Programmability in the Generic Ring and Group Models

Mario Larangeira*
Tokyo Institute of Technology
W8-55, 2-12-1 Ookayama Meguro-ku, Tokyo 152-8552, Japan
larangeira.m.aa@m.titech.ac.jp
Keisuke Tanaka †
Tokyo Institute of Technology
W8-55, 2-12-1 Ookayama Meguro-ku, Tokyo 152-8552, Japan
keisuke@is.titech.ac.jp

Abstract

The programmability has long been used as a tool to prove security of schemes in the random oracle model (ROM) even in the cases where schemes do not seem to have a security proof in the standard model [3, 8, 10]. On the other hand, it seems that a similar property has never been studied in the generic models, i.e., the generic ring and group models, respectively the GRM and the GGM. This work introduces this study. We start by proposing the use of the GRM and the GGM in simulation-based security proofs, instead of the classical two-step approach: (1) find an efficient reduction R from a problem P to an adversary breaking the scheme in some sense, and (2) use the GRM/GGM to find a lower bound in the complexity of solving P. We observe that in such a model the simulator can choose the outputs for the generic operation oracle in a similar fashion as the programmability property of the ROM. We introduce four models named *programmable and non-programmable* for the GGM and analogously for the GRM. We show that in the programmable generic models it is possible to turn around the negative result by Nielsen [21], regarding the non-committing encryption in the presence of an adversary who corrupts the receiver. We illustrate our idea by proving that the Goldwasser-Micali encryption scheme is a non-committing encryption scheme regarding corruption of the receiver in the programmable GRM. Whereas, for the programmable GGM, we show that the popular ElGamal encryption scheme is also non-committing despite the corruptions of the receiver and the sender. In both schemes the attack exposes the secret key.

Keywords: Non-committing, programmability, generic ring model, generic group model

1 Introduction

Before giving details of our contribution and motivation, we start by giving a short introduction of the evolution of generic models and a description of how the property of programmability has arisen originally in security proofs based on the random oracle model.

1.1 Background

In the cryptographic literature the following models appear very often: (1) the random oracle model (ROM), and (2) the generic group model (GGM) and, its generalization, the generic ring model (GRM).

The GGM, formally introduced by Shoup [23] and Nechaev [20], captures the situation that *no property* of the representation of the groups are available to be exploited. The only available features are the group operation and the equality test between two members of the group. Both features are modeled as a generic operation oracle. The generalized notion of GRM models all the ring operations,

Journal of Internet Services and Information Security, volume: 1, number: 2/3, pp. 57-73
*Supported by Ministry of Education, Culture, Sports, Science and Technology.
†Supported in part by NTT Information Sharing Platform Laboratories and Grant-in-Aid for Scientific Research.

i.e., $\{+,-,\times,/\}$, and tests for equality and existence of the inverse of the members of the ring. Both GGM and GRM have been used along the years to provide evidence for intractability of computational problems and to investigate relations between problems, e.g., [1, 18, 23]. One of the most famous use of the GGM is in provable security, precisely in the two-step security proofs. Namely, (1) find a reduction, say B, from a problem P to a successful adversary, and (2) show that the P is intractable in GGM.

Since its introduction in the Nineties, it has been used to argue about the assumed hardness of computational problems, i.e., computing discrete logarithms, the computational Diffie-Hellman problem, the decisional Diffie-Hellman problem and etc. Maurer introduced a different formulation [18], which Jager and Schwenk showed to be equivalent to Shoup's formulations [15] in a work which studies the models themselves. In 2008, the GGM was generalized to the GRM by Aggarwal and Maurer [1] using ideas already proposed by Boneh and Venkatesan [4]. Based on [1], Jager and Schwenk [16] improved once more the formulation of the GRM and showed that there are problems that can be solved efficiently in practice, however in the GRM they are equivalent to factoring the modulus $N = p \cdot q$.

More recently, the study of problems in the generic models has been improved by the introduction of a new model named the semi-generic group model by Jager and Rupp [14]. This new model aims to cover the recently increased number of problems specially after the advent of pairing cryptography.

1.2 Related Work

A main goal in cryptography is to implement adaptively secure protocols. So far only statically secure implementation from the information theoretical model[1] are known. The idea is to implement them in the cryptographic model without erasure[2]. This approach is based on Canetti's observation in [5, 6] that any CCA (*chosen ciphertext attack*) secure encryption scheme would work as a secure implementation as long as it has the *non-committing*[3] property [6], and includes in the message two identification strings in order to protect against retransmission of the message by an attacker: one for the message (id_m) and the other for the sender (id_{SEN}).

NON-COMMITTING ENCRYPTION AND PROGRAMMABILITY IN ROM. A particular result about the ROM was discovered by Nielsen in [21]. He was the first to observe that a specific property of the ROM is required to prove security for an entire class of encryption algorithms, i.e., non-interactive and non-committing encryption (NINCE). The property name was coined the *programmability* of the ROM. Briefly, programmability is the power that B has to choose the output of the random oracle when interacting with the adversary (e.g., B chooses y arbitrarily and sets $h(x) = y$). That, ultimately, may help B in solving P, giving the security proof. This result essentially means that there is no NINCE protocol in the standard model, since no hash function is known to exist with the programmability feature required by actual security proofs despite the efforts to construct this kind of primitives [13].

1.3 Motivation

The study of programmability in the generic models does not seem to have been proposed yet, although the idea of "programming the oracle" is not new, and even similarities, as *gaps* between the models and real situations, have been observed in the ROM and the GGM, e.g., Canetti, Goldreich and Halevi [7] and Dent [9]. Hence from a theoretical viewpoint, this work partially fills a gap regarding generic models in the current status of the research as shown in Table 1.

The work of Nielsen confirms the intuition that proofs which rely on the programmability of the model are indeed more powerful, in the sense that more schemes can be proved secure. Therefore, from

[1] Where secure channels are assumed to exist.

[2] Where there are no secure channels and the parties are not trusted to erase their computation.

[3] Briefly, the encryption scheme that has the property of generating ciphertexts which can be opened to two different plaintexts.

	ROM	GGM	GRM
Gap	[7, 19]	[9, 11, 17]	[16]
Programmability	[12, 21, 24]	**[This work]**	**[This work]**

Table 1: Research on gaps and programmability for the ROM, the GGM and the GRM.

a more practical viewpoint our approach of using the programmability of the generic operation oracle is interesting because it offers a new observation and an alternative to the reduction-centric two-step security proofs in the generic models.

1.4 Our Contribution

In this paper we introduce the use of the programmability in the generic models. The intuition of our models comes from the two-step approach for security proofs. Recall that the lower bound in the complexity in the GRM and the GGM only applies for algorithms in these models. Therefore, although the reduction B can be general, i.e., for all classes of algorithms, the security may not be carried from the generic models. Implicitly, these proofs assume that every party has access to the generic group (or ring) operation oracle. Our models explicitly capture this feature by using the G-hybrid model from the universal composable (UC) framework [5] for the GGM (and R-hybrid model for the GRM). This makes our programmable models equivalent to the regular GGM [23, 20] and GRM [1, 16].

