Sufficient Conditions for the Computational Intractability of Generic Group Problems

Andy Rupp[1], Gregor Leander[1], Endre Bangerter[2], Ahmad-Reza Sadeghi[1], Alexander W. Dent[3]

[1] Horst-Görtz Institute for IT-Security (Germany)
{arupp, sadeghi}@crypto.rub.de, gregor.leander@rub.de
[2] Bern University of Applied Sciences (Switzerland)
endre.bangerter@bfh.ch
[3] Royal Holloway, University of London (United Kingdom)
a.dent@rhul.ac.uk

Abstract. The generic group model is a valuable methodology for analysing the computational hardness some number-theoretic problems used in cryptography. Although generic hardness proofs exhibit many similarities, still the computational intractability of every newly introduced problem needs to be proven from scratch, a task that can easily become complicated and cumbersome when done rigorously. In this paper we make the first steps towards overcoming this problem by identifying verifiable criteria which guarantee the hardness of a problem in the generic group model. As useful means for formalisation of definitions and proofs we relate the concepts of generic algorithms and straight-line programs that have only been used independently in cryptography so far.

Keywords. Generic Group Model, Straight-Line Programs, Hardness Conditions

1 Introduction

The generic group model aims to estimate a bound for the success probability of an algorithm attempting to solve a group-based problem in a manner that is independent of the encoding of the group. These generic algorithms are equally applicable on all finite cyclic groups, from hyper-elliptic curve groups of order n to the group of integers under addition modulo n. Natural examples of this class of algorithm include the Pohlig-Hellman algorithm [PH78] and the Pollard-Rho algorithm [Pol78].

There are two common methods to allow an algorithm to manipulate group elements without revealing the encoding of these group elements. The first was proposed by Nechaev [Nec94] and Shoup [Sho97]. In this model, group elements are represented by random bitstrings and group operations are provided by a series of oracles. The other method was proposed by Maurer [Mau00,Mau05]. In this model, group elements are held in a special series of registers. The algorithm cannot access the data held in these registers but can, once again, manipulate them using a series of oracles. The algorithm is informed if two group elements are equal, but receives no other information. For simplicity, our results are proven in the Maurer generic group model; however, similar techniques can prove identical results in the Nechaev/Shoup generic group model.

It should be noted that a proof of intractability of a problem in the generic group model does not guarantee intractability for that problem in the standard model. Dent [Den02] has proven that there exist problems that are secure in the generic group model but insecure in any group. Furthermore, several difficulties have been shown to exist when applying the generic group methodology to the analysis of cryptographic protocols [Fis00,KM07,SPMLS02].

Nevertheless, it has become common to justify the use of a new intractability assumption with a proof in the generic group model. These proofs can be complicated, particularly if the group structure (e.g. the number of groups, the relations between groups) is complex. This results in a large number of long and difficult proofs being published in the literature.

Our contribution. Most intractability proofs in the generic group model have the same structure. We identify the core aspects of a problem that lead to intractability and provide generic conditions which guarantee the security of a problem in the generic group model. The conditions we supply can be adapted a large class of group structures and a large class of computational problems. Moreover, the conditions that have to be satisfied by the group structure and the conditions that have to be satisfied by the problem are almost entirely independent; thus allowing large sections of a proof to be re-used without needing to be re-proven.

Our results highlight the fundamental basis of proofs of the intractability of problems in the generic group model. In particular, we discuss the relationship between generic algorithms and straight-line programs. We believe that this relationship will be a key factor in developing automatically checkable proofs.

One technique that we use to prove our results is to provide an algorithm that can bound the degrees of Laurent polynomials in different groups. This novel algorithm may be of some academic interest independent of our results on the generic group model.

We note that our work does not consider generic algorithms that can generate group elements at random; however, our results can be trivially extended to cope with this case. This case has not been omitted to improve the clarity of the paper.

Related Work. There are two major papers that give results similar to those presented here. The first is the paper of Kiltz [Kil01] which analyses problems of the form "given (g, g^{x_1}, g^{x_2}), find $g^{P(x_1,x_2)}$, where g is a generator of a group $\mathbb{G}$ and P is a non-linear polynomial". The second is the work of Boneh, Boyen and Goh [BBG05, §A] which considers problems of the form "given $g_1^{p_i(\boldsymbol{x})}$ and $g_2^{q_i(\boldsymbol{x})}$, find $g_1^{f(\boldsymbol{x})}$; where g_1 is a generator of the group $\mathbb{G}_1$; g_2 is a generator of the group $\mathbb{G}_2$; p_i, q_i and f are multi-variate polynomials; and there exists a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$".

Our results improve these works in several ways:

- Our results allow for a much more flexible group structure. Our results include the case of a single group and a simple bilinear map, but also allow for more complex bilinear maps, decision oracles and other computational oracles that may be provided by the problem instances.
- Our results allow for a much more flexible set of problems. Our results cover both discrete logarithm style problems (in which the algorithm must compute a value $f(\boldsymbol{x})$) and Diffie-Hellman style problems (in which the algorithm must compute a group element $g^{f(\boldsymbol{x})}$). Our results also cover the case where the algorithm can output some supplementary information $\boldsymbol{c}$ which affects the form of the polynomial f. As a practical example, our results can be used to prove the security of the β-Strong-Diffie-Hellman problem [BB04]: a problem that does not fit into either the structure of Kiltz or of Boneh, Boyen and Goh.
- Our results highlight the intuitive reasons why certain problems are hard in the generic group model. This may lead to a deeper understanding of the generic group model and provide a step towards the construction of automated proof-verifiers or generators for generic group problems.

