in the Coq libraries), and with underlying equivalence relation $\equiv\ q$ where $\equiv$ is defined as

$$\lambda q \in \mathbb{N}.\ \lambda a, b \in \mathbb{Z}.\ \exists k \in \mathbb{Z}.\ a - b = k \times q$$

We have shown that $\mathbb{Z}_q$ is a commutative field for q prime. All ring operations are defined in the obvious way. Interestingly, the multiplicative inverse $\cdot^{-1} : \mathbb{Z}_q \to \mathbb{Z}_q$ is defined as $\lambda x.\ (\mathsf{mod}\ x\ q)^{q-2}$ where $\mathsf{mod}\ x\ q$ is the remainder of the Euclidean division of x by q, and we have used Fermat's little theorem, available from the Coq contributions, to prove that $\forall a \in \mathbb{Z}_q.\ a \not\equiv 0\ [q] \to a^{-1} * a \equiv a * a^{-1} \equiv 1\ [q]$.

2.3 Probabilities

As there is no appropriate library for probabilities in the reference libraries and contributions in Coq, we have developed a collection of basic definitions and results for probabilities over finite sets. Due to lack of space, we only provide the definition of probabilities and conditional probabilities, and the statement of one illustrative result.

Before delving into details, let us point out that there are several possible approaches for defining probabilities over finite setoids. One possibility is to assume that the setoid is finite, i.e. isomorphic to some initial segment of $\mathbb{N}$, for a suitable notion of isomorphism of setoids. We have found slightly more convenient to define probabilities w.r.t. an arbitrary type V and a finite subset E of V, given as a (non-repeating) V-list.

Given a fixed type V and a fixed enumeration $E : list_V$, we define an event to be a predicate over V, i.e. $Event : \ Type \ := V \rightarrow \mathbf{Prop}$. Then, we define the probability of an event A being true as the ratio between the number of elements in E for which A is true and the total number of elements in E, i.e.

$$Pr_E[A] = \frac{\mathsf{length}\ (\mathsf{filter}\ E\ A)}{\mathsf{length}\ E}$$

where length and filter are the usual functions on lists, i.e. $\mathsf{length}\ l$ computes the length of the list l, and $\mathsf{filter}\ l\ P$ removes from the list l all its elements that do not satisfy the predicate P.

Then, one can check that Pr_E satisfies the properties of a probability measure, e.g.:

- for every event A, $0 \leq Pr_E[A] \leq 1$;
- if $True$ is the trivial proposition, which always holds, then $Pr_E[\lambda a. True] = 1$;
- for any sequence A_i of disjoint events $Pr_E[\bigcup_{1 \leq i \leq n} A_i] = \sum_{1 \leq i \leq n} Pr_E[A_i]$, where $\bigcup_{1 \leq i \leq n} A_i = \lambda a.\ A_1(a) \vee \cdots \vee A_n(a)$.

Conditional probabilities are defined in the usual way, i.e.

$$Pr_E[A|B] = \frac{Pr_E[\lambda a.(Aa) \wedge (Ba)]}{Pr_E[B]}$$

Then, one can check that Pr_E satisfies properties such as

$$Pr_E[A] = Pr_E[A|B]\ Pr[B] + Pr_E[A|\neg B]\ (1 - Pr[B])$$

In the sequel, E will often be omitted to adopt the notation $Pr[A]$.

Discussion It would be useful for a number of purposes, including the formalization of GM and ROM, to develop a comprehensive library about probabilities. In particular, we believe that a formal treatment of negligible events is required, if we want to continue with machine-checked accounts of computational cryptography. To this end, it will also be necessary to develop a theory of limits. In this respect, it will be useful to consider existing works on machine-checked probabilities in HOL [19] or Mizar [25].

2.4 Polynomials

We have formalized a collection of basic definitions and results for multivariate polynomials. Due to lack of space, we only provide the definitions of polynomials and focus on Schwartz Lemma and its corollaries, which are crucial to prove Propositions 1 and 2. As in [4, 18], we use a list-based representation of polynomials; other possibilities for defining polynomials are discussed at the end of the paragraph.

