Machine-Checked Security Proofs of Cryptographic Signature Schemes

Sabrina Tarento

INRIA Sophia-Antipolis, France
Sabrina.Tarento@sophia.inria.fr

Abstract. Formal methods have been extensively applied to the certification of cryptographic protocols. However, most of these works make 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. A model that does not require the perfect cryptography assumption is the generic model and the random oracle model. These models provide non-standard computational models in which one may reason about the computational cost of breaking a cryptographic scheme. Using the machine-checked account of the Generic Model and the Random Oracle Model formalized in Coq, we prove the safety of cryptosystems that depend on a cyclic group (like ElGamal cryptosystem), against interactive generic attacks and we prove the security of blind signatures against interactive attacks. To prove the last step, we use a generic parallel attack to create a forgery signature.

1 Introduction

Cryptographic protocols are designed to provide certain security guarantees between agents communicating in a hostile environment. Numerous application domains including distributed systems and web services used cryptographic schemes. However, designing secure cryptographic mechanisms is extremely difficult to achieve [1], the literature abounds of attacks against cryptosystems that were previously proven correct. Recently, a significant research effort has been directed at linking the formal and computational approaches. One of the first result is presented by Abadi and Rogaway [2]: they prove the computational soundness of formal encryption in the case of a passive attacker. Since then, many results [3,15,19,14] have been obtained. Efforts are also under way to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [16,12,17] and a second step, to encode them into proof tolls. Therefore, there has lastly been an increasing interest in provable security. A system is said to have provable security if its security requirements are stated formally in an adversarial model and there is a proof that these security requirements can be met provided that some well studied cryptographic primitives (such as RSA) are secure. While provable cryptography has become an important tool in the validation of cryptographic schemes, there are regular attacks against cryptographic schemes that were deemed sound using methods

S. De Capitani di Vimercati et al. (Eds.): ESORICS 2005, LNCS 3679, pp. 140–158, 2005.

from provable security. Formal proofs enable to detail assumptions so by using a proof assistant like Coq, we do not have implicit requirements.

The objective of our work, initiated in [5], is to use proof assistants for formalizing provable cryptography. There are two motivations for our work. From the point of view of cryptography, proof assistants provide an excellent tool to highlight hidden assumptions that permeate proofs in cryptography. Furthermore, proof assistants solve another (milder) shortcoming of cryptographic proofs, namely the imprecision on bounds for the attacker's advantage. From the point of view of formal mathematics, provable cryptography covers a range of concepts including algebraic structures, polynomials, matrices and probabilities.

Contribution. In earlier work [5], we established the security of cryptographic schemes against non-interactive attacks, using the Generic Model (or GM for short), which provides non-standard computational models for reasoning about the probability and computational cost of breaking a cryptographic scheme. In such a scenario, the attacker tries to launch an attack without any external help. However 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:

- a hash oracle: an interaction with the hash oracle is a query to a random hash function $H : G \times M \to \mathbb{Z}_q$ where G is an arbitrary group of prime order q and M is the set of all cyphertexts.
- a signature oracle (signer for short): an interaction with the signer provides to the attacker the signature of a message.

The main contribution of this paper is to extend our security proofs to such interactive attacks, building on a combination of the GM and of the Random Oracle Model (or ROM for short) that assuming the hash function to be collision resistant (collisions of random functions have negligibly small probability).

We consider scenario in which we focus on signature forgery attacks, where the attacker aims at forging a message that will appear as having been signed by another party. In order to prove the security of a cryptographic signature scheme, one must at least establish that it is resistant to signature forgery attacks, since schemes that are subject to forgery attacks cannot guarantee the identity of signers nor can they enforce non-repudiation. In order to establish the security of signature schemes against forgery attacks, we are led to consider the ROS problem:
Find an overdetermined, solvable system of linear equations modulo q with random inhomogeneities. Specifically, given a random function $F : \mathbb{Z}_q^l \to \mathbb{Z}_q$ and coefficients $a_{k,l} \in \mathbb{Z}_q$, find a solvable system of $l+1$ distinct equations (1) in the unknowns $c_1, \ldots, c_l$ over $\mathbb{Z}_q$:

$$a_{k,1}c_1 + \ldots + a_{k,l}c_l = F(a_{k,1}, \ldots, a_{k,l}) \quad \text{for } k = 1, \ldots, t. \tag{1}$$

As in our earlier work, we improve on pen-and-paper proofs in two aspects:

- pen-and-paper proofs about GM and ROM are carried out on examples, rather than in the general case. In contrast, our results deal with arbitrary interactive generic algorithms;
- pen-and-paper 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 ∞. In contrast, we take all events into account and we provide accurate bounds on the attacker's advantage.

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 provides a brief review of our formalization of GM. Section 4 describes parallel attacks and Section 5 provides the formalization of ROM in which the attacker makes interactions with the signer. We conclude in Section 6.

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

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* [4], 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. The declaration mechanism allows the user to specify his own basic objects. Declared objects play the role of axioms or parameters in mathematics. To define simple inductive type, we use the command **`Inductive`** and to define recursive functions, we use the command **`Fixpoint`** that allows to define inductive objects using a fixed point construction.

2.2 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 discrete probabilities i.e, probabilities over finite sets [11]. 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 discrete probabilities i.e, 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. The probability space is the finite set `E` where every base element has the same probability.