The difference between the two models we propose is analogous to the Nielsen's programmable and non-programmable ROM. For example, in the case of the GGM, we make G widely available in the non-programmable GGM while *not* in the programmable counterpart.

Recall that the generic operation oracle collects the operations as steps, formally as a tuple $(i, j, \circ)$, where $\circ$ is one of the group operations (or ring operations) and i and j are pointers for previously computed steps. We propose the use of these steps to "program" the generic operation oracle similarly to the random oracle model. That is, when the simulator, in a simulation-based proof of security, needs to simulate the ciphertext between the parties without knowing the message m, it delivers steps with *no computation associated with it* as the ciphertext. This approach allows the simulator, after learning m, to associate a suitable computation consistent with m, therefore creating a non-committing ciphertext. Our main goal in this work is to study whether non-committing schemes are possible in generic models.

We present our four models in Section 3. They are the programmable and the non-programmable GGM, and the GRM versions: the programmable and the non-programmable GRM.

In Section 4.2, we illustrate our proof technique using the programmability by analyzing the ElGamal encryption scheme in the programmable GGM. Briefly, the proof gives a construction for a simulator and uses the Schwartz Lemma [22, 23] to find a lower bound in the distinguishing game. Thereby showing that ElGamal encryption scheme is non-committing under corruption of the sender and the receiver.

In Section 5.2, we rely on the formulation and machinery for the GRM by Jager and Schwenk [16] to show that the Goldwasser-Micali (GM) is also a non-committing encryption scheme regarding only receiver corruptions in the programmable GRM. This result does not rule out the possibility of a non-committing encryption scheme considering also sender corruptions, however it is sufficient to circumvent the result in [21] in the computational setting.

We point out that in both cases it is possible to prove despite the fact that the secret-key is exposed when the receiver is corrupted[4], which has been avoided in other schemes [2, Section 1.2 and 1.3]. In the case of the non-programmable GGM and GRM, the result in [21] rules out the existence of non-committing schemes.

[4]Which is a crucial step in the Nielsen's information-theoretic argument in the ROM without programmability.

One should have in mind that our proofs are in generic model, i.e., they provide the same caveats as the regular proofs in the ROM [7, 19] and generic model. In particular, those discussed in [16], regarding multiplicative groups of finite fields [14] and proofs which rely on programmability [12]. On the other hand, there are certain elliptic curve groups that seem to fit into the GGM, because not many properties are known out of the group operations. For those cases, our result in the programmable GGM shows that non-committing encryption are possible in such a model.

1.5 Comparing ROM and Generic Models

We observe that while the ROM is a more accepted model to base the security of cryptographic schemes, some skepticism is levered against generic models [9, 11]. However, in the case of the GGM, some researchers claim that the issues raised along the years are in fact due to proofs given in the models and not from weaknesses of the GGM, e.g., [17, Section 5], thus the model is as worth study as the ROM.

We recall that in the ROM, it is hopeless to use the model to try to justify the security of a system when a broken hash function is considered as a substitute for the random oracle (think, for example, the already broken MD5 hash function for signature schemes). Analogously, for the case of the GGM, it is hopeless to use the model to prove the security of schemes when the concrete instantiation of the group is a "bad" choice. Consider, for example, as "bad" choice the groups that are known to be susceptible to the use of the index calculus algorithm, that gives a much faster and potential alternative for adversaries which does not fit in the GGM.

Another common concern about proofs in generic models is that it applies to a restricted class of *generic* adversaries, i.e., adversaries that do not receive a description of the concrete group being considered. We emphasize that the same is true for proofs in the ROM. Recall, that the adversary in ROM does not receive the description of a hash function, whereas in an implemented system, i.e., when a good hash function is used instead of the random oracle, the security analysis must assume that the description of the hash function is widely available.

To strengthen our analogy, in this work, we see the ROM as a "generic hash function model". That is, a model where the adversary treats the hash function as a black box, much in the same way as for the generic group oracle used in the GGM. Thus, if the proofs that rely on the programmability of the ROM are accepted as reliable security guarantees for a family of good hash functions, then we believe the same should be true for the case of "good" cyclic groups. Where "good" means groups that properties beyond the group laws are not known to exist (certain elliptic curves groups, for example).

In comparison to the GGM, not too many schemes are known to have security proofs exclusively in the GRM. Our result shows that a similar technique applies to the GRM, although the model has been showed not to be soundness [16]. Our results in the GGM and the GRM, likewise those in the ROM, show that if security proofs in these models are the only guarantee of security of a scheme, then it might be possible to show that such a scheme is also non-committing through similar use of the programmability.

2 Preliminaries

For the rest of the paper, consider k as the security parameter. While $l_m(k)$, $l_r(k)$ and $q(k)$, respectively the number of bits of the message, the randomness and the number of the queries to the generic oracle, are polynomials of k.

Similarly, consider that negligible functions are negligible with respect to k, i.e., negl(k), if for any $c,d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and for all $x \in \{0,1\}^{k^d}$, it holds that $f(x,k) < k^{-c}$. Furthermore, *gcd* stands for the greatest common divisor. Finally, consider that x is uniform randomly chosen from some set X when it is written $x \xleftarrow{U} X$.

We start by reviewing the definitions for GGM, then we continue with a review of the GRM and a few lemmas for straight line programs. This section ends with a brief review of the universal composable framework and the non-committing functionality.

2.1 The Schwartz Lemma and the Generic Group Algorithms

Lemma 2.1 (Schwartz Lemma [22, 23])**.** *Let p be prime and $L(X_1,\ldots,X_\ell)$ an arbitrary non-zero polynomial on the random variables $X_1,\ldots,X_\ell \in \mathbb{Z}_p$ with total degree d for $\ell \in \mathbb{N}$. Then* $\Pr[L(x_1,\ldots,x_\ell) = 0] \leq \frac{d}{p}$ *for a uniformly random assignment $X_i \leftarrow x_i \in \mathbb{Z}_p$, $1 \leq i \leq \ell$.*

A generic group algorithm A is allowed to submit operation and equality queries to the generic group operation oracle G. The queries are tuples $(i,j,\circ)$ for $\circ \in \{=,\times\}$, where $\times$ is the group operation.

Definition 2.2 (Generic Group Oracle G [18, 23])**.** The oracle G initializes by receiving a value $x \in \mathbb{Z}_p^*$ and keeping a list of polynomials L_r, $1 \leq r \leq q(k)$ associated with every submitted operation $(i_r, j_r, \circ_r)$, with additional convention that $L_{-1} := 1$ and $L_0 := x$. The adversary submits queries $(i_r, j_r, \circ_r)$ and G runs the **operation** procedure:

operation$(i_r, j_r, \circ_r)$:
1. It takes a tuple
$(i_r, j_r, \circ_r) \in \{-1,\ldots,r-1\} \times \{-1,\ldots,r-1\} \times \{=,\times\}$
2. If $\circ_r \neq \{=\}$ then
   If $\exists r' \neq r : L_{r'} = L_{i_r} \times L_{j_r}$ then return r'
   Else
      $L_r := L_{i_r} \times L_{j_r}$
      return r
 Else
   If $L_{i_r} = L_{j_r}$ then return $true$
   Else return $false$

The complexity of A is measured as the number of oracle queries.