Definition Given a fixed ring R with underlying carrier R_{carr}, and a fixed set X of indeterminates, a monomial is a function that associates an exponent to each variable, so the type Mon_X of monomials is defined as $X \to \mathbb{N}$ (note that X is a type, not a setoid). Moreover, a polynomial is modeled as a list of coefficient-monomial pairs, so the type $Pol_{R,X}$ of polynomials is defined as $list_{(R_{carr} \times Mon_X)}$. The setoid structure of $Pol_{R,X}$ is defined using the book equality over R. Using the underlying operations of R, one can easily endow $Pol_{R,X}$ with a ring structure; in particular, addition of two polynomials is defined as list concatenation, negation of a polynomial is defined pointwise, and multiplication of polynomials is defined from multiplication of monomials using the map function.

The definition above allows an easy definition of the degree of a polynomial, which is needed in the proof of Schwartz Lemma below. To compute the degree of a polynomial, we must first assume that X is a finite type $\mathbb{X}_k$, defined inductively in the usual way (and with inhabitants $var_1, \ldots, var_k$), and define the degree of a monomial as the sum of the degrees of the variables, i.e. $deg_{mon} m = \sum_{1 \leq i \leq k} (m\ var_i)$ where the sum operation can be defined using the elimination principle for finite sets. Then, the degree of a polynomial is defined as the maximum degree of its monomials, i.e.

$$deg_{pol} p = \mathsf{max}\ (\mathsf{map}\ (\lambda \langle a, m \rangle.\ deg_{mon}\ m)\ p)$$

where max is the function that computes the maximum element of a $\mathbb{Z}_q$-list.

Interpretation The possibility of evaluating polynomials from $Pol_{R,X}$ into R is crucial to our approach to modeling GM and ROM. In this paragraph, we briefly describe how to evaluate polynomials.

Assume that X is a finite type $\mathbb{X}_k$, and let R be a ring with underlying carrier R_{carr}. Then we can define an evaluation function $Eval$ that, given an interpretation function $f : X \to R_{carr}$ and a polynomial $p : Pol_{R,X}$, returns an element of R_{carr} that corresponds to the value of p under the interpretation f. The evaluation function $Eval$ is defined by structural recursion over lists, i.e. $Eval\ f\ (\mathsf{nil}) = 0_R$, where 0_R is the neutral element w.r.t. addition, and

$$Eval\ f\ (\mathsf{cons}\ \langle a, m \rangle\ l) = (a \times_R (Eval_{Mon}\ f\ m)) +_R (Eval\ f\ l)$$

where $\times_R$ and $+_R$ respectively denote the ring multiplication and addition, and $Eval_{Mon} : (X \to R_{carr}) \to Mon_X \to R_{carr}$ is an auxiliary function that computes the interpretation of monomials, i.e.

$$Eval_{Mon}\ f\ (x_1^{l_1} \dots x_k^{l_k}) = (f\ x_1)^{l_1} \times_R \dots \times_R (f\ x_k)^{l_k}$$

Schwartz Lemma In order to prove Schwartz Lemma, we assume the fundamental theorem of algebra for integers.

Theorem 1. *Let p be a polynomial in $\mathbb{Z}_q$, in one variable, of degree n, not identical to 0. Then there are at most n roots of $p(x) = 0$.*

We have not proved this result yet. However, we do not expect any difficulty in performing the proof, which proof proceeds by induction over the degree, using the division algorithm in the inductive step[4].

Lemma 1 (Schwartz Lemma). *Let $p(x_1, \dots, x_k)$ be a polynomial in k variables, not identical to 0, with degree at most d, and the values chosen uniformly and independently in $[0, q-1]$. Then $Pr[p(x_1, \dots, x_k) = 0] \leq d/q$.*

Proof (sketch). By induction on k. For $k = 0$ we just note that p is constant distinct from 0, thus the probability is 0. For the induction step, we rewrite $p(x_1, \dots, x_{k+1})$ according to the powers of x_{k+1}

$$p(x_1, \dots, x_{k+1}) = p'(x_1, \dots, x_{k+1}) + p_l(x_1, \dots, x_k)x_{k+1}^l$$

where $l \leq d$ is the degree of x_{k+1} in p, and p' is a polynomial in which the degree of x_{k+1} is less than l and p_l is a polynomial in $x_1, \dots, x_k$ not identical to 0.