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 →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.

Definition. Pr_E`(L:Event):=length (filter E L)/(length E).`

where `length` and `filter` are the usual functions on lists, i.e. `(length l)` computes the length of the list `l`, and `(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`(λa.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.

Definition. `Pr_cond(L M:Event):=`Pr_E`(L ∧ M)/`Pr_E`(M).`

In the sequel, we denote `Pr_cond(L M)` by Pr_E`(L|M)`.
Then, one can check that Pr_E satisfies properties such as
Pr_E`(A) =`Pr_E`(A|B) `Pr_E`(B) + `Pr_E`(A|¬B) (1-`Pr_E`(B))`
In the sequel, `E` will often be omitted to adopt the notation `Pr(A)`.

2.3 Polynomials

For our work, we need to have a formalization of polynomials in which we can compute easily the degree of a polynomial in several variables; and in particular for proving Schwartz lemma, we need to have a formalization that allows us to view a polynomial in $n+1$ variables as a polynomial in `n` or `1` variables. We have extended the formalization of polynomial on one variable.

`Pol1` is the type of all polynomials in one variable in `C`.

```
Inductive Pol1:Type:=
 |Pc : C→ Pol1
 |PX : Pol1 → C → Pol1.
```

We define the equality of polynomials with inference rules.

```
Inductive ≡:Pol1 → Pol1 → Prop :=
  |Eq1_Pc_Pc : ∀ p q: C, p=q→(Pc p)≡(Pc q)
  |Eq1_Pc_PX :∀ p q: C, ∀ Q1:Pol1,
       p=q→Q1≡0→(Pc p)≡(PX Q1 q)
  |Eq1_PX_Pc :∀ p q: C, ∀ P1:Pol1,
       p=q→P1≡0→(PX P1 p)≡(Pc q)
  |Eq1_PX_PX :∀ p q:C, ∀ P1 Q1 :Pol1,
       p=q→P1≡Q1→(PX P1 p)≡(PX Q1 q).
```

where `Pc` is the constructor for constant polynomials and `PX P c=P*X+c`.

We formalize the set of coefficients as a ring with usual operations and properties; with this formalization of polynomials on one variable, we can recover the operations and properties of a ring for `C[X]`. By induction we extend it for several variables so a polynomial in `n` variables in `C` is of the type `(C[`X_1`]...[`X_{n-1}`])[`X_n`]`.

Having this formalization of polynomials, we can formalize and prove an useful lemma for our proofs on security.

Lemma Schwartz:
$\forall$ `(p :` $\mathbb{Z}_q[X_1, \ldots, X_n]$`), q ≠ 0 → p ≢ 0 →`
$Pr_{x_1,\ldots,x_n \in \mathbb{Z}_q^n}$`(p(`$x_1, \ldots, x_n$`)≡0) ≤ (degree p)/q.`

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`. Here, the ring is the set $\mathbb{Z}_q$ and we have `n` variables $X_1, \ldots, X_n$.

2.4 Remarks on Formalization of Group

In our work, we consider a cyclic group G of prime order q; as we have an isomorphism between the group G and the ring $\mathbb{Z}_q$ [13], assuming that g is the generator of the group, each element of the group are an exponentiation of the generator and the function:

$$\begin{aligned} G &\to \mathbb{Z}_q \\ g^a &\mapsto a \end{aligned}$$

is a one-to-one correspondance (more precisely an isomorphism).

Thus, instead of considering an element g^a of the group, we will always consider the exponent a.

For example, for the multivariate exponentiation, we do not use the function

$$\begin{aligned} mex : \mathbb{Z}_q^d \times G^d &\to G \\ (a_1, \ldots, a_d, g_1, \ldots, g_d) &\mapsto \prod_{i=1}^{d} g_i^{a_i} \end{aligned}$$

but by assuming $g_i = g^{s_i}$, we define the function

$$mex : \mathbb{Z}_q^d \to \mathbb{Z}_q^d \to \mathbb{Z}_q$$
$$(a_1, \ldots, a_d), (s_1, \ldots, s_d) \mapsto \sum_{j=1}^{d} a_j s_j.$$

3 A Review of the Generic Model

The generic model, GM for short, was introduced by Shoup [22] and Nechaev [18], and can be used to provide an overall guarantee that a cryptographic scheme is not flawed [20,21,24]. For example, the GM is useful for establishing the complexity of the discrete logarithm or the decisional Diffie-Hellman problem, which we describe below.

3.1 Informal Account

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 $s_1, \ldots, s_n \in \mathbb{Z}_q$, its list of inputs, say $l_1, \ldots, l_{t'} \in \mathbb{Z}_q$, which depends upon secrets, and a generic algorithm, which is a sequence of 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 $f_i = \prod_{j=1}^{t'} f_j^{a_{i,j}}$, where $f_j = g^{l_j}$ for $1 \leq j \leq t'$. The output of the generic algorithm is the list $f_1, \ldots, f_t$, from which the attacker will test for collisions, i.e. equalities $f_j = f_{j'}$ with $1 \leq j < j' \leq t$.

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 model 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). The 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 probability is negligible for a sufficiently large order q of the group G and a reasonable number of steps t of the generic algorithm.