2.2 Straight Line Programs and Generic Ring Algorithms

From here, assume that $N = p \cdot q$ for primes p and q, where $|\mathbb{Z}_N^*| = \phi(N) = (p-1)\cdot(q-1)$ is the Euler totient function.

We adapted the next definitions and lemmas from [16, Lemmas 2, 3 (Appendix C) and 4 (Appendix D), respectively] and ideas from [1]. The main change is that now we consider the tuples X and Y defined as $X = (x_1,\ldots,x_\ell)$ and $Y = (y_1,\ldots,y_\ell)$ where $\ell \in \mathbb{N}$ and x_i and y_i are sampled independently. The originals are defined over single values. We skip the proofs of the lemmas, therefore we refer the reader both works for more details.

Definition 2.3 (Straight Line Programs (SLP) [16, 4])**.** A straight line program P of length m over $\mathbb{Z}_N$ is a sequence of tuples $P = ((i_1, j_1, \circ_1), \ldots, (i_m, j_m, \circ_m))$, where $-\ell \leq i_k, j_k < k$ and $\circ_r \in \{+,-,\times,/\}$ for $r \in \{1,\ldots,m\}$, with the additional convention $P_0(X) = 1$, $P_{-1}(X) = x_1, \ldots, P_{-\ell}(X) = x_\ell$ for all $x_i \in \mathbb{Z}_N$, where $1 \leq i \leq \ell$.

The triple $(i,j,\circ) \in P$ is an *SLP-step*. We denote by P_k the SLP defined by the first k SLP-steps of P and $P_k \sqsubseteq P$ to denote that the SLP P_k means the first k SLP-steps of P.

The output $P(X)$ of an SLP P is computed as follows.

1. For $1 \leq i \leq \ell$, initialize $L_{-i} := x_i \in \mathbb{Z}_N$ and $L_0 := 1$.
2. For k from 1 to m do:
   - If $\circ_k = /$ and $L_{j_k} \notin \mathbb{Z}_N^*$ then return $\perp$;
   - else set $L_k := L_{i_k} \circ_k L_{j_k}$.
3. Return $P(x) = L_m$.

Lemma 2.4. *Suppose there exist a straight line program P such that for $X, X' \in (\mathbb{Z}_N^*)^\ell$ it holds that $P(X') \neq \perp$ and $P(X) = \perp$. Then there exists $P_j \sqsubseteq P$ such that $P_j(X') \in \mathbb{Z}_N^*$ and $P_j(X) \notin \mathbb{Z}_N^*$.*

Lemma 2.5. *For every SLP P,*

$$\Pr[gcd(P(Y),N) \notin \{1,N\} \mid Y \xleftarrow{U} (\mathbb{Z}_N^*)^\ell] \geq \Pr[P(X) \notin \mathbb{Z}_N^* \text{ and } P(X') \in \mathbb{Z}_N^* \mid X, X' \xleftarrow{U} (\mathbb{Z}_N^*)^\ell].$$

Lemma 2.6. *For every two SLP P and Q,*

$$\begin{aligned} \Pr[gcd(P(Y) - Q(Y), N) \notin \{1,N\} \quad &| \quad Y \xleftarrow{U} (\mathbb{Z}_N^*)^\ell] \\ &\geq \quad \Pr[P(X) \neq_N Q(X) \text{ and } P(X') \equiv_N Q(X') | X, X' \xleftarrow{U} (\mathbb{Z}_N^*)^\ell]. \end{aligned}$$

A generic ring algorithm A is allowed to submit SLP-steps $(i, j, \circ)$ to the generic ring operation oracle R.

Definition 2.7 (Generic Ring Oracle R [16])**.** The oracle R is initialized by keeping a SLP P initially empty except for $P_{-1}(x) = x$ and $P_0(x) = 1$ and the received value $x \in \mathbb{Z}_N$. The checking procedures are done by implementing the routines **test** and **equal**. The algorithm A submits SLP-steps $(i, j, \circ)$ to perform computation. For every tuple $(i, j, \circ)$, R executes the procedure **operation**:

test$(j, \circ)$:
1. It takes a tuple $(j, \circ) \in \{-1, \ldots, |P|\} \times \{+, -, \times, /, =\}$
2. If $\circ = /$ and $P_j(x) \notin \mathbb{Z}_N^*$ then
   Return false
Else
   Return true

equal(j, i):
1. It takes a tuple $(j, i) \in \{-1, \ldots, |P|\} \times \{-1, \ldots, |P|\}$
2. If $P_i(x) \equiv P_j(x) \mod n$ then
   Return true
Else
   Return false

operation$(i, j, \circ)$:
1. It takes a tuple $(i, j, \circ)$
2. If $\circ \neq \{=\}$ then
   If **test**$(j, \circ) \rightarrow$false then
      Return $\perp$
   Else add $(i, j, \circ)$ to P
Else
   Return **equal**(i, j)

The complexity of A is measured as the number of oracle queries.

Remark 2.8. In order to program the oracles we need an additional convention. For a member x of the ring or group, $[x]$ means the pointer which G, or R, give to A, so the adversary can carry out the computation.

We review the universal composable framework, in particular the hybrid extension of the framework. We refer the reader to [5] for more details.

2.3 The Universal Composable Framework and the Non-committing Functionality

In the framework all the parties are probabilistic polynomial time (PPT) interactive Turing machines[5]. Giving the sender and receiver parties, *SEN* and *REC*, the protocol $\pi = (SEN, REC)$ performs a REAL execution of the protocol when *SEN* and *REC* run the code of π in the presence of the adversary A. Moreover, A can see and schedule the messages between the parties. On the other hand, there is the IDEAL execution which is defined by the functionality F, the simulator S, and the parties $\widetilde{SEN}$ and $\widetilde{REC}$. The parties in the IDEAL execution only deliver its inputs to F and passes the outputs of F without performing any computation. Both executions are driven by the environment Z, which plays the role of the distinguisher and controls the inputs of the parties as well as the actions of A.

THE HYBRID MODEL. In addition with the previous model, the hybrid model is the extension where all parties have secure access to a special machine G named also functionality. At the beginning, the environment receives the security parameter k and an auxiliary input $z \in \{0,1\}^*$. Let $\mathrm{HYB}^{\mathsf{G}}_{\pi,\mathsf{A},\mathsf{Z}}(k,z)$ be the random variable describing the output of Z in the hybrid execution. Whereas $\mathrm{HYB}^{\mathsf{G}}_{\tilde{\pi},\mathsf{S},\mathsf{Z}}(k,z)$ is for the ideal execution. Let $\mathrm{HYB}^{\mathsf{G}}_{\tilde{\pi},\mathsf{S},\mathsf{Z}}(k,z) = \mathrm{IDEAL}_{\mathsf{F},\mathsf{S},\mathsf{Z}}(k,z)$.

It is said that π *securely realizes* F *in the* G*-hybrid model*, when for any adversary A there exists an ideal adversary S such that for any environment Z,

$$|\Pr[\mathrm{IDEAL}_{\mathsf{F},\mathsf{S},\mathsf{Z}}(k,z)=1] - \Pr\left[\mathrm{HYB}^{\mathsf{G}}_{\pi,\mathsf{A},\mathsf{Z}}(k,z)=1\right]| < negl(k).$$

3 Our Models for GGM/GRM and the Non-Commiting Protocol

Here we introduce the *programmable* and *non-programmable* models of GRM and GGM.