In our formalization, we consider as type of events the type $(\mathbb{X}_k \to \mathbb{Z}) \to \mathbf{Prop}$, and as enumeration list the non-repeating list of all functions $g : \mathbb{X}_k \to \mathbb{Z}$ s.t. $0 \leq g\ x \leq q-1$ for all $x : \mathbb{X}_k$. Note that functions are treated up to extensional equality, i.e. for every function $g : \mathbb{X}_k \to \mathbb{Z}$ s.t. $0 \leq g\ x \leq q-1$ for all $x : \mathbb{X}_k$, there exists $g' : \mathbb{X}_k \to \mathbb{Z}$ in V s.t. $g\ x = g'\ x$ for all $x : \mathbb{X}_k$ (which does not imply that $g = g'$ in Coq). Using this approach, then the interpretation of variables in X becomes random and uniformly distributed. This observation is crucial for the validity of our models.

We now state some corollaries of Schwartz Lemma that are used in Section 3. The first corollary follows rather directly from Schwartz Lemma, while the second corollary is proved by induction.

Lemma 2. *Let $p_1, \dots, p_n$ be a sequence of polynomials, not identical to 0, with degree at most d, and the values are chosen uniformly and independently in $[0, q-1]$. Then $Pr[p_n = 0 | \forall j < n. p_j \neq 0] \leq \frac{d}{q-nd}$.*

Lemma 3. *Let $p_1, \dots, p_n$ be a sequence of polynomials, not identical to 0, with degree at most d, values of the variables chosen uniformly and independently in $[0, q-1]$ and $nd < q$. Then $Pr[p_1 = 0 \vee \dots \vee p_n = 0] \leq \frac{nd}{q-nd}$.*

[4] In fact, the result has been proved already by Geuvers et al. in their formalization of FTA and is available in the Coq contributions. However, it may not be straightforward to use their result in our work; indeed, the setting is slightly different, e.g. the underlying notion of set is provided by constructive setoids that feature an apartness relation, and algebraic structures are not formalized in exactly the same way.

Discussion There are many possible definitions of polynomials. To cite a few that have been used in Coq developments, Geuvers *et al.*'s formalization of FTA use the Hörner representation for polynomials in one variable, whereas Théry's formalization of Buchberger's algorithm uses an order on the monomials in order to avoid repeated entries of monomials, and Pottier's formalization of algebraic structures uses an inductive definition of polynomial expressions. In a very recent (unpublished) work, Grégoire and Mahboubi have explored alternative representations of multivariate polynomials that allow for efficient reflexive tactics.

However, there lacks a standard and comprehensive library about polynomials. To provide a solid basis for further work about GM and ROM, it seems relevant to develop such a library, possibly taking advantage of the isomorphism between the different representations of polynomials.

3 Non-interactive generic algorithms

The framework of non-interactive generic algorithms is useful for establishing the security of the discrete logarithm problem, as well as several other related problems, most notably the Diffie-Hellman problem. This section presents our formalization of non-interactive generic algorithms.

3.1 Informal account

Let G be a cyclic group of prime order q with generator g. A generic algorithm $\mathcal{A}$ over G is given by:

- its input $l_1, \ldots, l_{t'} \in \mathbb{Z}_q$, which depends upon a set of secrets, typically secret keys, say $s_1, \ldots, s_k \in \mathbb{Z}_q$. In the sequel, we define the group input $f_1, \ldots, f_{t'} \in G$ of the algorithm by $f_k = g^{l_k}$;
- a run, i.e. a sequence of t steps. A step can either be an input step, or a multivariate exponentiation (mex) step. An input step reads some input from the group input; for simplicity, we assume that all inputs are read exactly once at the beginning, for $1 \leq i \leq t'$, the algorithm at step i reads f_i from the group input. For $t' < i \leq t$, we assume that the algorithm at step i takes a mex step, i.e. selects arbitrarily $a_1^i, \ldots, a_{t'}^i \in \mathbb{Z}_q$ and computes $f_i = \prod_{1 \leq j \leq t'} f_j^{a_j^i}$.

The output of the generic algorithm $\mathcal{A}$ is the list $f_1, \ldots, f_t$. Further, we define *collisions* to be equalities $f_j = f_{j'}$ with $1 \leq j < j' \leq t$, and say that a collision $f_j = f_{j'}$ is *trivial* if it holds with probability 1, i.e. if it holds for all choices of secret data. In the sequel, we write $\mathcal{CO}(\mathcal{A})$ if the algorithm $\mathcal{A}$ finds non-trivial collisions.