3.2 Formalization

The main difficulty in formalizing generic algorithms is to pinpoint the notion of secret. The formal definition of a generic algorithm is given in the 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). If the set `Sec` has n secrets $s_1, \ldots, s_n$, $\mathbb{Z}_q[Sec] = \mathbb{Z}_q[s_1, \ldots, s_n]$. Then, we consider symbolic algorithms (see line 4) in which the attacker selects arbitrarily and independently of the secrets a list of coefficients $a_{i,1}, \ldots, a_{i,t'} \in \mathbb{Z}_q$. The function *mex* (see line 8) takes the exponents instead of the group elements and returns the logarithm of the results of the function *concretemex*. Symbolic outputs (see line 16) are polynomials constructed as linear combinations of inputs, i.e. of the form $a_{i,1}l_1 + \cdots + a_{i,t'}l_{t'}$ (which correspond to the logarithm of the mex-steps) and concrete outputs (see line 21) 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 $\mathbb{Z}_q$. We can define the set of non-trivial collision (see line 24), we can find a non-trivial collision if we can find two polynomials `e e' ∈(SymbOutput r)` non identically equal such that the interpretation of the polynomial `e-e'` under σ is 0. By considering only polynomials non identically equal, we eliminate trivial collisions.

The advantage of the attacker is 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).

In order to give an upper bound for the probability of finding non-trivial collisions, one can then rely on Schwartz Lemma (see the section 2), as usual with the generic model.
In the sequel, we write $\mathcal{CO}(\mathcal{A})$ if the attacker $\mathcal{A}$ finds non-trivial collisions. Furthermore, 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. $\forall \mathcal{A} : \mathtt{GA},\ \mathit{Advantage}(\mathcal{A}) = Pr(\mathcal{CO}(\mathcal{A})) \leq \dfrac{\binom{t}{2}d}{q - \binom{t}{2}d}$

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 an extension of Schwartz Lemma to deduce the expected result.

We can instantiate the proposition to specific cryptographic schemes.

```
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 mex(a:list ℤ_q)(e:list ℤ_q[Sec]):ℤ_q[Sec]:=
9   match a with nil⇒ 0
10           |b::bs ⇒
11     match e with nil ⇒ 0
12              |x::xs ⇒ x*b + (mex xs bs)
13     end
14   end.
15
16 Fixpoint SymbOutput( 𝒜 :GA):(list ℤ_q[Sec]):=
17   match 𝒜 with nostep ⇒ nil
18       | (step 𝒜' e) ⇒(mex e input)::(SymbOutput 𝒜')
19   end.
20
21 Definition ConcrOutput( 𝒜 :GA)(σ:Sec→ℤ_q):list ℤ_q:=
22 map λx:ℤ_q[Sec].[|x|]_σ  (SymbOuput 𝒜).
23
24 Definition 𝒞𝒪 ( 𝒜 :GA)(σ:Sec →ℤ_q):=
25 ∀ e e':ℤ_q[Sec], e ∈ (SymbOutput 𝒜) ∧
26         e' ∈ (SymbOutput 𝒜) ∧ e-e'≠0 ∧ [|e − e|]_σ=0.
```

Fig. 1. Formalization of the GM

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 polynomials $1 := log_g g$ and $r := log_g g^r$ as we take the exposant instead of the exponentiation, thus the maximal degree of an input is $d = 1$ so the probability of finding the secret is $\mu + \frac{1}{q(1-\mu)}$, where $\mu = \frac{\binom{t}{2}}{q-\binom{t}{2}}$.

Note that Proposition 1 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 $\mu + \frac{1}{q'(1-\mu)}$, 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 2 (Decisional Diffie-Hellman Problem [9]). 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, y and z, and the formal inputs are the polynomials $1 := log_g g$, $x := log_g g^x$, $y := log_g g^y$, $xy := log_g g^{xy}$ and $z := log_g g^z$, thus the maximal degree of an input is $d = 2$. Here $q' = 2$ so the probability of finding the secret is $\mu + \frac{1}{2(1-\mu)}$, where $\mu = \frac{2\binom{t}{2}}{q-2\binom{t}{2}}$.

4 The Random Oracle Model

Interactive generic algorithms are 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 [6] but its idea originates from earlier work by Fiat and Shamir [10].

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 signers.

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. Let G be a cyclic group of prime order q with generator g and H be an hash function, modelled as an oracle, that given an input (query) outputs a random number in the range of H. An interactive generic algorithm $\mathcal{A}$ 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 signers. These forms of interaction are used in particular in the signed ElGamal encryption protocol. $\mathcal{A}$ 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 $s_1, \ldots, s_n \in \mathbb{Z}_q$. 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 including t'' mex-steps, τ query to the hash oracle and l interactions with the signer. A step can either be an input step, a multivariate exponentiation (mex) step, a query to the hash function or a interaction with the signer. An input step reads some input from the group input. For $1 \leq i \leq t''$, we assume that the algorithm at step i takes a mex step, i.e. selects arbitrarily $a_{i,1}, \ldots, a_{i,t'} \in \mathbb{Z}_q$ and computes $f_i = \prod_{1 \leq j \leq t'} g^{l_j a_{i,j}}$.
- For a query to the hash oracle, we apply to H a group element f_j and a claimed ciphertext m_j to compute $c = H(f_j, m_j)$
- A interaction with the signer is a three rounds deterministic protocol.