As in [21], the technical difference is as follows: In the programmable setting, Z does not have access to the generic functionality, whereas it does in the non-programmable setting. Moreover, let the functionality G execute the routines described in Definition 2.2 for GGM (resp. R and in Definition 2.7 for GRM).

3.1 Our Models

We now formalize the models.

PROGRAMMABLE-GGM (RESP. GRM). This is the regular GGM (resp. GRM). All parties of protocol π have access to the generic group oracle G (resp. R), and we say that π securely realizes some functionality F in the programmable GGM (resp. GRM), if π *securely realizes* F *in the* G*-hybrid model* (resp. R*-hybrid model*).

[5] Which are Turing machines equipped with three extra tapes: (1) The input tape, (2) the communication tape, and (3) the subroutine output tape.

NON-PROGRAMMABLE-GGM (RESP. GRM) In this variant, the environment Z has access to the generic operation oracle G (resp. R). Thus, any computation made by the parties of π can be carried out by the environment itself. It is said that π *securely realizes some functionality* F *in the non-programmable-GGM* (resp. non-programmable-GRM).

3.2 The Non-Committing Encryption Protocol

We define the non-committing protocol π by adding an identification header in the ciphertexts. That is three unique strings id_m, id_{SEN} and id_{REC}. Let π be defined as follows:

send: On input $(\texttt{send}, id_m, id_{REC}, m)$, *SEN* computes $c = enc^{\mathsf{G}}(\mathsf{pk}, id_m||id_{SEN}||id_{REC}||m)$ and sends (id_m, c) to *REC*.

receive: If *REC* receives (id_m, c), it computes $dec^{\mathsf{G}}(\mathsf{sk}, c) = id_m||id_{SEN}||id_{REC}||m$ and outputs $(\texttt{receive}, id_m, id_{SEN}, m)$.

Let $l_h(k)$ be the size in bits of $id_m||id_{SEN}||id_{REC}$, while $l_m(k)$ is the size of m in bits regarding the security parameter k. The negative result from [21] is related to non-committing schemes that are modeled through the functionality $\mathsf{F}_{\texttt{nce}}$, and which we now review.

3.3 The Non-Committing Encryption Functionality $\mathsf{F}_{\texttt{nce}}$

The functionality can be described for the protocol $\tilde{\pi} = (\widetilde{SEN}, \widetilde{REC})$ in the IDEAL execution as follows:

1. $\mathsf{F}_{\texttt{nce}}$ sends $(\texttt{send}, id_m, id_{SEN}, id_{REC}, l_h(k) + l_m(k))$ to S, whenever $\widetilde{SEN}$ inputs $(\texttt{send}, id_m, id_{REC}, m)$ to $\mathsf{F}_{\texttt{nce}}$.

2. $\mathsf{F}_{\texttt{nce}}$ waits S schedule the delivery of $(\texttt{receive}, id_m, id_{SEN}, m)$ from $\mathsf{F}_{\texttt{nce}}$ to $\widetilde{REC}$.

3. When S allows it, $\mathsf{F}_{\texttt{nce}}$ sends $(\texttt{receive}, id_m, id_{SEN}, m)$ to $\widetilde{REC}$.

4 The ElGamal Encryption Scheme in the Programmable GGM

Here we review the ElGamal encryption scheme and study its security in the programmable GGM.

4.1 The ElGamal Encryption Scheme

Let $[m_{\mathbb{G}}]$ be the pointer to the group element $m_{\mathbb{G}} \in \mathbb{G}$ in the simulation of G, where $\mathbb{G}$ is a group of k-bit long prime order p generated by g, both outputted by a generation algorithm *gen*. There is no need to specify the size of the message $|m_{\mathbb{G}}|$ because it is known that in the ElGamal's case the message is encoded as a group member. Therefore, we assume the existence of an encoding function $e : \{0,1\}^{log_2 p} \rightarrow \mathbb{G}$.

Note that in GGM the computation $e^{\mathsf{G}}(id_m||id_{SEN}||id_{REC}||m)$ gives the pointer $[m_{\mathbb{G}}]$, where $1 < m_{\mathbb{G}} \leq p-1$. Moreover, $m_{\mathbb{G}}$ does not depend on the secret key, and $[m_{\mathbb{G}}]$ must have been delivered by G.

The scheme is described in Table 2.

$gen(1^k)$	$enc(\mathsf{pk}, m_{\mathbb{G}})$	$dec(\mathsf{sk}, c)$
$(g,p) \leftarrow \mathsf{gen}(1^k)$	$(g, g^x) \leftarrow \mathsf{pk}$	$(g,x) \leftarrow \mathsf{sk}$
$x \xleftarrow{U} \mathbb{Z}_p^*$	$r \xleftarrow{U} \mathbb{Z}_p^*$	$y \leftarrow (c_1)^x$
$\mathsf{pk} \leftarrow (g, g^x)$	$c_1 \leftarrow g^r$	$m_{\mathbb{G}} \leftarrow c_2 \cdot y^{-1}$
$\mathsf{sk} \leftarrow (g,x)$	$c_2 \leftarrow (g^x)^r \cdot m_{\mathbb{G}}$	return $m_{\mathbb{G}}$
return$(\mathsf{pk}, \mathsf{sk})$	$c \leftarrow (c_1, c_2)$	
	return c	

Table 2: The ElGamal encryption scheme.

4.2 The Security of the ElGamal Encryption Scheme in the Programmable GGM

We show that our protocol π based on the ElGamal encryption scheme is non-committing in the programmable GGM in the presence of the adversary who corrupts the receiver and the sender and expose the secret key sk.

Theorem 4.1. *Let k be the security parameter and p a k-bit long prime. Given the protocol $\pi = (REC, SEN)$ which allows every party to send $n(k)$ messages, there is a simulator S such that for every environment Z, and every adversary A which can corrupt REC and SEN and makes $q(k)$ queries to the generic group operation oracle G, then*

$$|\Pr[IDEAL_{\mathsf{F},\mathsf{S}^{\mathsf{G}},\mathsf{Z}^{\mathsf{G}}}(k,z) = 1] - \Pr[HYB^{\mathsf{G}}_{\pi,\mathsf{A},\mathsf{Z}}(k,z) = 1]| \leq \frac{q(k)^2 \cdot (n(k)+1)}{p} + negl(k),$$

for the probabilities taken over the random choices of the environment Z in the programmable GGM.

Proof.
Overview. We construct a simulator S which simulates the ideal execution for the adversary A including the generic group operation oracle G_{PRO}. In comparison to G from Definition 2.2, G_{PRO} does not have a fixed challenge value x. Instead, it leaves as random variable X unless the adversary corrupts the receiver. Upon this event, G_{PRO} fixes X with a random x. We show that the probability of Z in distinguishing the hybrid and the ideal executions.