The generic model considers an attacker as a generic algorithm $\mathcal{A}$, which tries to find information about secrets through testing equalities between outputs, i.e. through non-trivial collisions (trivial collisions do not reveal any information about secrets), and expresses a random guess for secrets if it fails to find them by the first method. Hence, the probability of $\mathcal{A}$ finding the secret s_j is deduced from the probability $ProbColl(\mathcal{A})$ that the algorithm discovers a non-trivial collision, i.e. that $\mathcal{CO}(\mathcal{A})$ holds.

In order to give an upper bound for $ProbColl(\mathcal{A})$, the Generic Model relies on Schwartz Lemma. To this end, the Generic Model assumes that $\mathcal{A}$ is a generic algorithm whose group inputs f_j are of the form $g^{m_j(s_1,\ldots,s_k)}$ where $m_j(x_1,\ldots,x_k)$ is a multivariate monomial over the set $X = \{x_1,\ldots,x_k\}$ of secret parameters, and $s_1,\ldots,s_k$ are the actual secrets.

Example 1 (Discrete logarithm). The algorithm is given as input the group generator $g \in G$ and the public key $h = g^r \in G$, and outputs a guess y for $\log_g h = r$. Observe that any non-trivial collision reveals the value of r: indeed, every f_i will be of the form $g^{a_i}(g^r)^{a'_i} = g^{(a_i + ra'_i)}$. Hence for any collision $f_i = f_j$, we have $g^{(a_i+ra'_i)} = g^{(a_j+ra'_j)}$, and so $r(a'_i - a'_j) \equiv a_j - a_i\ [q]$. If the collision is non-trivial, then $a'_i - a'_j \neq 0$ and we can deduce the value of r.

In this example, there is a single secret r and the formal inputs are the monomials $1 = r^0$ and r.

Example 2 (Decisional Diffie-Hellman Problem). The algorithm is given as input the group generator $g \in G$, the group elements g^x and g^y, and the group elements g^{xy} and g^z in random order, where x, y, z are random in $\mathbb{Z}_q$, and outputs a guess for g^{xy} (or equivalently, for the order of g^{xy} and g^z).

In this example, there are three secrets x and y and z, and the formal inputs are the monomials 1 and x and y and xy and z.

3.2 Formal account

The main difficulty in formalizing generic algorithms is to capture formally the idea of a secret. We choose to model secrets by introducing a type *Sec* of formal secret parameters and an interpretation function $\sigma : Sec \to \mathbb{Z}_q$ that maps formal secret inputs to actual secrets.

Further, we assume given a non-repeating list of monomials $input : list_{mon_{Sec}}$ of length t', and let $m_1, \ldots, m_{t'}$ be the elements of *input*. These monomials constitute the formal inputs of the algorithm; the actual inputs can be defined as $\mathsf{map}\ (Eval_{mon}\ \sigma)\ input : list_{\mathbb{Z}_q}$.

Finally, the type of generic algorithms is defined as the record type

$$GA = \{run : list_{list_{\mathbb{Z}_q}};\ ok : \ldots\}$$

where *run* is the list of exponents selected by the algorithm at each step (the exponents are themselves gathered in a list), and *ok* is a predicate that guarantees some suitable properties on *run*, in particular that:

- all elements of *run* also have length t';
- for $1 \leq j \leq t'$, the j-th element of *run* is the list whose j-th element is 1, and whose remaining elements are 0;
- *run* is a non-repeating list, so as to avoid trivial collisions, see below.

The output of a generic algorithm is obtained by computing from the exponents $a^i_1, \ldots, a^i_{t'}$ the polynomial $p_i = \sum_{1 \le j \le t'} a^i_j \ m_j$, then evaluating each polynomial p_i with σ, finally obtaining in each case an element f_i of $\mathbb{Z}_q$ (as compared to the informal account, we find it more convenient to outputs as elements of $\mathbb{Z}_q$, which is legitimate since $\mathbb{Z}_q$ and G are isomorphic).

Formally, the output of the generic algorithm is modeled as

$$\mathit{output} : \mathit{list}_{\mathbb{Z}_q} := \mathsf{map}\ (\mathsf{eval\_pol}\ \sigma)\ (\mathsf{map}\ (\lambda l.\ \mathsf{zip}\ l\ \mathit{input})\ \mathit{run})$$

where zip is the obvious function of type $\forall A, B : \mathbf{Type}.\ \mathit{list}_A \to \mathit{list}_B \to \mathit{list}_{A \times B}$.