Each interaction with the signer provides an element r_i and increases the length of the input by one, so after the ith interaction, the input becomes:

$$l_1, \ldots, l_{t'}, r_1, \ldots, r_i.$$

Schnorr Signatures. Blind signatures are generated by an interaction with the signer who controls the secret signature key. Schnorr signatures refer to an arbitrary group G of prime order q and a arbitrary message space M. A signer interaction is an interactive protocol that enables a user to generate Schnorr signatures of message of its choice. Signatures will be based on a ideal hash function $H : G \times M \to \mathbb{Z}_q$, where M is the set of messages.

The private key x of the signer is random in $\mathbb{Z}_q$ and the corresponding public key $h = g^x$ is a random group element.

A Schnorr signature on a message m is a triple $(m, c, z) \in M \times \mathbb{Z}_q^2$ such that $H(g^z h^{-c}, m) = c$. The standard signature (m, c, z) on a message m is constructed as follow: we pick random $r, s \in \mathbb{Z}_q$, compute g^r, $c = H(g^r, m)$ and $z := r + cx$. The result is valid since we have $g^z h^{-c} = g^{r+cx} g^{-cx} = g^r$, and thus $H(g^z h^{-c}, m) = c$.

A signer interaction is a three rounds interactive protocol between the signer and a user. The user can generate from this protocol the standard signature (m, c, z) by selecting $c = H(g^r, m)$ but he has more options than that (he can generate a transformation (m, c', z') of this signature). The signer picks a random $r \in \mathbb{Z}_q$ and sends the commitment g^r to the user. The user selects a challenge $c \in \mathbb{Z}_q$ and sends c. The signer responds by sending $z := r + cx \in \mathbb{Z}_q$.

4.1 Signature Forgery Attack

We study security against the one-more signature forgery, security means that an attacker can not obtain $l + 1$ valid signatures from l interactions with the signer (signature oracle).

Parallel attack. This is the generic parallel attack for Schnorr signatures. We assume that the attacker makes τ queries to the hash oracle, l interactions with a signer and we construct a $l + 1$ valid signature. For the attack to succeed, we do not use the generator g and the public key h.

The signer picks $r_1, \ldots . r_l$ and sends commitments $g_1 := g^{r_1}, \ldots, g_l := g^{r_l}$. The attacker computes the group elements $f_i := \prod_{j=1}^{l} g_j^{a_{i,j}}$ and $H(f_i, m_i)$ for $i = 1, \ldots, \tau$. Then the attacker takes a subsystem of $l+1$ equations among these τ equations (2) in the unknowns $c_1, \ldots, c_l$ over $\mathbb{Z}_q$.

$$H(f_i, m_i) = \sum_{j=1}^{l} a_{i,j} c_j \quad for\ i = 1, \ldots, \tau \tag{2}$$

If the attacker solves this subsystem, it obtains a solution $c_1, \ldots, c_l$ and it sends the obtained solution to the signer that responds by $z_j := r_j + c_j x \in \mathbb{Z}_q$ for $j = 1, \ldots, l$. The attacker gets a valid signature (m_i, c, z) by setting

$$c := \sum_{j=1}^{l} a_{i,j} c_j = H(f_i, m_i) \quad and\ z := \sum_{j=1}^{l} a_{i,j} z_j$$

In the ROM, the coefficients $a_{i,j}$ selected by the attacker are arbitrary values and the values $H(f_i, m_i)$ are random. The generic parallel attack uses a solution of the ROS-problem (see (1)).

The objective of this interactive model is to etablish upper bounds for the probability of a generic attacker to construct a one more signature forgery.

Let us explain the ways to make a valid signature. We assume that the algorithm outputs a signature $sig := (m_i, c'_i, z'_i)$; a signature (m, c, z) is valid if $c = H(g^z h^{-c}, m)$; so sig is valid if $c'_i = H(g^{z'_i} h^{-c'_i}, m_i)$ and the group element $g^{z'_i} h^{-c'_i}$ must be among the computed elements $f_1, \ldots, f_{t'}$ (because c'_i is taken among the results of the hash queries, so there exists $k \in \{1, \ldots, \tau\}$ such that $c'_i = H(f_k, m_i)$); we let $f_k = g^{z'_i} h^{-c'_i}$. By the equations $g^{z'_i} h^{-c'_i} = f_k = g^{a_{k,-1} + a_{k,0} x + \sum_{j=1}^{l} a_{k,j} r_j}$ and $r_j = z_j - c_j x$, we have:

$$z'_i = log_g\ g^{z'_i} h^{-c'_i} + c'_i x$$

$$z'_i = a_{k,-1} + \sum_{j=1}^{l} a_{k,j} z_j + (a_{k,0} - \sum_{j=1}^{l} a_{k,j} c_j + c'_i) x \tag{3}$$

z'_i is valid one of the following two cases occur:

- if $c'_i = -a_{k,0} + \sum_{j=1}^{l} a_{k,j} c_j$ (4.1) then the equation (3) does not depend on the secret key x and $z'_i = a_{k,-1} + \sum_{j=1}^{l} a_{k,j} z_j$ where the coefficients $a_{k,-1}, \ldots, a_{k,l}$ and the signer responses are known to $\mathcal{A}$.
- $c'_i \neq -a_{k,0} + \sum_{j=1}^{l} a_{k,j} c_j$ and so we solve the equation (3) in x.