First, consider that the correctness of the scheme gives

$$\Pr[\mathrm{HYB}^{\mathsf{G}}_{\pi,\mathsf{A},\mathsf{Z}}(k,z) = 1] = 1 - negl(k),$$

for the random choices of the environment Z.

We now construct a simulator S for the ideal execution $\mathrm{IDEAL}_{\mathsf{F},\mathsf{S}^{\mathsf{G}},\mathsf{Z}^{\mathsf{G}}}(k,z)$. The simulator S must handle two situations: (1) when $\mathsf{F}_{\mathtt{nce}}$ sends $(\mathtt{send}, id_m, id_{SEN}, id_{REC})$ and (2) when SEN sends $(id_m, [c])$ to REC in the simulation of the hybrid execution.

Send: **(1)** S must deliver to A $(id_m, [c])$ whenever it receives $(\mathtt{send}, id_m, id_{SEN}, id_{REC})$ from $\mathsf{F}_{\mathtt{nce}}$.

- If $\widetilde{REC}$ is corrupted:
  - S allows $(\mathtt{receive}, id_m, id_{SEN}, m_{\mathbb{G}})$ to be sent to $\widetilde{REC}$.
  - Since $\widetilde{REC}$ is corrupted, S learns $m_{\mathbb{G}}$. Then it computes $c = enc^{\mathsf{G}_{PRO}}(\mathsf{pk}, [m_{\mathbb{G}}])$ and delivers $(id_m, [c])$ to A.
- If $\widetilde{REC}$ is honest:

– S does not know $[m_{\mathbb{G}}]$, therefore it adds to its simulation of G_{PRO} (as in Definition 2.2) two random variables L_{c_1} and L_{c_2} both sampled from $\mathbb{Z}_p^*$.
– S gives A the pointers $[L_{c_1}]$ and $[L_{c_2}]$, as it would normally do if it knew the values for L_{c_1} and L_{c_2}.
– S continues simulating G_{PRO}, now with X, L_{c_1} and L_{c_2} as unknowns.

If at a later moment *SEN* or *REC* is corrupted:

– If *SEN* was corrupted
  * S corrupts $\widetilde{SEN}$ and learns $[m_{\mathbb{G}}]$, and therefore $m_{\mathbb{G}}$.
  * S partially commits to the ciphertext by choosing uniformly at random $1 \leq r < p-1$. If there is some i such that $L_i = r$, then S sets $L_{c_1} \leftarrow L_i$. Otherwise S sets $L_{c_1} \leftarrow r$. Therefore the partially committed ciphertext $[c] = ([L_{c_1}], [L_{c_2}])$ is

$$L_{c_1} \leftarrow r, L_{c_2} \leftarrow r \cdot X + m_{\mathbb{G}},$$

    with X still as an unknown.
  * S delivers r and $[m_{\mathbb{G}}]$ to A as its internal view of *SEN*.
– If *REC* was corrupted
  * S corrupts $\widetilde{REC}$ and learns $[m_{\mathbb{G}}]$ and therefore $m_{\mathbb{G}}$.
  * S chooses the secret key x uniformly at random from $\mathbb{Z}_p^*$.
  * Then S commits to the correct ciphertext by choosing uniformly at random $1 \leq r < p-1$ and fixing $X \leftarrow x$ and $m_{\mathbb{G}}$,

$$L_{c_1} \leftarrow r, L_{c_2} \leftarrow r \cdot x + m_{\mathbb{G}}.$$

  * S aborts if the simulation was not perfect. Let this event be denoted by abort.
  * If S did not abort, it delivers x and $[m_{\mathbb{G}}]$ to A as its internal view of *REC*.

Receive: **(2)** S must make $\mathsf{F}_{\mathtt{nce}}$ output $(\mathtt{receive}, id_m, id_{SEN}, [m_{\mathbb{G}}])$ to $\widetilde{REC}$, whenever *SEN* sends $(id_m, [c])$ to *REC*.

- If *SEN* is honest
  Then $(id_m, [c])$ was sent by S in response to what happen in the ideal world. Then, S must have received $(\mathtt{send}, id_m, id_{SEN}, id_{REC})$ from $\mathsf{F}_{\mathtt{nce}}$. Therefore, S allows $(\mathtt{receive}, id_m, id_{SEN}, [m_{\mathbb{G}}])$ to be sent to $\widetilde{REC}$ in the ideal execution.
- If *SEN* is corrupted
  – S looks for $[L_{c_1}]$ and $[L_{c_2}]$ such that $[c] = ([L_{c_1}], [L_{c_2}])$ and it must find two polynomials L_{c_1} and L_{c_2} on X.
  – S creates a pointer $[m_{\mathbb{G}}]$ associated with the polynomial $L_{c_2} - L_{c_1} \cdot X$ (as in the decrypting algorithm). Note that the addition of $[m_{\mathbb{G}}]$ adds one degree in the polynomials carried by G_{PRO} every time *SEN* sends a new pair $(id_m, [c])$. Therefore, the degree can be at most $n(k)+1$, where $n(k)$ is the number of messages a single party sends.
  – S inputs $(id_m, id_{REC}, [m_{\mathbb{G}}])$ in $\widetilde{SEN}$, since $\widetilde{SEN}$ is also corrupted.
  – S allows $\mathsf{F}_{\mathtt{nce}}$ to send $(\mathtt{receive}, id_m, id_{SEN}, [m_{\mathbb{G}}])$ to $\widetilde{REC}$.

It remains to analyze the abort probability of S. If the simulation does not abort the views of the hybrid and ideal executions are indistinguishable. Therefore, let us assume for the sake of contradiction that the simulation aborts on event abort. This event occurs when some polynomial in the simulation are different, however they evaluate to the same value after fixing L_{c_1} and L_{c_2} with x chosen uniformly at random.

Recall that on the event abort the simulated oracle G_{PRO} has at most $q(k)$ polynomials: $L_1, \ldots, L_{q(k)}$. Lemma 2.1 gives that the polynomial defined as $L_i - L_j$, for $1 \leq i, j \leq q(k)$, $i \neq j$, has $\Pr[(L_i - L_j)(x) = 0] \leq \frac{n(k)+1}{p}$ for uniformly random choices of $x \leftarrow \mathbb{Z}_p$. Since the total degree of $(L_i - L_j)$ can be at most $n(x)+1$, therefore,

$$\begin{aligned}
\Pr[\mathsf{abort}] &= \Pr_{\substack{1 \leq i,j \leq q(k) \\ i \neq j}}[(L_i - L_j)(x) = 0 | x \xleftarrow{U} \mathbb{Z}_p] \\
&= q(k) \cdot (q(k)-1) \cdot \Pr[L(x) = 0 | x \xleftarrow{U} \mathbb{Z}_p] \\
&\overset{Lemma 2.1}{\leq} q(k) \cdot (q(x)-1) \cdot \frac{n(k)+1}{p} \\
&\leq \frac{q(k)^2 \cdot (n(k)+1)}{p},
\end{aligned}$$

where L is a polynomial with total degree $n(k)+1$ and probabilities are taken over the choices of x. That is

$$\Pr[\mathrm{IDEAL}_{\mathsf{F},\mathsf{S}^{\mathsf{G}},\mathsf{Z}^{\mathsf{G}}}(k,z) = 1] \leq 1 - \frac{q(k)^2 \cdot (n(k)+1)}{p},$$

thereby giving the theorem. □

5 The Goldwasser-Micali Encryption Scheme in the Programmable GRM

Here we review the Goldwasser-Micali (GM) encryption scheme and study its security in the programmable GRM.