2 The Generic Group Model

2.1 Preliminaries

We use standard notations. If $k \in \mathbb{N}$, then $\{0,1\}^k$ is the set of all bit strings of length k and 1^k is the string of k ones. Let $\{0,1\}^*$ be the set of all bit strings. If $\mathcal{A}$ is a randomised algorithm, then $y \overset{\text{R}}{\leftarrow} \mathcal{A}(x)$ denotes the assignment to y of the output of $\mathcal{A}$ run on x with fresh random coins; we write $y \leftarrow \mathcal{A}(x)$ if $\mathcal{A}$ is deterministic. If S is a set, then $x \overset{\text{R}}{\leftarrow} S$ denotes the random generation of an element $x \in S$ using the uniform distribution. A function ν is said to be negligible if all all $c \in \mathbb{N}$, there exists a constant λ_c such that $\nu(\lambda) < 1/k^c$ for all $\lambda > \lambda_c$

We will make use of the ring of finite Laurent polynomials over $\mathbb{Z}_n$. For our purposes, it is convenient to define a Laurent polynomial as a finite sum of monomials of the form $aX_1^{\alpha_1}\dots X_m^{\alpha_m}$ where $a, \alpha_1, \dots, \alpha_m \in \mathbb{Z}_n \setminus \{0\}$. We define the total degree of the monomial to be $\sum |\alpha_i|$ and the total degree $\deg P$ of the Laurent polynomial P to be the maximum total degree of any monomial in the polynomial. We let $\mathcal{L}_n^{(l,c)}$ denote the set of Laurent polynomials over $\mathbb{Z}_n$ in the variables $X_1, \dots, X_\ell$ where only the variables $X_{c+1}, \dots, X_\ell$ can occur with a negative exponent.

We will also make use of rational functions $f \in \mathbb{Z}(X_1, \dots, X_m)$. We may express a rational function f as f_1/f_2 where $f_1, f_2 \in \mathbb{Z}[X_1, \dots, X_m]$. The total degree of f is defined to be the sum of the degrees of f_1 and f_2.

2.2 Problems in the Generic Group Model

We will work exclusively in the generic group model presented by Maurer [Mau00,Mau05]. In this model, an algorithm never explicitly gains access to group elements; instead all group elements that the algorithm is given are held in a special set of registers. The algorithm cannot access the data inside the registers but can manipulate group elements by passing pointers to registers holding group elements to a series of oracles. For example, the algorithm can add two group elements by passing pointers to two registers containing group elements to a special *add* oracle. This oracle adds the two group elements and returns the result into a new register. The only information that the oracle returns to the algorithm is a list of registers that contain the same group element as the returned value. Hence, the algorithm knows when two group elements are equal, but no other information about the group.

We use notation similar to that of Boneh, Boyen and Goh [BBG05, §A]. We will assume a fixed group structure. In other words, we will assume that there are a fixed number of groups $(\mathbb{G}_1, \dots, \mathbb{G}_k)$. Each of the these groups has order $n \overset{\text{R}}{\leftarrow} \texttt{GGen}(\lambda)$ and each group $\mathbb{G}_i$ has a generator g_i. However, we stress that the number of groups is independent of the security parameter λ. We allow the algorithm the ability to compute group operations via access to a series of oracles, known as the operations set. Each operation in this set is a map of the form

$$op : \mathbb{G}_{s_1} \times \mathbb{G}_{s_2} \times \dots \times \mathbb{G}_{s_m} \to \mathbb{G}_t \tag{1}$$

where

$$op(g_{s_1}^{y_1}, g_{s_2}^{y_2}, \dots, g_{s_m}^{y_m}) = g_t^{OP(y_1,y_2,\dots,y_m)} \tag{2}$$

and $OP \in \mathbb{Z}[Y_1, \dots, Y_\ell]$ is a multi-variate polynomial over $\mathbb{Z}$. Note that these operations are defined relative to the fixed bases $g_{s_1}, \dots, g_{s_m}$ and g_t. Note also that the definitions of the operations are independent of the security parameter λ. This is the reason why the polynomials are defined over the infinite ring $\mathbb{Z}$ despite the fact that the actual computations always take place in the finite ring $\mathbb{Z}_n$.

Examples of operations that could be included in the operations set include:

- Addition, which is given by $add : \mathbb{G}_i \times \mathbb{G}_i \to \mathbb{G}_i$ such that $add(g_i^{y_1}, g_i^{y_2}) = g_i^{y_1+y_2}$.
- Inversion, which is given by $inv : \mathbb{G}_i \to \mathbb{G}_i$ such that $inv(g_i^{y_1}) = g_i^{-y_1}$.
- A computable homomorphism from $\phi : \mathbb{G}_i \to \mathbb{G}_j$, which is given by $\phi(g_i^{y_1}) = g_j^{y_1}$.
- A bilinear map $e : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_t$, which is given by $e(g_i^{y_1}, g_j^{y_2}) = g_t^{y_1 y_2}$.

We assume that the algorithm has access to an infinite number of registers, and that the contents of a register, once defined, cannot be altered or erased.

In this paper we will be considering computational problems similar to the Diffie-Hellman and discrete logarithm problem. The problem will consist of a set of input group elements and some specification of the output required to solve the problem. We assume that there is a sequence of private values $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ chosen uniformly at random from $\mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell-c}$ which relate to the given inputs and required output of the problem. For each group $\mathbb{G}_i$ we define a sequence of ℓ-variable Laurent polynomials $P_i = (p_{i,0}, p_{i,1}, \ldots)$ over $\mathbb{Z}_n$; i.e. $p_{i,j} \in \mathcal{L}_n^{(l,c)}$. These define the inputs that the algorithm receives – the algorithm is given (registers containing) the group elements $g_i^{p_{i,j}(\boldsymbol{x})}$ $\forall i, j$. We assume that the form of the polynomials $p_{i,j}$ can depend upon on the security parameter λ and that the number of input elements in each group is bounded by some polynomial $z(\lambda)$.

We let L_i be the ordered set of group elements computed in group $\mathbb{G}_i$. Hence, the list L_i initially contains the group elements $(g_i^{p_{i,0}(\boldsymbol{x})}, g_i^{p_{i,1}(\boldsymbol{x})}, \ldots)$ and can only be extended by applying group operations.