A one-more signature forgery can only succeed in either of four cases:

1. $\mathcal{A}$ find among the τ equations $H(f_k, m_i) = -a_{k,0} + \sum_{j=1}^{l} a_{k,j} c_j$ (4.1) a solvable subsystem of $l+1$ equations. We must apply the ROS-problem to find a bound of the probability of finding a solvable subsystem of $l+1$ equations. This corresponds to the first case to obtain a valid signature z'_i. This is the generic parallel attack. We let $Par\_Attack(\mathcal{A})$ be this condition for an attacker $\mathcal{A}$.
2. For some i, $1 \leq i \leq l+1$ equation (4.1) does not hold but equation (3) holds. Each interaction with the signer provides a polynomial $log_g g^{z'_i} h^{-c'i} + c'_i * x - z'_i$ (2), thus after l interactions with the signer, we obtain a list of l polynomials (2). We obtain information on the secret if we can find a zero of a polynomial that belongs to this list. This corresponds to the second case to obtain a valid signature z'_i. We let $Sign(\mathcal{A})$ be this condition for an attacker $\mathcal{A}$.
3. There is a collision of group elements. We let $\mathcal{CO}(\mathcal{A}))$ be this condition for an attacker $\mathcal{A}$.
4. There is a collision of hash values $H(f_i, m_i) = H(f_j, m_j)$, where $m_i = m_j$, $f_i \neq f_j$ and $a_{i,k} = a_{j,k}$ for $k = 0, \ldots, l$. We let $\mathcal{CO}_H(\mathcal{A}))$ be this condition for an attacker $\mathcal{A}$.

By having a collision (of group elements or hash values), we get a bound of the probability of finding the secret x; if we have the secret x, we can generate a valid signature $z = c + rx$ where c and r are knowned by the algorithm and x is found by the algorithm.

In an interactive generic algorithm, the attacker might obtain a one more signature forgery either through collisions on computed group elements or on hash values, or through interactions with the signer. Thus its advantage will be bounded by the probability of finding a collision on computed group elements plus the probability of finding a collision on hash values plus the probability of finding informations on the secret by an interaction with the signer. In the latter case, we show that the attacker can only obtain information if it succeeds on a parallel attack or if it can find a zero of a polynomial equation derived from the equality tested to know if a signature is valid, i.e. $c = H(g^z \bar{h}^{-c}, m)$.

5 Formalization of an Interactive Generic Algorithm

5.1 Formalization

The main difficulty in formalizing interactive generic 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.

An interactive generic algorithm is defined in the figure 2. 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 3). We assume that all ciphertexts have the type *SymbM* (see line 2). *SymbH* (see line 4) takes a list of coefficients instead of a computed group element i.e, a formal result for an hash query $i : SymbH$ is of the form (a, m, c), where m is a ciphertext.

Interactive generic algorithms are defined inductively (see line 6) 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 signstep i.e, an interaction with the signer; let us remenber what is an interaction with the signer, the signer picks a random r in $\mathbb{Z}_q$ and sends g^r to the user which sends $c = H(g^r, m)$ and the signer responds $z := r + cx$.

To make a *signstep* (see line 10) i.e, an interaction with the signer, we need to have an hash value $i : SymbH$ and a secret $r : \mathbb{Z}_q$ (it is the secret that the signer picks in $\mathbb{Z}_q$ and sends g^r to the attacker); and we take an $i : SymbH := (a, m, c)$ to send to the signer $c = H(a, m)$.

Interactive generic algorithms have three kinds of outputs:

- The symbolic hash outputs (see line 12) are just the list of *SymbH* and we can obtain the list of concrete hash outputs (see line 20) by applying

```
Parameter Sec Val:Set.
Parameter SymbM:Set.
Parameter input:list ℤ_q[Sec].
Definition SymbH:=(list ℤ_q)*SymbM*Val.

Inductive IGA : Type :=
   erun: IGA
 | mexstep: IGA → list ℤ_q →  IGA
 | hashstep: IGA → SymbH →  IGA
 | signstep: IGA →SymbH → Sec→  IGA .

Fixpoint SymbHOut (𝒜 : IGA) : list SymbH:=
match 𝒜 with
    erun ⇒ nil
  | mexstep 𝒜' e ⇒ SymbHOut 𝒜'
  | hashstep 𝒜' (a,m,c) ⇒(a,m,c)::(SymbHOut 𝒜')
  | signstep 𝒜' _ _⇒ SymbHOut 𝒜'
 end.

Definition ConcrHashOutput(r:IGA)(τ:Val→ℤ_q):(list ℤ_q):=
map τ (map λ(x,y,z).z (SymbHOut 𝒜)).

Fixpoint SymbMexOutput(𝒜:IGA): list ℤ_q[Sec]:=
  match 𝒜 with
  | erun ⇒nil
  | mexstep 𝒜' e ⇒ (mex e input)::(SymbMexOutput 𝒜')
  | hashstep 𝒜' _ ⇒ SymbMexOutput 𝒜'
  | decstep 𝒜' _ ⇒ SymbMexOutput 𝒜'
end.

Definition ConcrmexOutput(𝒜: IGA)(σ:Sec→ℤ_q):(list ℤ_q):=
map λx.[|x|]_σ (SymbMexOutput 𝒜).

Definition mk_z(τ:Val→ℤ_q)(r:ℤ_q)(c:Val)(x:ℤ_q):ℤ_q:=
(τ c)+ r*x.

Fixpoint IdealSign(𝒜:IGA)(τ:Va →ℤ_q)(σ:Sec→ℤ_q): listT ℤ_q:=
 match 𝒜 with
    erun ⇒ nil
  | mexstep 𝒜' e ⇒ IdealSign 𝒜' τ σ
  | hashstep 𝒜' _ ⇒ IdealSign 𝒜' τ σ
  | signstep 𝒜' (a,m,c) r ⇒ (mk_z τ (σ r) c
    (Eval (head (tail input))))::(IdealSign 𝒜' τ σ)
 end.

Definition 𝒞𝒪_H(𝒜: IGA)(τ:Val→ℤ_q):Prop:=
∀ e e':Val, e ∈ (map λ(x,y,z).z (SymbHOut 𝒜)) ∧
       e' ∈ (map λ(x,y,z).z (SymbHOut 𝒜)) ∧ e - e'≢0 ∧
(τ e)-(τ e')=0 .
```