5.1 The Goldwasser-Micali Encryption Scheme

First, let the algorithm gen denote the generation algorithm which gives two large prime numbers p and q, and their product N. Let QR_N denote the set of the quadratic residue modulo N, while QNR_N denote the set of non-residues. Moreover, let QNR_N^+ denote the set of the quadratic non-residues which the Jacobi symbol modulo N gives $+1$. That is, given $x \in \mathsf{QNR}_N$, $\mathsf{J}_N(x) = +1$.

For the scheme, consider that m_i, $1 \leq i \leq l_m(k)$, are the bits of the message m, while $c_i \in \mathbb{Z}_N$ and $r_i \in \{0,1\}^{l_r(k)}$ are respectively the ciphertexts and the randomness for each bit.

The scheme is described in Table 3.

Remark 5.1. The ciphertext $c = (c_1, \ldots, c_{l_m(k)})$ of a message $m \in \{0,1\}^{l_m(k)}$ is composed by c_i which are members of the ring. Therefore they are represented in the model as SLP-steps. The public key pk is the SLP's $P_0(x)$ and $P_{-1}(x)$ as in Definition 2.3.

5.2 The Security of the GM Encryption Scheme in the Programmable GRM

We show that our protocol π for GM is non-committing in the programmable GRM even in the presence of the adversary who corrupts the receiver and expose the secret key sk.

$gen(1^k)$	$enc(\mathsf{pk}, m)$	$dec(\mathsf{sk}, c)$
$(p,q,N) \leftarrow \mathsf{gen}(1^k)$	$(N,x) \leftarrow \mathsf{pk}$	$(p,q) \leftarrow \mathsf{sk}$
$x \xleftarrow{\text{U}} \mathsf{QNR}_N^+$	For $i = 1$ to $l_m(k)$	For $i = 1$ to $l_m(k)$
$\mathsf{pk} \leftarrow (N,x)$	$r_i \xleftarrow{\text{U}} \mathbb{Z}_N^*$	If $\mathsf{J}_p(c_i) = +1$
$\mathsf{sk} \leftarrow (p,q)$	$c_i \leftarrow r_i^2 \cdot x^{m_i}$	and $\mathsf{J}_q(c_i) = +1$
return $(\mathsf{pk}, \mathsf{sk})$	return c	$m_i \leftarrow 0$
		Else $m_i \leftarrow 1$
		return m

Table 3: The operations are modulo N

Theorem 5.2. *Let k be the security parameter. Given the protocol $\pi = (REC, SEN)$ there is a simulator* S *such that for every environment* Z*, and every adversary* A *which can corrupt the receiver party REC and makes at most $q(k)$ queries to the generic ring operation oracle* R*, then*

$$|\Pr[IDEAL_{\mathsf{F},\mathsf{S}^{\mathsf{G}},\mathsf{Z}^{\mathsf{G}}}(k,z) = 1] - \Pr[HYB^{\mathsf{G}}_{\pi,\mathsf{A},\mathsf{Z}}(k,z) = 1]| \leq 2q(k) \cdot (q(k)^2 + 5q(k) + 3) \cdot \varepsilon_{FACT}$$

and probability ε_{FACT} of finding a factor of N in the programmable GRM.

Proof.
Overview. We construct a simulator S which simultaneously tries to factor the modulus N and simulates R and interacts with the ideal execution with $\mathsf{F}_{\mathtt{nce}}$. The simulator generates SLP-steps $P^{(m_i)}(x)$ as the ciphertext, and provides A with the pointers $[P^{(m_i)}(x)]$ to them until learning the message m.

The simulator provides A with the simulation of the operation oracle R_{PRO} when simulating the ciphertexts. After receiving m, S commits to the given SLP-steps with the message m, and instead of R_{PRO} simulates again R. We show that the probability of S in simulating the ideal execution is upper bounded by the probability of factoring N.

A challenger runs $gen(1^k) \rightarrow (p,q,N)$, and gives S the value N. The simulator S is allowed to give up the value, and then receives (p,q) and loses the game. It wins if outputs a value y such that $gcd(y,N) \neq \{N,1\}$. Denote the success probability in the game $\varepsilon_{\text{FACT}}$.

We need to define a simulated oracle R_{PRO}. The simulated oracle R_{PRO} is the same as R, however instead of implementing **test** and **equal**, R_{PRO} implements **testpro** and **equalpro**, which are defined in Table 4.

We present the construction for S. Initially it receives the modulus N and picks $x \xleftarrow{\text{U}} \mathsf{QNR}_N^+$ and runs R as in Definition 2.7 with $P_{-1}(x) = x$ and $P_0(x) = 1$. The simulator must handle two situations: (1) when $\mathsf{F}_{\mathtt{nce}}$ sends $(\mathtt{send}, id_m, id_{SEN}, id_{REC}, l_h(k) + l_m(k))$ and (2) when SEN sends $(id_m, [c])$ to REC in the simulation of the hybrid execution.

Send: **(1)** S receives $(\mathtt{send}, id_m, id_{SEN}, id_{REC}, l_h(k) + l_m(k))$ from $\mathsf{F}_{\mathtt{nce}}$.

- If $\widetilde{REC}$ is corrupted:
  - S allows $(\mathtt{receive}, id_m, id_{SEN}, [m])$ to be sent to $\widetilde{REC}$.
  - Since $\widetilde{REC}$ is corrupted, S learns m. Then it computes $c = enc^{\mathsf{R}}(\mathsf{pk}, id_m || id_{SEN} || id_{REC} || m)$ and delivers $(id_m, [c])$ to A, where $[c]$ are the pointers to the ciphertexts c_i.
- If $\widetilde{REC}$ is honest:
  - S adds to the sequence P of its simulation of R_{PRO} with $P^{(m_i)}(x) = R_{l_h(k)+i}$ and $1 \leq i \leq l_m(k)$, where $R_{l_h(k)+i}$ are random variables sampled uniformly at random from $\mathbb{Z}_N^*$.

testpro$(j,\circ)$:
1. It takes a tuple
$(j,\circ) \in \{-1,\ldots,|P|\} \times \{+,-,\times,/\}$
2. Pick $x_r \xleftarrow{U} \mathbb{Z}_N^*$ and
3. If $l_m(k)$ is known, then
pick $R_{l_h(k)+r} \xleftarrow{U} \mathbb{Z}_N^*, 1 \leq r \leq l_m(k)$
4. If $\circ = \{\ /\ \}$ and
$P_j(x_r, R_{l_h(k)+1},\ldots,R_{l_h(k)+l_m(k)}) \notin \mathbb{Z}_N^*$,
then
return false
Else
return true

equalpro(j,i):
1. It takes a tuple
$(j,i) \in \{-1,\ldots,|P|\} \times \{-1,\ldots,|P|\}$
2. Pick $x_r \xleftarrow{U} \mathbb{Z}_N^*$ and
3. If $l_m(k)$ is known, then
pick $R_{l_h(k)+r} \xleftarrow{U} \mathbb{Z}_N^*, 1 \leq r \leq l_m(k)$
4. If $P_j(x_r, R_{l_h(k)+1},\ldots,R_{l_h(k)+l_m(k)}) \equiv_N$
$P_i(x_r, R_{l_h(k)+1},\ldots,R_{l_h(k)+l_m(k)})$ then
return true
Else
return false

Table 4: Definition of the routines **testpro** and **equalpro** for the size of the heading $l_h(k)$ for protocol π and the simulated ciphertexts $R_{l_h(k)+r}$.