The output that the algorithm is required to give is defined by a *rational* function $f : \mathbb{Z}_n^\ell \times \mathbb{Z}_n^{\ell'} \rightarrow \mathbb{Z}_n$. In a Diffie-Hellman problem, the algorithm is required to output a vector $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$ and a (register containing a) group element $h \in \mathbb{G}_t$ such that $h = g_t^{f(\boldsymbol{x},\boldsymbol{c})}$. Note that the problem specifies the group $\mathbb{G}_t$ that the output must lie within. In a discrete logarithm problem, the algorithm is required to output a vector $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$ and a value $\alpha \in \mathbb{Z}_n$ such that $f(\boldsymbol{x}, \boldsymbol{c}) = \alpha$. We assume that if $f(\boldsymbol{x}, \boldsymbol{c})$ is undefined then the algorithm has not solved the problem.

Examples of problems that can be expressed as discrete logarithm problems include:

– The discrete logarithm problem itself. The private vector $\boldsymbol{x}$ consists of one element $x_1 \in \mathbb{Z}_n$. The input variables are defined by the polynomials $p_{1,0} = 1$ and $p_{1,1}(\boldsymbol{x}) = x_1$; i.e. the algorithm is given the group elements $(g_1, g_1^{x_1})$. The output is defined by the polynomial $f(\boldsymbol{x}) = x_1$; i.e. the algorithm is required to output a value α such that $\alpha = x_1$ in $\mathbb{Z}_n$.

Examples of problems that can be expressed as Diffie-Hellman problems include:

– The Diffie-Hellman problem. The private vector is $\boldsymbol{x} = (x_1, x_2) \in \mathbb{Z}_n^2$. The input variables are defined by the polynomials $p_{1,0}(\boldsymbol{x}) = 1$, $p_{1,1}(\boldsymbol{x}) = x_1$ and $p_{1,2}(\boldsymbol{x}) = x_2$; i.e. the algorithm is given the group elements $(g_1, g_1^{x_1}, g_1^{x_2})$. The output is defined by the polynomial $f(\boldsymbol{x}) = x_1 x_2$ in the group $\mathbb{G}_1$; i.e. the algorithm is required to output a (pointer to a register containing a) group element $g_1^{x_1 x_2}$.
– The β-Strong Diffie-Hellman problem. Suppose the group order n is prime. The private vector is $\boldsymbol{x} = x_1$. The input variables are defined by the polynomials $p_{1,0}(\boldsymbol{x}) = 1$ and $p_{1,z}(\boldsymbol{x}) = x_1^\alpha$ for $1 \leq \alpha \leq \beta$. The output is defined by the polynomials $f(\boldsymbol{x}, \boldsymbol{c}) = g^{1/(x_1+c_1)}$ where $\boldsymbol{c}$ consists of a single element $c_1 \in \mathbb{Z}_n$; i.e. the algorithm must output an integer $c_1 \in \mathbb{Z}_n$ and a group element $g_1^{1/(x_1+c_1)}$.

3 Hardness Conditions

We now present conditions for Diffie-Hellman (DH) and discrete logarithm (DL) problems that guarantee their security in the generic group model.

3.1 Abstract Conditions

As described above, we may represent group elements in a generic group $\mathbb{G}_i$ as a series of polynomials in the variables $\boldsymbol{X} = (X_1, \ldots, X_\ell)$ rather than as a series of group elements defined by the randomly-chosen vector $\boldsymbol{x} = (x_1, \ldots, x_\ell)$. If we recall that L_i denotes the sequence of group elements computed by an algorithm in the group $\mathbb{G}_i$ then the difference becomes clear. In the group encoding, L_i contains a list of group elements

$$L_i = (g_i^{p_{i,0}(\boldsymbol{x})}, g_i^{p_{i,1}(\boldsymbol{x})}, \ldots)\,. \tag{3}$$

In the polynomial encoding, L_i contains a list of polynomials

$$L_i = (p_{i,0}(\boldsymbol{X}), p_{i,1}(\boldsymbol{X}), \ldots)\,. \tag{4}$$

Let $\mathcal{I}_n$ denote the ideal of $\mathcal{L}_n^{(l,c)}$ containing all Laurent polynomials that are effectively zero in $\mathbb{Z}_n$; more precisely,

$$\mathcal{I}_n = \{p \in \mathcal{L}_n^{(l,c)} \mid \forall (x_1, \ldots, x_\ell) \in \mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell-c},\ p(x_1, \ldots, x_\ell) \equiv 0 \bmod n\} \tag{5}$$

We define two polynomials P and P' contained in L_i to be equal if $P - P' \in \mathcal{I}_n$.

We note that this polynomial encoding doesn't completely simulate the group encoding. This is because it is possible that distinct polynomials in L_i represent the same group element for a particular choice of $\boldsymbol{x}$. To use the generic group model, we must find a condition that demonstrates that this can only happen with small probability. We call such an occurrence a *collision*.

Our main observation is that if we fix a random tape, group order n and security parameter λ, then any algorithm attempting to solve a particular problem acts as a straight line program up until the moment a collision occurs. This straight line program specifies the order in which the operations from the operations set are applied to the input polynomials. The choice of random tape can therefore be thought of as selecting a straight line program from a set of alternatives.

Definition 1 (SLP Generator). *A q-SLP-generator $\mathcal{S}$ is a probabilistic, polynomial-time algorithm that takes as input a security parameter λ and a group order $n \overset{R}{\leftarrow} \texttt{GGen}(\lambda)$. It outputs a sequence of lists $(L_1, \ldots, L_k)$ of Laurent polynomials in the variables $\boldsymbol{X} = (X_1, \ldots, X_\ell)$ representing group elements in the groups $(\mathbb{G}_1, \ldots, \mathbb{G}_k)$. These lists are initially populated with the input polynomials $L_i = (p_{i,0}(\boldsymbol{X}), p_{i,1}(\boldsymbol{X}), \ldots)$ but $\mathcal{S}$ can append up to q Laurent polynomials to these lists by applying the operations from the operation set to existing Laurent polynomials.*

The polynomial encoding of the groups is perfect unless a collision occurs. We propose Condition 1 as method to bound the probability that a collision occurs:

Condition 1 (Collision-resistance). *An operations set is (ν, q)-collision-resistant if for all q-SLP-generators $\mathcal{S}$, $n \overset{R}{\leftarrow} \texttt{GGen}(\lambda)$, $\boldsymbol{x} \overset{R}{\leftarrow} \mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell-c}$, and $(L_1, \ldots, L_k) \overset{R}{\leftarrow} \mathcal{S}(n, \lambda)$ we have that*

$$\Pr[\exists i \text{ and } P, P' \in L_i : (P - P')(\boldsymbol{x}) \equiv 0 \text{ mod } n \ \wedge\ P - P' \notin \mathcal{I}_n] \leq \nu(\lambda) \tag{6}$$

where the probability is taken over the random choice of $\boldsymbol{x}$ and the random coins of `GGen` *and $\mathcal{S}$.*

If there are no collisions, then it is easy to stipulate conditions which guarantee that a DL-/DH-type problems is intractable. The condition for DL-type problems is simply that the output polynomial f is non-constant.