Fig. 2. Formalizatiom of interactive algorithm for parallel attack

an evaluation function *rom* in the last element of the list of symbolic hash outputs.
- Symbolic group outputs (see line 23) are polynomials constructed as linear combinations of inputs in the same way of non-interactive generic algorithms. Concrete group outputs (see line 31) are obtained from the symbolic group outputs by using the extension of an interpretation function σ from polynomial expressions to elements in $\mathbb{Z}_q$. `mk_z` (see line 34) returns the $z := c + rx$ computed by the signer by evaluating the hash result c with an interpretation function *rom*.
- The outputs of the signer (see line 37) are the computed elements z.

We can find a non-trivial collisions among hash outputs (see line 46) if we can find two polynomials $e\ e' : Val$ non identically equal in the hash outputs such that the interpretation of $e - e'$ under *rom* is 0.

Interactive generic algorithm can succeed on a one-more signature forgery if it can find a collision on computed group elements or on hash values or if the interpretation under the function σ of the derived polynomial (3) is 0 (knowing that the equality (4.1) does not hold) or if it succeed on a parallel attack.

5.2 Security of Blind Signatures Against Interactive Attacks

The way to obtain a one more valid signature is given in the figure 3.

A one-more signature forgery can only succeed in either of four cases:

1. $\mathcal{A}$ find among the τ equations $H(f_k, m_i) = -a_{k,0} + \sum_{j=1}^{l} a_{k,j} c_j$ (4.1) a solvable subsystem of $l+1$ equations, let us notice that we see the equation (4.1) as the polynomial $\sum_{j=1}^{l} a_{k,j} c_j - a_{k,0} - H(f_k, m_i)$ in $\mathbb{Z}_q[c_1, \ldots, c_l]$. We must apply the ROS-problem to find a bound of the probability of finding a solvable subsystem of $l+1$ equations. To formalize the ROS-problem, we do not take a function $F : \mathbb{Z}_q^l \to \mathbb{Z}_q$ but a function $F : (list\ \mathbb{Z}_q) \to \mathbb{Z}_q$ with a list of length l. The variables $c_1, \ldots, c_l$ are of the type Val so we see each equation (4.1) like a polynomials in $\mathbb{Z}_q[Val]$. And we compose the function $F : (list\ \mathbb{Z}_q) \to \mathbb{Z}_q$ like $G \circ F' + coeff$ where $F' : Val \to \mathbb{Z}_q$ is an interpretation function that maps formal variables to actual values, $G : (list\ \mathbb{Z}_q) \to Val$ and $coeff$ is an element of $\mathbb{Z}_q$. Finally, be given the coefficients $a_{k,j} \in \mathbb{Z}_q$ for $j = 1, \ldots, l$ and $k = 1, \ldots, \tau$ ($list\_a_k$), the list of coefficients $coeff \in \mathbb{Z}_q$ ($list\_coeff$) and a function $G : (list\ \mathbb{Z}_q) \to Val$, we have a list of τ polynomials of the form $a_{k,1}c_1 + \ldots + a_{k,l}c_l - coeff - F'(G(a_{k,1}, \ldots, a_{k,l})$ and the ROS-problem is to find a sublist of $l+1$ polynomials having a common solution. If we let *ROS_pb* be this condition, *ROS_pb* is an event in $Val \to \mathbb{Z}_q$.

   For the moment, we have a paper proof of the ROS-problem but for this proof, we need to formalize the determinant of a matrix in Coq and we did not ever do it, so for the moment it is an axiom.