Then, $\mathcal{CO}(\mathcal{A})$ is defined as doubles *output*, where doubles is the boolean-valued function that checks whether there are repeated entries in a list. Note that collisions occur iff there exist pairwise distinct i and i' s.t.

$$\mathsf{eval\_pol}\ \sigma\ p_i =_{\mathbb{Z}_q} \mathsf{eval\_pol}\ \sigma\ p_{i'}$$

Furthermore, trivial collisions are defined to satisfy the stronger property

$$\forall I : \mathit{Sec} \to \mathbb{Z}_q.\ \mathsf{eval\_pol}\ I\ p_i =_{\mathbb{Z}_q} \mathsf{eval\_pol}\ I\ p_{i'}$$

Such collisions never occur in our setting, since we assume that *input* and *run* are non-repeating lists, and hence that $p_i - p_{i'}$ cannot be identical to 0.

Finally, a necessary condition *SecFound* for an algorithm $\mathcal{A}$ to find a secret is that, either there was a collision or, there were no collisions but the algorithm happens to guess the correct value. Formally, given an algorithm $\mathcal{A}$, a secret $x : \mathit{Sec}$, and $g : \mathbb{Z}_q$ expressing the guess of the algorithm in case no collision is found, we define the relation $\mathit{SecFound}_{\mathcal{A}}(g, x)$ by the clauses:

$$\frac{\mathit{Coll}(\mathcal{A})}{\mathit{SecFound}_{\mathcal{A}}(g, x)} \qquad\qquad \frac{\neg \mathit{Coll}(\mathcal{A}) \quad \sigma x =_{\mathbb{Z}_q} g}{\mathit{SecFound}_{\mathcal{A}}(g, x)}$$

3.3 Properties of generic algorithms

In this section, we let $\mathcal{A}$ be a generic algorithm. Further, we let d be the maximal degree of the monomials m_j for $1 \le j \le t'$, let t be the number of steps $\mathcal{A}$ performs.

Proposition 1. $\mathit{ProbColl}(\mathcal{A}) \le \frac{\binom{t}{2} d}{q - \binom{t}{2} d}$

Proof. All outputs are of the form $f_i = \sum_{1 \le j \le t'} a^i_j \ m_j(s_1, \ldots, s_k)$, where $p_i = \sum_{1 \le j \le t'} a^i_j \ m_j(x_1, \ldots, x_k)$ 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 $\le d$. So we can apply Lemma 3 to deduce the expected result.

In the sequel, we let $\mathit{ProbSecFound}_{\mathcal{A}}(x) = \mathit{Pr}[\lambda g.\ \mathit{SecFound}(g, x)]$.

Proposition 2. *Let $\mu = \frac{\binom{t}{2}d}{q-\binom{t}{2}d}$. For any secret $x : Sec$,*

$$ProbSecFound_{\mathcal{A}}(x) \leq \mu + \frac{1-\mu}{q}$$

Proof. Immediate from the definition of $SecFound(g, x)$.

We can now instantiate the proposition to specific cryptographic schemes.

Example 3 (Discrete Logarithm, continued). Here $d = 1$ so the probability of finding the secret is $\mu + \frac{1-\mu}{q}$, where $\mu = \frac{\binom{t}{2}}{q-\binom{t}{2}}$.

Note that Proposition 2 only holds for a secret $x : Sec$ ranging uniformly over $\mathbb{Z}_q$. For some problems however, such as the Decisional Diffie-Hellman problem below, x ranges uniformly over a subset of $\mathbb{Z}_q$. In this case, the probability of finding a secret is $\mu + \frac{1-\mu}{q'}$, where $\mu = \frac{\binom{t}{2}d}{q-\binom{t}{2}d}$ and q' is the cardinal of the set of possible values for $x : Sec$.

Example 4 (Decisional Diffie-Hellman problem, continued). Here $q' = 2$ and $d = 2$ so the probability of finding the secret is $\frac{1+\mu}{2}$, where $\mu = \frac{2\binom{t}{2}}{q-2\binom{t}{2}}$.