Condition 2 (SLP-intractability of DL-type problems). *A DL-type problem is ν-SLP-intractable if for all $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$, and for $n \overset{R}{\leftarrow} \texttt{GGen}(\lambda)$ and $\boldsymbol{x} \overset{R}{\leftarrow} \mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell-c}$, for all $\alpha \in \mathbb{Z}_n$, we have that*

$$\Pr[f(\boldsymbol{x}, \boldsymbol{c}) \equiv \alpha \text{ mod } n] \leq \nu(\lambda) \tag{7}$$

where the probability is taken over the random choice of $\boldsymbol{x}$ and the random coins of `GGen`.

The condition for DH-type problems is that the problem is not trivially solvable by a straight line programs.

Condition 3 (SLP-intractability of DH-type problems). *A DH-type problem is (ν, q)-SLP-intractable if for every q-SLP-generator $\mathcal{S}$, for all $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$, and for $n \xleftarrow{R} \texttt{GGen}(\lambda)$, $\boldsymbol{x} \xleftarrow{R} \mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell-c}$, and $(L_1, \ldots, L_k) \xleftarrow{R} \mathcal{S}(n, \lambda)$ we have that the polynomial $P \in L_t$ representing the output of the algorithm satisfies*

$$\Pr[(P - f(\cdot, \boldsymbol{c}))(\boldsymbol{x}) \equiv 0 \ mod\ n] \leq \nu(\lambda) \tag{8}$$

where the probability is taken over the random choice of $\boldsymbol{x}$ and the random coins of `GGen` *and $\mathcal{S}$.*

Theorem 1 (Intractability of DL-/DH-type Problems).

– *Let $\mathcal{A}$ be an algorithm against a DL-type problem that makes at most q queries to the group oracles in the generic group model. If the DL-type problem is (ν_1, q)-collision-resistant and ν_2-SLP-intractable, then the probability that $\mathcal{A}$ solves the problem is bounded by $\nu_1 + \nu_2$.*
– *Let $\mathcal{A}$ be an algorithm against a DH-type problem that makes at most q queries to the group oracles in the generic group model. If the DH-type problem is (ν_1, q)-collision-resistant and (ν_3, q)-SLP-intractable, then the probability that $\mathcal{A}$ solves the problem is bounded by $\nu_1 + \nu_3$.*

3.2 Practical Conditions: Collision Resistance

In this section, we derive easily checkable conditions to prove collision resistance. We do this by showing that any two distinct Laurent polynomials have a very small probability of being equal when the variables $\boldsymbol{x}$ are chosen at random. This probability is related to the total degrees of the Laurent polynomials. It is therefore imperative that we bound the total degree of the Laurent polynomials that can be computed in each group. This is done via a novel graph algorithm.

We begin by adapting a lemma of Shoup [Sho97]:

Lemma 1. *Let p be the largest prime divisor of n and let $P \in \mathcal{L}_n^{(l,c)}$ be a non-zero Laurent polynomial of total degree d. Then for $\boldsymbol{x} \xleftarrow{R} \mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell-c}$ we have that*

$$\Pr[P(\boldsymbol{x}) \equiv 0 \ mod\ n] \leq \frac{(\ell - c + 1)d}{p - 1} \tag{9}$$

Our aim is to show that the total degree of any polynomial encoding can be bounded by a polynomial. We begin by ensuring that the total degrees of the input polynomials are not too large.

Condition 4. *There exists a polynomial r such that for all λ and $p_{i,j} \in P_i$ we have $\deg p_{i,j} \leq r(\lambda)$.*

We must next ensure that the degree of these polynomials cannot grow super-polynomially due to the operations in the operations set. In order to bound the total degree of any polynomial computable in a group, we represent the operations set as a graph.

The *operations set graph* $G_{OPSet} = (V, E)$ is a directed multi-edge multi-vertex graph. There are four types of vertices, namely group, sum, product and power vertices. Initially, V contains k group vertices labelled with the numbers 1 to k representing the k different groups. Recall that each operation in the operation set can be represented as a map

$$op : \mathbb{G}_{s_1} \times \mathbb{G}_{s_2} \times \ldots \times \mathbb{G}_{s_m} \to \mathbb{G}_t \qquad \text{where} \qquad op(g_{s_1}^{y_1}, g_{s_2}^{y_2}, \ldots, g_{s_m}^{y_m}) = g_t^{OP(y_1, y_2, \ldots, y_m)}$$

and $OP \in \mathbb{Z}[Y_1, \ldots, Y_\ell]$ is a polynomial. We may represent OP as the sum of monomials $OP = \sum OP_i$. We construct the operation set graph as follows:

1. For each operation (polynomial) OP we add a sum vertex Σ and an edge from the sum vertex to the vertex representing the output group.