   Proposition 2. $Pr(ROS\_pb\ list\_a_k\ list\_coeff\ G) \leq \frac{\binom{\tau}{l+1}}{q}$

```
1 Definition list_ak(𝒜:IGA):(list (list ℤ_q)):=
2 map λ(x,y,z).x (SymbHOut 𝒜)).
3
4 Definition list_coeff(𝒜:IGA):(list ℤ_q):=
5 map λ(x,y,z).x (map λ(x,y,z).x (SymbHOut 𝒜))).
6
7 Definition mk_G(l:listSymbH)(a:list ℤ_q):Val:=
8 if ((h::a),m,c) ∈ l then c
9                 else ?.
10
11 Definition G(r:IGA):(list ℤ_q) →Val:=
12 mk_G (SymbHOut r).
13
14 Definition Pol_H_neq(g:listT Val)(τ:Val→ℤ_q)(h:SymbH):Prop=
15 let h:=(f_k,m,c) in
16 let Σ_{j=1}^{l} a_{k,j}c_j :=sum (tail (tail (Ft h))) (map τ g) in
17 let a_{i,0} :=(head (Ft h)) in
18 let c_i:=(τ (Td h)) in
19 a_{i,0} -Σ_{j=1}^{l} a_{k,j}c_j +c_i≠0.
20
21 Definition Pol_H_Pred(r:IGA)(τ:Val→ℤ_q):=
22 nb_P_true (Pol_H_neq (map λ(x,y,z).z(SymbHOut r))τ)(IdealOutput r)≤l+1.
23
24 Definition Sign(r:IGA)(τ:Val→ℤ_q)(h:SymbH)(r_i:Sec):ℤ_q[Sec]=
25 let h:=(f_k,m,c) in
26 let  log_g f_k :=(mk_pol (Ft h) (inp r)) in
27 let c_i:=(τ (Td h)) in
28 let x:=(mk_pol (head (tail input))) in
29 let z_i:=(mk_pol r_i)+ci*x in
30  log_g f_k +c_i*x-z_i.
31
32 Fixpoint list_Sign(r:IGA)(τ:Val→ℤ_q):list ℤ_q[Sec]:=
33 match r with erun ⇒ nil
34            | mexstep r' _ ⇒ list_Sign r τ
35            | hashstep r' _ ⇒list_Sign r τ
36            | signstep r' h r_i ⇒(Sign r' τ h z)::(list_Sign r' τ)
37   end.
38
39 Fixpoint Signer(𝒜:IGA)(σ:Sec→ℤ_q)(τ:Val→ℤ_q){struct 𝒜}:Prop:=
40 ∀ p:ℤ_q[Sec], p ∈ (list_Sign r τ) ∧ [|p|]_σ ≡ 0.
```

Fig. 3. Make a valid signature

To apply the ROS-problem i.e, to have the list of the τ polynomials $\sum_{j=1}^{l} a_{k,j}c_j - a_{k,0} - H(f_k, m_i)$, we must have the list of coefficients $a_{k,j} \in \mathbb{Z}_q$ for $j = 1, \ldots, l$ and $k = 1, \ldots, \tau$ (see line 1 of the figure 3), the list of coefficients $coeff := a_{k,0}$ for $k = 1, \ldots, \tau$ (see line 4) and a function

$G : (list\ \mathbb{Z}_q) \rightarrow Val$ (see line 11). As `SymbH:=(list` $\mathbb{Z}_q$`)*SymbM*Val`, we define the function `G` by:

```
Fixpoint mk_G(l:list SymbH)(a:list Zq):Val:=
match l with      nil    ⇒ ?
            |(h,a',c)::tl⇒ if a=a' then c
                           else (mk_G tl a)
end.
```

`(Ros_pb` τ`(list_ak r) (list_coeff r) (G r))` is the condition to find $l+1$ polynomials having a common solution $c_1, \ldots, c_l$. Then we send the solution $c_1, \ldots, c_l$ to the signer that responds $z_1, \ldots . z_l$. We obtain one-more signature by the setting:

$$c'_i = -a_{k,0} + \sum_{j=1}^{l} a_{k,j} c_j \quad and\ z'_i = a_{k,-1} + \sum_{j=1}^{l} a_{k,j} z_j$$

2. For some i, $1 \leq i \leq l+1$ equation (4.1) does not hold (see line 21) but equation (3) holds i.e, there exists at least a polynomial `p` $\in$`(list_Sign r` τ`)` such that its interpretation under the function σ is 0, each interaction with the signer provides a polynomial $a_{i,0} - \sum_{j=1}^{l} a_{k,j} c_j + c'_i$ (2), thus after l interactions with the signer, we obtain a list of l polynomials (2) (see line 32). This corresponds to the second case to obtain a valid signature z'_i. We let $Sign(\mathcal{A})$ be this condition for an attacker $\mathcal{A}$.
3. There is a collision of group elements.
4. There is a collision of hash values $H(f_i, m_i) = H(f_j, m_j)$, where $m_i = m_j$, $f_i \neq f_j$ and $a_{i,k} = a_{j,k}$ for $k = 0, \ldots, l$.

5.3 Properties of Interactive Generic Algorithms

In this section, we let a interactive generic algorithm $\mathcal{A}$ be given the generator g, the public key h, and oracles for the hash function H and for signature oracle. Let $\mathcal{A}$ performs t generic steps including t'' mex-steps, τ queries to the hash oracle and l interactions $(r_1, c_1, z_1), \ldots, (r_l, c_l, z_l)$ with the signer.We assume the input $l_1, \ldots, l_{t'}$ to be polynomial expressions over secrets . Further, we let d be the maximal degree of the polynomials l_j for $1 \leq j \leq t'$ where t' is the number of input.

Proposition 3. $\forall \mathcal{A}$ `:IGA`, $Pr(\mathcal{CO}(\mathcal{A})) \leq \frac{\binom{t'}{2} d}{q - \binom{t'}{2} d}$

Proof. All outputs are of the form $f_i = \sum_{1 \leq j \leq t'} a_j^i\ l_j(s_1, \ldots, s_k)$, where $p_i = \sum_{1 \leq j \leq t'} a_j^i\ l_j(x_1, \ldots, x_k)$ 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 $\leq d$. So we can use an extension of the Schwartz lemma [5] to deduce the expected result.