- S computes $enc^{\mathsf{G}}(\mathsf{pk}, id_m||id_{SEN}||id_{REC}) = (c_1,\ldots,c_{l_h(k)})$.
- S gives A the access to the pointers of the simulated ciphertext, i.e., $[c] = [c_1]$, …, $[c_{l_h(k)}]$, $[P^{(m_1)}(x)]$, …, $[P^{(m_{l_m(k)})}(x)]$ through its simulation of R_{PRO}.

If at a later moment *REC* is corrupted:

- If *REC* was corrupted
  - S corrupts $\widetilde{REC}$ and learns $m = (m_1,\ldots,m_{l_m(k)})$.
  - S commits to the correct ciphertext by computing $R_{l_h(x)+i} \leftarrow r^2_{l_h(k)+i} \cdot x^{m_i}$, for $r_{l_h(k)+i} \xleftarrow{U} \mathbb{Z}_N^*$ and $1 \leq i \leq l_m(k)$.
  - S aborts if the simulation was not perfect. Let this event be denoted as abort.
  - If S did not abort, then it sends a give up request on N to its challenger in the factoring game, and receives p and q.
  - S uses its simulation of R_{PRO} and $P_0(x) = 1$ to compile p and q as SLP's. In other words, S defines $P^{(p)}(x) := p$ and $P^{(q)}(x) := q$ and adds both to the sequence P.
  - S gives A the access to the pointers to the SLP-steps associated with p and q, i.e., $[P^{(p)}(x)]$ and $[P^{(q)}(x)]$, as the secret key sk.

If S does not abort, then it continues simulating R_{PRO}, however with the fixed values $P^{(m_i)}(x) = r^2_{l_h(k)+i} \cdot x^{m_i}$.

Receive: (2) S must make $\mathsf{F}_{\mathtt{nce}}$ output $(\mathtt{receive}, id_m, id_{SEN}, [m])$ to $\widetilde{REC}$, when *SEN* sends $(id_m, [c])$ to *REC*.

- If *SEN* is honest
Then $(id_m, [c])$ was sent by S in response to what happen in the ideal world. Then, S must have received $(\mathtt{send}, id_m, id_{SEN}, id_{\widetilde{REC}}, l_h(k) + l_m(k))$ from $\mathsf{F}_{\mathtt{nce}}$. Therefore, S allows $(\mathtt{receive}, id_m, id_{SEN}, [m])$ to be sent to $\widetilde{REC}$ in the ideal execution.

Since we are not considering corruption of the receiver, this construction is enough.

If the simulation does not abort, then it is indistinguishable from the real case. Therefore, let us assume for the sake of contradiction that the simulation aborts. The event abort occurs when the simulated oracle R_{PRO} does not behave like R. That is, whenever either one of the two following events happens:

1. when **testpro** does not behave as **test**, or
2. when **equalpro** does not behave as **equal**.

This translates in

$$\Pr[\mathsf{abort}] \leq \Pr[\mathsf{abort}_{test}] + \Pr[\mathsf{abort}_{equal}], \tag{1}$$

where abort_{test} and abort_{equal} are the events (1) and (2) respectively.

We estimate the probabilities $\Pr[\mathsf{abort}_{test}]$ and $\Pr[\mathsf{abort}_{equal}]$ separately[6]. From this point, for convenience of the reader, assume that X are the values sampled by R_{PRO} on every operation. That is $X = (x_r, R_{l_h(k)+1}, \ldots, R_{l_h(k)+l_m(k)})$.

Using the Lemma 2.4, a fixed SLP P_i and the committed values $X_c = (x, r^2_{l_h(k)+1} \cdot x^{m_1}, \ldots, r^2_{l_h(k)+l_m(k)} \cdot x^{m_{l_m(k)}})$, we have

$$\begin{aligned}
\Pr[\mathsf{abort}_{test}] &\leq \Pr[(P_i(X_r) \notin \mathbb{Z}_N^* \textit{ and } P_i(X_c) \in \mathbb{Z}_N^*) \textit{ or} \\
&\qquad (P_i(X_c) \notin \mathbb{Z}_N^* \textit{ and } P_i(X_r) \in \mathbb{Z}_N^*) | X_1, \ldots, X_{q(k)} \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}] \\
&\leq 2q(k) \cdot \Pr[P_i(X) \notin \mathbb{Z}_N^* \textit{ and } P_i(X') \in \mathbb{Z}_N^* |\, X, X' \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}].
\end{aligned}$$

For, $1 \leq i \leq q(x)$, that is, for any $P_i \sqsubseteq P$, we have

$$\begin{aligned}
\Pr[\mathsf{abort}_{test}] &\leq 2q(k) \cdot \sum_{i=0}^{q(k)} \Pr[P_i(X) \notin \mathbb{Z}_N^* \textit{ and } P_i(X') \in \mathbb{Z}_N^* | X, X' \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}] \\
&\leq 2q(k) \cdot (q(k)+1) \cdot \max_{0 \leq i \leq q(k)} \{\Pr[P_i(X) \notin \mathbb{Z}_N^* \textit{ and} \\
&\qquad P_i(X') \in \mathbb{Z}_N^* \mid X, X' \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}\}.
\end{aligned} \tag{2}$$

Similarly with probability $\Pr[\mathsf{abort}_{equal}]$, we have

$$\begin{aligned}
\Pr[\mathsf{abort}_{equal}] \leq\ & 2q(k) \cdot (q(k)^2 + 3q(k) + 1) \cdot \max_{-1 \leq i \leq j \leq q(k)} \{\Pr[P_i(X) \equiv_N P_j(X) \textit{ and} \\
& P_i(X') \not\equiv_N P_j(X') \mid X, X' \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}]\} + \Pr[\mathsf{abort}_{test}].
\end{aligned} \tag{3}$$

Equation 3 relates to the similar test from Lemma 2.5, that is a test which can be made sampling randomly from $\mathbb{Z}_N^*$. Similarly, Equation 2 relates to Lemma 2.6. Both tests are used in the factoring algorithm to be presented next.

We now construct the factoring algorithm.