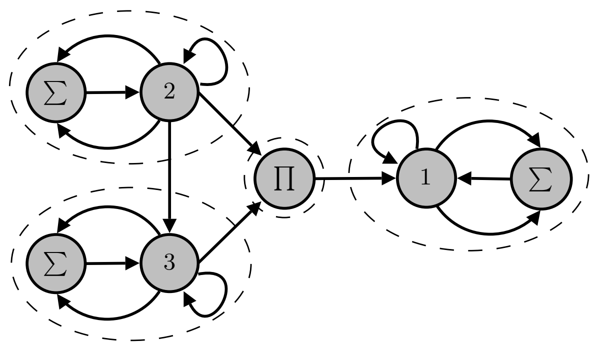

Fig. 1. The reduced operation set graph for a common bilinear map structure. The sum vertices are associated with addition operations. The product vertex is associated with the bilinear map. The edges from a group vertex to itself are associated with the inversion operations. The edge from $\mathbb{G}_2$ to $\mathbb{G}_3$ is associated with the isomorphism. The strongly connected components are marked by dashed borders.

2. For each monomial OP_i in OP we add a product vertex Π and an edge from the product vertex to the sum vertex.
3. For each variable that occurs in the monomial, we add a power vertex $(\cdot)^\alpha$ labelled with the value of the exponent α of that variable. We add edges from the input group vertex to the power vertex, and from the power vertex to the product vertex.

This graph contains a lot of redundancy. The *reduced operation set graph* is the graph where:

– Every power vertex with label $(\cdot)^1$ is removed.
– Every sum and product vertex with indegree 1 is removed.

The two edges associated with each removed vertex are replaced with a single edge going from the predecessor vertex to the successor vertex.

As an example, consider a system of groups with a bilinear map. Suppose there exists three groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$ with the following operations acting on them:

– Each group has an addition function $add_i : \mathbb{G}_i \times \mathbb{G}_i \to \mathbb{G}_i$ defined by $add_i(g_i^{y_1}, g_i^{y_2}) = g_i^{y_1+y_2}$.
– Each group has an inverse function $inv_i : \mathbb{G}_i \to \mathbb{G}_i$ defined by $inv_i(g_i^{y_1}) = g_i^{-y_i}$.
– There exists an isomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_3$ defined by $\phi(g_2^{y_1}) = g_3^{y_1}$.
– There exists a bilinear map $e : \mathbb{G}_2 \times \mathbb{G}_3 \to \mathbb{G}_1$ defined by $e(g_2^{y_1}, g_3^{y_2}) = g_1^{y_1 y_2}$.

The reduced operation set graph for this system of operations is given in Figure 1.

We claim that, in order to eliminate collisions, it is sufficient for the reduced operation set graph to satisfy the following condition:

Condition 5. *Let G_{OPSet} be a reduced operation set graph. For every strong connected component*[4] *C of G_{OPSet} we have:*

1. *If C contains a power vertex, then it contains no other vertices.*
2. *Every product vertex contained in C has at most one incoming edge from a group vertex in C.*

If we have a reduced operation set graph that satisfies Condition 5, then we can bound the total degree of the polynomials that can be computed in each group. We do not give a direct bound, but rather describe an algorithm for computing a bound. The principle of the algorithm is simple. We recursively bound the total degree in each group. This is done by noting that:

[4] A strongly connect component of a directed graph $G = (V, E)$ is a maximal set of vertices $U \subseteq V$ such that there exists a path between every pair of vertices in U.

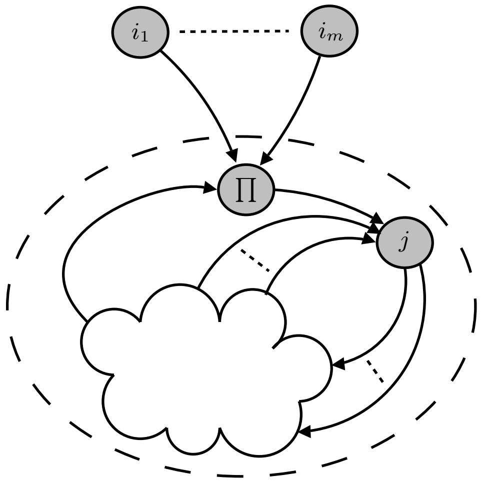

Fig. 2. The structure of a component containing a group vertex and a product vertex. The 'cloud' may contain more group vertices and more product vertices.

- Exponentiation of a polynomial of degree d_s by α results in a polynomial of degree αd_s.
- Multiplication of polynomials of degree d_s and $d_{s'}$ results in a polynomial of degree $d_s + d_{s'}$.
- Addition of polynomials of degree d_s and $d_{s'}$ results in a polynomial of degree $\max\{d_s, d_{s'}\}$.

The algorithm to bound the degree of polynomials computable in each group is given in Table 1.

If we define the positive (respectively negative) degree of a monomial in a Laurent polynomial to be the the sum of the positive (respectively negative) exponents in any monomial, then we can also use the algorithm in Table 1 to bound the positive and negative degrees of any Laurent polynomial that can be computed in any group. Furthermore, we can use the algorithm to bound the positive or negative degree of a single variable X_i in any Laurent polynomial computable in any group.

Theorem 2. *Consider an operation set that satisfies Conditions 5, and suppose that p is the largest prime dividing n, that the number of input polynomials in each group is bounded by z and that $(d_1, \ldots, d_k)$ bound the total degrees of the polynomials computable in each group. Then the operation set is (q, ν)-collision-resistant (Condition 1) where*

$$\nu \leq \frac{(q+z)^2(\ell + c - 1)(d_1 + \ldots + d_k)}{p-1}\,. \tag{10}$$

Proof The number of polynomials contained in L_i is bounded by $z + q$ and for every pair of distinct pair of polynomials $P, P' \in L_i$ satisfying $P - P' \notin \mathcal{I}_n$ we have that the probability that $(P - P')(\boldsymbol{x}) \equiv 0 \bmod n$ for randomly chosen $\boldsymbol{x} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell - c}$ is bounded by $(\ell - c + 1)d_i/(p-1)$. Hence, the probability that any two polynomials cause a collision is bounded by the above bound. □

Corollary 1. *Suppose an operation set satisfies Conditions 4 and 5, that the largest prime divisor of n is a prime p of length λ, and that the number of input polynomials in each group is bounded by a polynomial $z(\lambda)$. Then for polynomially bounded values of q, we have that the operation set is (q, ν)-collision-resistant for a negligible value ν.*

Proof The proof immediately follows from Theorem 2 and the observation that degree-bounding algorithm only increases the input degrees by a polynomial factor. A generic polynomial bound for the total degree of any computable polynomial is given in the full version of the paper. □

Note that these results are for a particular operations set. Once these bounds have been proven for one operations set, they hold for all problems that make use of that operations set.