Proposition 4. $\forall \mathcal{A}$:IGA, $Pr(\mathcal{CO}_H(\mathcal{A})) \leq \frac{\binom{\tau}{2}d}{q-\binom{\tau}{2}d}$

Proof. In the same way of proposition 3.

Proposition 5. $\forall \mathcal{A}$:IGA, $Pr(Sign(\mathcal{A})) \leq \frac{(d+1)l}{q}$

Proof. The proof is by induction of the interactive generic algorithm $\mathcal{A}$. The only interesting case is when the algorithm interacts with the signer. In this case, we can find information on the secret iff σ is a solution of the extracted polynomial corresponding to the equation (3) which is of degree d.

Proposition 6. $\forall \mathcal{A}$:IGA, $Pr(ParAttack(\mathcal{A})) \leq \frac{\binom{\tau}{l+1}}{q}$

Proof. $ProbPar\_Attack(\mathcal{A}) \leq Pr(ROS\_pb\ \ (list\_ak\ \mathcal{A})\ (list\_coeff\ \mathcal{A})\ (G\ \mathcal{A}))$ In our formalization, we have an hash function of type $(list\ \mathbb{Z}_q) \times M \rightarrow Val$, where M is the set of all ciphertexts.

More precisely, we have a type $SymbH := (list\ \mathbb{Z}_q) \times M \times Val$, i.e, instead of having $c = H(f, m)$, where $f = g^{a_{-1}+a_0+a_1x+\sum_{j=1}^{l} a_j r_j}$, we have $((a_{-1}, \ldots, a_l), m,\ c) : SymbH$ where $c = H(g^{a_{-1}+a_0+a_1x+\sum_{j=1}^{l} a_j r_j}, m)$. Moreover, in our formalization on the ROS-problem, we do not take a function $F : \mathbb{Z}_q^l \rightarrow \mathbb{Z}_q$ but a function $F : (list\ \mathbb{Z}_q) \rightarrow \mathbb{Z}_q$ with a list of length l. So to have a function in $(list\ \mathbb{Z}_q) \rightarrow \mathbb{Z}_q$, we define a function $G : Run \rightarrow (list\ Zq) \rightarrow Val$ (see line 11). Having an interactive generic algorithm $\mathcal{A}$, we have the function $rom(G(\mathcal{A})) : (list\ \mathbb{Z}_q) \rightarrow \mathbb{Z}_q$. So we can apply the ROS-problem.

In the interactive setting, we consider that the advantage of the attacker is bounded the probability of finding non-trivial collisions on computed group elements or hash values plus the probability of finding a zero of a polynomial resulting on an interaction with the signer plus the probability of succeeding on a parallel attack.

Proposition 7. $\forall \mathcal{A}$:IGA, $Advantage(\mathcal{A}) \leq Pr(\mathcal{CO}(\mathcal{A})) + Pr(\mathcal{CO}_H(\mathcal{A})) + Pr(Sign(\mathcal{A})) + Pr(ROS\_pb\ (list\_ak\ \mathcal{A})\ (list\_coeff\ \mathcal{A})\ (G\ \mathcal{A}))$.

6 Conclusion

We have extended our previous machine-checked account of the GM and ROM to establish security bounds for interactive algorithms, and in particular to show the security of signature schemes against forgery attacks. Our results generalize existing results to the case of an arbitrary generic algorithm, and provide more rigorous bounds than those present in the literature.

Machine-checked proofs of provable cryptography has barely been scratched. Much work remains to be done in the context of the GM and ROM: in particular, we intend to provide a machine-checked treatment of ROS, and to exploit our formalizations to prove the security of realistic protocols, following e.g. [7,23]. An even more far-fetched goal would be to give a machine-checked account of a formalism that integrates the computational view of cryptography, and provable cryptography.

Acknowledgments. I am grateful to Gilles Barthe, for his constructive and detailed advices and to the anonymous referees for their useful 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. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, 2000. Springer-Verlag, Berlin Germany.
3. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library, 2003.
4. 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.
5. 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.
6. 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.
7. D. Brown. Generic Groups, Collision Resistance, and ECDSA, 2002. Available from `http://eprint.iacr.org/2002/026/`.
8. Coq Development Team. *The Coq Proof Assistant User's Guide. Version 8.0*, January 2004.
9. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
10. 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.
11. formalization of probability . http://www-sop.inria.fr/everest/personnel/Sabrina.Tarento/useful_files.html.
12. R. Impagliazzo and B. Kapron. Logics for reasoning about cryptographic constructions, 2003.
13. S. Lang. *Algebra.* Addison Wesley, 1983.
14. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In J. M. Wing, J. Woodcock, and J. Davies, editors, *Proceedings of FM'99—Volume I*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793. Springer-Verlag, 1999.
15. 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.
16. J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols (preliminary report). In S. Brookes and M. Mislove, editors, *17-th Annual Conference on the Mathematical Foundations of Programming Semantics*, number 45, Aarhus, Denmark, 2001.

17. John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *IEEE Symposium on Foundations of Computer Science*, pages 725–733, 1998.
18. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.
19. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of SOSP'01*, pages 184–201. IEEE Press, 2001.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.