We conclude this section with some brief remarks:

– our estimates are higher than the ones given for example in [27, 28], since we must take negligible events into account;
– in our presentation, we have followed [27, 28] in that attackers that do not find non-trivial collisions provide a random guess without taking advantage of the computations that they have performed. However, in our formalisation we also consider more powerful attackers that express a random guess over the set of values that have not yet been considered by the run, as every step that does not induce a collision reduces the set of possible values for the secret. In any case, the probability of such attackers is only negligibly higher than our probabilities.
– for the sake of readability, formal inputs are taken to be monomials. However, in our formalisation we also consider more general forms of formal inputs, namely polynomials. The main difficulty introduced by this more general form of formal inputs is the possibility for trivial collisions to occur, and hence one must add a new conjunct in the definition of *ok* that ensures the absence of such collisions.

4 Formalization of interactive algorithms

In this section, we present the main steps towards a formalization of interactive generic algorithms. However, we have not proven any result about interactive generic algorithms at this stage.

4.1 Informal account

An interactive generic algorithm can read an input, or take a mex-step, or perform an interaction. We consider two common forms of interactions in cryptographic algorithms: queries to hash functions and decryptors. These forms of interaction are used in particular in the signed ElGamal encryption protocol. Note that interactions provide the algorithm with values, and that, in this setting, mex-steps select perform computations of the form $f_i = \prod_{1 \leq j \leq t'} f_j^{a_j^i}$, where for $1 \leq j \leq t'$, f_j is an input of the algorithm, and where $a_1^i, \ldots, a_{t'}^i$ are arbitrary but may depend on values that the algorithm received through interactions. More generally, values obtained from oracle interactions can be used in mex-steps as well as in future interactions.

Example 5. 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.

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.

Let us conclude this section by pointing that the fundamental assumption of ROM is that the hash function $H : G \to G \to G \to \mathbb{Z}_q$ is chosen at random with uniform probability distribution over all functions of that type. This assumption must of course be reflected in our formalization.

4.2 Formalization

The main difficulty in formalizing interactive algorithms is of course to capture the idea of hash function. In our formalization, we introduce a type *Val* of random variables, disjoint of *Sec*, for communication with oracles. These variables, once given a value through an interaction, can be used in mex-steps.

Formally, we assume given a type *Sec* of formal secret parameters and an interpretation function $\sigma : \mathit{Sec} \to \mathbb{Z}_q$ that maps formal secret inputs to actual secrets, as well as a type *Val* of random variables, disjoint of *Sec*, and a function $\tau : \mathit{Val} \to \mathbb{Z}_q$. Further, we assume given a non-repeating list of monomials $\mathit{input} : \mathit{list}_{\mathit{mon}_{\mathit{Sec}}}$ of length t', and let $m_1, \ldots, m_{t'}$ be the elements of *input*. These monomials constitute the formal inputs of the algorithm; the actual inputs can be defined as map $(\mathit{Eval}_{mon}\ \sigma)\ \mathit{input} : \mathit{list}_{\mathbb{Z}_q}$.

Then, the type of interactive generic algorithms is defined as the record type

$$\mathit{IGA} = \{\mathit{run} : \mathit{Run};\ \mathit{ok} : \ldots\}$$

where Run is defined inductively by the clauses

$$\frac{}{\mathit{erun} : \mathit{Run}} \qquad \frac{\begin{array}{l} r : \mathit{Run} \\ a : \mathit{list}_{\mathit{Poly}_{\mathbb{Z}_q,\mathit{Val}}} \end{array}}{(\mathit{step}\ r\ a) : \mathit{Run}}$$

$$\frac{\begin{array}{l} r : \mathit{Run} \\ a, b, d : \mathit{list}_{\mathit{Poly}_{\mathbb{Z}_q,\mathit{Val}}} \\ c : \mathit{Val} \end{array}}{(\mathit{HashQuery}\ r\ a\ b\ d\ c) : \mathit{Run}} \qquad \frac{\begin{array}{l} r : \mathit{Run} \\ m : \mathit{Val} \\ c, z : \mathit{Poly}_{\mathbb{Z}_q,\mathit{Val}} \\ \bar{h}, \bar{f} : \mathit{list}_{\mathit{Poly}_{\mathbb{Z}_q,\mathit{Val}}} \end{array}}{(\mathit{DecQuery}\ r\ m\ \bar{h}\ \bar{f}\ c\ z) : \mathit{Run}}$$

and ok is a predicate that guarantees some suitable properties on run.