Table 1. Algorithm to bound the degree of the polynomials computable in a set of groups

Input: A reduced operation set graph G
Output: A list $(d_1, \ldots, d_k)$ where d_i bounds the degree in group $\mathbb{G}_i$

1. Reduce G to a graph G' by removing all sum vertices and connecting all of their predecessor vertices directly to their successor vertex. Also remove all edges that go from a vertex to itself. By Condition 5, the resulting graph only contains four types of component:
   - components containing only group vertices,
   - components consisting of a single power vertex,
   - components consisting of a single product vertex, and
   - components containing both group vertices and product vertices.
2. Compute the graph $\overline{G'}$ of the strongly connected components of G'. An edge exists from one component to another if there exists an edge from one vertex of the component to a vertex in the other component. Note that $\overline{G'}$ is a cycle-free graph.
3. Consider each strongly connected component that has indegree 0 (of which there must be at least one since $\overline{G'}$ is a cycle-free graph). By Condition 5, this component must contain only group vertices. Suppose the component contains vertices for the groups $\mathbb{G}_{s_1}, \ldots, \mathbb{G}_{s_u}$. Compute the largest total degree d of any polynomial in the set $P_{s_1} \cup \ldots \cup P_{s_u}$. Label each vertex in the component with the label d.

   We claim that the total degree of every polynomial computable in each of these groups is bounded by d. This is because the operation set graph in this component only contains group vertices and sum vertices – i.e. vertices associated with operations that do not increase the degree of a polynomial. Hence, the largest total degree of any polynomial computable in any of the groups is bounded by the largest degree of any input polynomial in the component.
4. Consider each component which has edges leading into it only from labelled vertices. At each stage of the process there must exist at least one such component because $\overline{G'}$ is a cycle free graph.
   - If this component contains only the group vertices for groups $\mathbb{G}_{s_1}, \ldots, \mathbb{G}_{s_u}$, then compute the largest total degree d of any polynomial in the set $P_{s_1} \cup \ldots \cup P_{s_u}$ and the largest value d' of all the vertices that have an edge leading into the component. Label every vertex in the component with $\max\{d, d'\}$.
   - If this component contains a power vertex, then it must be the only vertex in the component. If the exponent in the power vertex is $(\,\cdot\,)^\alpha$ and the input is from a vertex labelled with degree d, then label the power vertex with degree αd.
   - If the component contains only a product vertex, and the inputs to the product vertex are vertices labelled with degrees $d_1, \ldots, d_m$, then the product vertex is labelled with degree $d_1 + \ldots + d_m$.
   - If the component contains at least one group vertex and at least one group vertex, then it must be of the form shown in Figure 2. Note that the maximum total degree of any polynomial computable in a group in this component *without* using an operation associated with a product vertex is bounded by the maximum of the largest total degree of any of the input polynomials in the component and the largest label of an edge that leads directly into that group. Let this value be d.

     The first use of a product vertex must take as input a polynomial from the component (with total degree bounded by d) and polynomials from the vertices $i_1, \ldots, i_m$ (which have total degree bounded by some known values $d_{i_1}, \ldots, d_{i_m}$). The output polynomial will have degree bounded by $d + d_{i_1} + \ldots + d_{i_m}$. Therefore, the maximal possible value of the degree of a polynomial after one use of a product vertex is given by choosing the maximum possible value of $d_{i_1} + \ldots + d_{im}$. If the algorithm applies q of these operations to a polynomial originally in the group, then the largest value that can be output by that product vertex is $d+q(d_{i_1}+\ldots+d_{i_m})$.

     Compute the largest possible value of $d_{i_1} + \ldots + d_{i_m}$ associated with any product vertex in the component, and label each vertex in the component with the degree $d + q(d_{i_1} + \ldots + d_{i_m})$.

   Repeat until all the vertices are labelled.
5. In each step, we have labelled each group vertex with the maximal possible total degree of any polynomial computable in that group. Therefore, output the labels $(d_1, \ldots, d_k)$ of the groups $(\mathbb{G}_1, \ldots, \mathbb{G}_k)$.

3.3 Practical Conditions: Discrete Logarithm Problems

If we assume that there are no collisions in our polynomial encoding, then we may simulate the group oracles for an algorithm. The problem-solving algorithm then becomes an SLP-generator. We wish to relate the success probability of an SLP-generator to the form of the polynomial f. In order to do this, we have make sure that a solution cannot be found by chance. In other words, we wish to ensure that the algorithm does not output a polynomial P such that $P(\boldsymbol{X}) \neq f(\boldsymbol{X}, \boldsymbol{c})$ in $\mathcal{L}_n^{(l,c)}$ but $P(\boldsymbol{x}) \equiv f(\boldsymbol{x}, \boldsymbol{c}) \bmod n$ for the particular values of $\boldsymbol{x}$ and $\boldsymbol{c}$ chosen. This can easily be done using the techniques of the previous section provided that the degree of f is bounded.

Condition 6. *There exists a polynomial r' such that for all λ and $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$ we have $\deg f(\boldsymbol{X}, \boldsymbol{c}) < r'$.*

For a discrete logarithm problem, we wish to prevent the situation where the output of the polynomial is constant. This translates to the condition below:

Condition 7. *For all λ and $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$, the polynomial $f(\boldsymbol{X}, \boldsymbol{c})$ is not a constant function in $\mathcal{L}_n^{(l,c)}$.*

Theorem 3. *If a DL-problem satisfies Condition 6 and 7, and p is the largest prime divisor of n, then the problem is ν-SLP-intractable (Condition 2) where*

$$\nu \leq \frac{(\ell - c + 1)r'}{p-1} \tag{11}$$

3.4 Practical Conditions: Diffie-Hellman Problems

To prove that a DH-type problem is intractable, we have to show that no straight line program can trivially solve the problem. Again, this could happen because the SLP outputs the polynomial f or because the the SLP outputs a polynomial $P \neq f$ which frequently "collides" with f. To prevent this latter condition, we require that f has low degree. We thus re-state Condition 6.