Let FACTOR be the factoring algorithm. The algorithm evaluates every SLP in the sequence P kept by R_{PRO}. FACTOR will run both tests from Lemmas 2.5 and 2.6, that is

1. Whenever R_{PRO} receives the r-th query operation $(i, j, \circ)$ with $\circ \in \{+, -, \times, /\}$, FACTOR samples $Y = (y_1, \ldots, y_{l_m(k)+1}) \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}$ and computes $gcd(P_n(Y), N)$ for $0 \leq n \leq r$ (Lemma 2.5).
2. Whenever R_{PRO} receives the r-th equality test $(i, j, \circ)$ with $\circ \in \{=\}$, FACTOR samples $Y = (y_1, \ldots, y_{l_m(k)+1}) \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}$ and computes $gcd(P_i(Y) - P_j(Y), N)$ for $-1 \leq i < j \leq r$ (Lemma 2.6).

[6]These probabilities for single values x are estimated in detail in [16].

Assume that γ is the maximum value between the following two probabilities from Equations 3 and 2:

$$max_{0\leq i\leq q(k)}\{\Pr[P_i(X)\notin\mathbb{Z}_N^* \text{ and } P_i(X')\in\mathbb{Z}_N^* \mid X,X' \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}\}$$

and

$$max_{-1\leq i\leq j\leq q(k)}\{\Pr[P_i(X)\equiv_N P_j(X) \text{ and } P_i(X')\neq_N P_j(X') \mid X,X' \xleftarrow{U} (\mathbb{Z}_N^*)^{l_m(k)+1}]\}.$$

Then, the algorithm FACTOR, and the Lemmas 2.5 and 2.6 give $\gamma\leq\varepsilon_{\text{FACT}}$.

The maximum value γ upper bounds the failure probability

$$\begin{aligned}\Pr[\mathsf{abort}] &\leq 4q(k)\cdot(q(k)+1)\cdot\gamma+2q(k)\cdot(q(k)^2+3q(k)+1)\cdot\gamma\\ &\leq 2q(k)\cdot(q(k)^2+5q(k)+3)\cdot\gamma,\end{aligned}$$

and Equation 1 upper bounds the probability of the event abort

$$\Pr[\mathsf{abort}]\leq 2q(k)\cdot(q(k)^2+5q(k)+3)\cdot\varepsilon_{\text{FACT}},$$

and this gives the end of the proof. □

6 Acknowledgments

We would like to thank the anonymous reviewers for pointing out the parts of the text which needed clarification in the early draft of this work.

References

[1] Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. In *Proc. of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'09), Cologne, Germany, LNCS*, volume 5479, pages 36–53. Springer-Verlag, April 2009.

[2] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *Proc. of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'09), Cologne, Germany, LNCS*, volume 5479, pages 1–35. Springer-Verlag, April 2009.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. of the 1st ACM conference on Computer and communications security (CCS'93), Fairfax, Virginia, USA*, pages 62–73. ACM Press, November 1993.

[4] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In *Proc. of the 17th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'98), Espoo, Finland, LNCS*, volume 1403, pages 59–71. Springer-Verlag, May-June 1998.

[5] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Technical Report 16, Electronic Colloquium on Computational Complexity, Watson Research Center, 2001.

[6] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proc. of the 28th annual ACM symposium on Theory of computing (STOC'96), Philadephia, Pennsylvania, USA*, pages 639–648. ACM Press, May 1996.

[7] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, July 2004.

[8] Jean-Sébastien Coron. On the exact security of full domain hash. In *Proc. of the 20th Annual International Cryptology Conference (CRYPTO'00), Santa Barbara, California, USA, LNCS*, volume 1808, pages 229–235. Springer-Verlag, August 2000.

[9] Alexander Dent. Adapting the weaknesses of the random oracle model to the generic group model. In *Proc. of the 22nd Annual International Cryptology Conference (CRYPTO'02), Santa Barbara, California, USA, LNCS*, volume 2501, pages 95–104. Springer-Verlag, August 2002.

[10] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In *Proc. of the 25th Annual International Cryptology Conference (CRYPTO'05), Santa Barbara, California, USA, LNCS*, volume 3621, pages 449–466. Springer-Verlag, August 2005.

[11] Marc Fischlin. A note on security proofs in the generic model. In *Proc. of the 6th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'00), Kyoto, Japan, LNCS*, volume 1976, pages 458–469. Springer-Verlag, December 2000.

[12] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In *Proc. of the 16th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'10), Singapore, LNCS*, volume 6477, pages 303–320. Springer-Verlag, December 2010.

[13] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In *Proc. of the 28th Annual International Cryptology Conference (CRYPTO'08), Santa Barbara, CA, USA, LNCS*, volume 5157, pages 21–38. Springer-Verlag, August 2008.

[14] Tibor Jager and Andy Rupp. The semi-generic group model and applications to pairing-based cryptography. In *Proc. of the 30th Annual Cryptology Conference (CRYPTO'10), Santa Barbara, CA, USA, LNCS*, volume 6477, pages 539–556. Springer-Verlag, August 2010.

[15] Tibor Jager and Jörg Schwenk. On the equivalence of generic group models. In *Proc. of the 2nd International Conference on Provable Security (ProvSec'08), Shanghai, China, LNCS*, volume 5324, pages 200–209. Springer-Verlag, October -November 2008.

[16] Tibor Jager and Jörg Schwenk. On the analysis of cryptographic assumptions in the generic ring model. In *Proc. of the 15th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'09), Tokyo, Japan, LNCS*, volume 5912, pages 399–416. Springer-Verlag, December 2009.

[17] Neal Koblitz and Alfred Menezes. Another look at generic groups. Technical Report 1, Advances in Mathematics of Communications, University of Washington, 2006.

[18] Ueli Maurer. Abstract models of computation in cryptography. In *Proc. of the 10th IMA International Conference (Cryptography & Coding'05), Cirencester, UK, LNCS*, volume 3796, pages 1–12. Springer-Verlag, Dcember 2005.

[19] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Proc. of the 1st Theory of Cryptography Conference (TCC'04), Cambridge, MA, USA, LNCS*, volume 2951, pages 21–39. Springer-Verlag, February 2004.

[20] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, February 1994.

[21] Jesper Buus Nielsen and Ny Munkegade. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Proc. of the 22nd Annual International Cryptology Conference (CRYPTO'02), Santa Barbara, California, USA, LNCS*, volume 2442, pages 111–126. Springer-Verlag, August 2002.

[22] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[23] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Proc. of the 16th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'97), Konstanz, Germany, LNCS*, volume 1233, pages 256–266. Springer-Verlag, May 1997.

[24] Hoeteck Wee. Zero knowledge in the random oracle model, revisited. In *Proc. of the 15th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'09), Tokyo, Japan, LNCS*, volume 5912, pages 417–434. Springer-Verlag, December 2009.

Mario Larangeira is graduated as a computer engineer at Aeronautical Institute of Technology in 2004 and worked as a development engineer at Embraer. He received his M.S. from the Tokyo Institute of Technology in 2008 and is currently enrolled in the Department of Mathematical and Computing Sciences of the same institute as a Ph. D. student in the area of computer science.

Keisuke Tanaka is Associate Professor of Department of Mathematical and Computing Sciences at Tokyo Institute of Technology. He received his B.S. from Yamanashi University in 1992 and his M.S. and Ph. D. from Japan Advanced Institute of Sciences and Technology in 1994 and 1997, respectively . For each degree, he majored in computer science. Before joining Tokyo Institute of Technology, he was Research Engineer at NTT Information Plataform Labs.