Further, observe that hash queries implicitly define a hash "function" $H : \mathit{Poly}_{\mathbb{Z}_q,\mathit{Val}} \rightarrow \mathit{Poly}_{\mathbb{Z}_q,\mathit{Val}} \rightarrow \mathit{Poly}_{\mathbb{Z}_q,\mathit{Val}} \rightarrow \mathit{Val}$. Hence ok must contain a conjunct that guarantees that the resulting hash "function" is well-behaved, and in particular that it is indeed a function.

The length of the run is defined as the number of steps taken to type $r : \mathit{Run}$ and is defined by straightforward structural recursion. Further, the output of the run is obtained by computing from the exponents $a_1^i, \ldots, a_{t'}^i$ the polynomial $p_i = \sum_{1 \leq j \leq t'} a_j^i\ m_j$, then evaluating each polynomial p_i with σ, obtaining in each case an element q_i of $\mathit{Pol}_{\mathbb{Z}_q,\mathit{Val}}$, and then evaluating each polynomial q_i with τ, obtaining in each case an element f_i of $\mathbb{Z}_q$.

5 Conclusion

Using the proof assistant Coq, we have given a formal account of GM and ROM and of some of its applications; in the case of non-interactive generic algorithms, Propositions 1 and 2 generalize existing results to the case of an arbitrary generic algorithm.

Much work remains to be done to provide a more extensive machine-checked account of GM and ROM and its applications. In particular, we have formalized interactive algorithms but have not provided any formal proofs about them. The next natural step to take is to formalize the results about such algorithms. We are planning to prove the security of signed ElGamal encryption, by relying on the results of [27, 28]. A more ambitious objective would be to exploit our formalizations to prove the security of realistic protocols, following e.g. [9, 30]. An even more far-fetched goal would be to give a machine-checked account of a formalism that integrates the computational and formal views of cryptography.

Acknowledgments We are grateful to the anonymous referees for their constructive and detailed comments, and to Guillaume Dufay for his help and comments on an earlier version of the paper.

References

1. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theoretical Computer Science*, 298(3):387–415, April 2003.
2. M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *Transactions on Software Engineering*, 22(1):6–15, January 1996.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
4. A. Bailey. Representing Algebra in LEGO. Master's thesis, University of Edinburgh, 1993.
5. 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.
6. G. Bella, F. Massacci, and L.C. Paulson. Verifying the set registration protocols. *IEEE Journal on Selected Areas in Communications*, 21:77–87, 2003.
7. 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.
8. D. Bolignano. Towards a mechanization of cryptographic protocol verification. In O. Grumberg, editor, *Proceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 131–142. Springer-Verlag, 1997.
9. D. Brown. Generic groups, collision resistance, and ecdsa, 2002. Available from `http://eprint.iacr.org/2002/026/`.
10. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
11. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proceedings of STOC'98*, pages 209–218, New York, 1998. ACM Press.
12. Y. Chevalier, R. K˝sters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of LICS'03*, pages 261–270. IEEE Computer Society Press, 2003.
13. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of LICS'03*, pages 271–280. IEEE Computer Society Press, 2003.
14. Coq Development Team. *The Coq Proof Assistant User's Guide. Version 7.4*, February 2003.
15. A. W. Dent. Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model. In Y. Zheng, editor, *Proceedings of ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 100–109. Springer-Verlag, 2002.
16. 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.
17. A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of CSFW'01*, pages 145–159. IEEE Computer Society Press, 2001.

18. J. Harrison. *Theorem Proving with the Real Numbers*. Distinguished Dissertations. Springer-Verlag, 1998.
19. J. Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, 2002.
20. G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
21. C. Meadows. The nrl protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, February 1996.
22. C. Meadows. Analysis of the internet key exchange protocol using the NRL protocol analyzer. In *Proceedings of SOSP'99*, pages 216–233. IEEE Computer Society Press, 1999.
23. C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In V.I. Gorodetski, V.A. Skormin, and L.J. Popyack, editors, *Proceedings of MMMACNS*, volume 2052 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
24. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.
25. A. Nedzusiak. σ-fields and probability. *Journal of Formalized Mathematics*, 1, 1989.
26. L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, 1998.
27. 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.
28. 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.
29. 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.
30. 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.
31. 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.