Condition 6. *There exists a polynomial r' such that for all λ and $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$ we have $\deg f(\boldsymbol{X}, \boldsymbol{c}) < r'$.*

We also require that no straight line program can compute $f(\boldsymbol{X}, \boldsymbol{c})$.

Condition 8. *For every polynomial q there exists a λ' such that for all $\lambda \geq \lambda'$, $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$, q-SLP-generators $\mathcal{S}$, and $(L_1, \ldots, L_k) \overset{R}{\leftarrow} \mathcal{S}(n, \lambda)$, the output polynomial $P \in L_t$ satisfies $P(\boldsymbol{X}) \neq f(\boldsymbol{X}, \boldsymbol{c})$ in $\mathcal{L}_n^{(l,c)}$.*

Theorem 4. *Suppose an operations set has that the total degree of any polynomial computable in the output group $\mathbb{G}_t$ is bounded by d_t and that the p is largest prime divisor of n. If a DH-problem satisfies Conditions 6 and 8, then for all polynomials q there exists a λ' such that for all $\lambda \geq \lambda'$ the problem is (q, ν)-SLP-intractable where*

$$\nu \leq \frac{(\ell - c + 1)d_t r'}{p-1} \,. \tag{12}$$

Condition 8 is still difficult to check. We can simplify this condition as follows:

Condition 9. *Suppose we can represent the output polynomial f as a finite Laurent polynomial. Then Condition 8 holds if, for all $\boldsymbol{c} \in \mathbb{Z}_n^{c'} \times (\mathbb{Z}_n^*)^{\ell'-c'}$, any of the following conditions hold:*

– *The total degree of any polynomial computable in the group $\mathbb{G}_t$ is bounded by a value which is smaller than the total degree of the polynomial $f(\boldsymbol{X}, \boldsymbol{c})$.*
– *The positive/negative degree of any polynomial computable in the group $\mathbb{G}_t$ is bounded by a value which is smaller than the positive/negative degree of the polynomial $f(\boldsymbol{X}, \boldsymbol{c})$.*
– *The positive/negative degree of any variable X_i of any polynomial computable in the group $\mathbb{G}_t$ is bounded by a value which is smaller than the positive/negative degree of that variable in the polynomial $f(\boldsymbol{X}, \boldsymbol{c})$.*

3.5 Decision Oracles

One weakness of our model is that there is no mechanism for including a decision oracle. This means we cannot model gap problems [OP01]. We can remove this weakness for certain classes of gap oracle by allowing an algorithm access to an oracle for a function of the form

$$op : \mathbb{G}_{s_1} \times \ldots \mathbb{G}_{s_m} \to \{0, 1\} \tag{13}$$

defined by

$$op(g_{s_1}^{y_1}, \ldots, g_{s_m}^{y_m}) = 1 \qquad \text{iff} \qquad OP(y_1, \ldots, y_m) \equiv 0 \bmod n \tag{14}$$

where $OP \in \mathbb{Z}[Y_1, \ldots, Y_m]$ is a polynomial.

This formulation is particularly useful as it allows us to express decision oracles defined rational functions. Consider the Gap Diffie-Hellman problem with respect to a randomised generator. The inputs are group elements (g^r, g^{rx}, g^{ry}) and the algorithm is attempting to find the group element g^{rxy}. The algorithm has access to a DDH oracle, which takes as input $(g^\alpha, g^\beta, g^\gamma)$ and returns 1 if $\alpha\beta r^{-1} \equiv \gamma \bmod n$. This is clearly a rational equation and so not computable in our usual model. However, we can express this as testing whether $\alpha\beta \equiv r\gamma \bmod n$, which falls within the scope of out model. All rational functions can be adapted to a form acceptable to our model.

Furthermore, it is clear to see that, if no collisions occur, then this oracle can be completely simulated by an algorithm using the polynomial encoding of group elements. The abstract condition for collision resistance (Condition 1) remains the same. The practical conditions for collision resistance (Conditions 4 and 5) remain the same providing the degree of OP respects the polynomial bound given in Condition 4. We therefore conclude that a decision oracle is no significant help in solving a computational problem in the generic group model.

4 Conclusions

We have presented conditions by which are sufficient to prove that a discrete-logarithm- or Diffie-Hellman-type problem is hard in the generic group model. These conditions fell into two distinct categories. Conditions on the operation set which guaranteed that the polynomial encoding of group elements was sufficient to represent the group (collision resistance) and conditions which guaranteed that the output function f was not computable. It is worth noting that if the collision resistance of an operations set is proven, then that proof can be re-used in all situations which use the same group structure. Collision resistance has already been proven for many common group structures, including a single group $\mathbb{G}$, a pair of groups $(\mathbb{G}, \mathbb{G}_T)$ that admit a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and a trio of groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3)$ that admit a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ (with or without a computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$).

The model we proposed does have a number of weakness. In particular, the model insists that group operations are of the form $g^{\boldsymbol{x}} \mapsto g^{OP(\boldsymbol{x})}$ where OP is a polynomial. This restriction means that we are unable to consider group operations or oracles which are based on rational functions, such as $g^x \mapsto g^{1/x}$. We were also forced to restrict our attention to group element inputs defined by finite Laurent polynomials. This restriction prevents the use of input elements of the form, for example, $g^{1/(x+c)}$, which has an infinite Laurent expansion.

These two problems are related. We cannot allow non-polynomial group operations as they give rise to group elements encoded by rational functions. We cannot allow group elements encoded by rational functions as it is too difficult to control the total degree of rational functions: simple operations such as addition can cause the total degree of the rational function representing the group element to increase exponentially.

Acknowledgements The authors would like to thank Dan Brown for his comments. The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the authors' views, is provided as is and no guarantee or warranty that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

[BB04] D. Boneh and X. Boyen. Short signatures without random oracles. In *Advances in Cryptology: Proceedings of EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer-Verlag, 2004.

[BBG05] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext (full paper). Cryptology ePrint Archive, Report 2005/015, 2005. `http://eprint.iacr.org/`.

[BR04] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology: Proceedings of EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer-Verlag, 2004.

[Den02] A. W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In *Advances in Cryptology: Proceedings of ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 100–109. Springer-Verlag, 2002.

[Fis00] M. Fischlin. A note on security proofs in the generic model. In *Advances in Cryptology: Proceedings of ASIACRYPT 2002*, volume 1976 of *Lecture Notes in Computer Science*, pages 458–469. Springer-Verlag, 2000.

[Kil01] E. Kiltz. A tool box of cryptographic functions related to the Diffie-Hellman function. In *INDOCRYPT '01: Proceedings of the Second International Conference on Cryptology in India*, volume 2247 of *Lecture Notes in Computer Science*, pages 339–350. Springer-Verlag, 2001.

[KM07] N. Koblitz and A. Menezes. Another look at generic groups. *Advances in Mathematics of Communication*, 1(1):13–28, 2007.

[Mau00] U. Maurer. Index search, discrete logarithms, and Diffie-Hellman. Talk at MRSI Number-theoretic cryptography workshop, Mathematical Sciences Research Institut (MRSI), Berkeley, October 2000.

[Mau05] U. Maurer. Abstract models of computation in cryptography. In Nigel Smart, editor, *Cryptography and Coding 2005*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2005.

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[OP01] T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer-Verlag, 2001.

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing discrete logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.

[Pol78] J. M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology: Proceedings of EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1997.

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

[SPMLS02] J. Stern, D. Pointcheval, J. Malone-Lee, and N. P. Smart. Flaws in applying proof methodologies to signature schemes. In *Advances in Cryptology: Proceedings of CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 93–110. Springer-Verlag, August 2002.

A Referees' Appendix

This appendix contains proofs for certain propositions. It is not intended to form part of the published paper and is only provided to let the referees check that details of the proofs.

Proof of Theorem 1 We prove the theorem through a sequence of games, using standard game hopping techniques [BR04,Sho04]. Let S_i be the event that the algorithm $\mathcal{A}$ solves the problem in *Game i*.

Let *Game 1* be the normal game in which the algorithm $\mathcal{A}$ is given access to group elements. Let *Game 2* be the game in which the algorithm is given access to oracles that act on the polynomial encoding of group elements; i.e. group elements are stored as polynomials in the indeterminate vector $\boldsymbol{X}$. The two games proceed identically up to a point at which the algorithm computes two group elements that are equal as group elements but have different polynomial encodings. These two group elements form a collision in the group. Hence, by Condition 1, we have that

$$|\Pr[S_1] - \Pr[S_2]| \leq \nu_1 \tag{15}$$

Now, in Game 2 the algorithm can simulate the oracles to which it has access. Since the algorithm is completely independent of the group oracles, we may think of it as an SLP-generator in which the choice of SLP output by the generator is defined by the random tape of the algorithm. We note that these SLPs are deterministic, even though the SLP-generator is probabilistic. We now consider DL- and DH-type problems separately.

For a DL-type problem, each SLP always outputs the same solutions α and vector $\boldsymbol{c}$. Therefore, by Condition 2, the probability that this SLP solves the problem is bounded by ν_2 and so we conclude that the probability that the SLP-generator solves the problems is also bounded by ν_2.

For a DH-type problem, by Condition 3, we know that the probability that the any SLP-generator solves the problem is bounded by ν_3. Hence, the result follows trivially. □

Proof of Lemma 1 Suppose $\boldsymbol{x}$ is a solution of $P(\boldsymbol{x}) \equiv 0 \bmod n$, then $\boldsymbol{x}$ is certainly a solution of $P(\boldsymbol{x}) \equiv 0 \bmod p$. Since $P \in \mathcal{L}_n^{(l,c)}$ we have that only the variables $X_{c+1}, \ldots, X_\ell$ can occur with negative exponents and that these exponents must be bounded by the total degree d. Hence, $P' = X_{c+1}^d \ldots X_\ell^d P$ is a polynomial over $\mathbb{Z}_p$ with degree bounded by $(\ell - c + 1)d$. Furthermore, if $\boldsymbol{x}$ is a root of P, then $\boldsymbol{x}$ is a root of P' and so the number of roots of P is bounded by the number of roots of P'. However, the number of roots of P' can be bounded by $(\ell - c + 1)d/(p - 1)$, by the lemma of Shoup[Sho97]. □

Proof of Theorem 3 Express f as f_1/f_2 where f_1, f_2 are polynomials. By Condition 6 and 7, for any $\alpha \in \mathbb{Z}_n$ we have that $f_1(\boldsymbol{X}, \boldsymbol{c}) - f_2(\boldsymbol{X}, \boldsymbol{c})\alpha$ is a non-zero polynomial of total degree bounded by r'. Hence, the probability that $f_1(\boldsymbol{X}, \boldsymbol{c}) - f_2(\boldsymbol{X}, \boldsymbol{c})\alpha \equiv 0 \bmod n$ for randomly chosen $\boldsymbol{x} \leftarrow \mathbb{Z}_n^c \times (\mathbb{Z}_n^*)^{\ell - c}$ is bounded by the given value. □

Proof of Theorem 4 Let P be the polynomial output by the SLP and express f as f_1/f_2 where f_1 and f_2 are polynomials. The SLP solves the problem if $P(\boldsymbol{x}) \equiv f(\boldsymbol{x}, \boldsymbol{c}) \bmod n$. By Condition 8, we have that $P(\boldsymbol{X}) \neq f(\boldsymbol{X}, \boldsymbol{c})$ in $\mathcal{L}_n^{(l,c)}$. So $P(\boldsymbol{X})f_2(\boldsymbol{X}, \boldsymbol{c}) - f_1(\boldsymbol{X}, \boldsymbol{c})$ is a non-zero polynomial of degree bounded by $d_t r'$ and so the probability $P(\boldsymbol{x}) = f(\boldsymbol{x}, \boldsymbol{c}) \bmod n$ is bounded by the given value. □