UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Security Proofs for Pairing-Based Cryptography in the Generic Group Model

Bachelor's Thesis

by
Jan Bobolz
jbobolz@mail.uni-paderborn.de

Thesis Supervisor:
Prof. Dr. rer. nat. Johannes Blömer

Paderborn, September 19, 2013

Declaration

(Translation from German)

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Original Declaration Text in German:

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder singemäß übernommen worden sind, sind als solche gekennzeichnet.

City, Date

Signature

Contents

1 Introduction

The security of current cryptographic constructions is often based on the difficulty of the *Diffie-Hellman problem* or one of its variants. The *computational Diffie-Hellman assumption* states that given a generator g of a finite cyclic group $(G, \cdot)$ and two elements $g^a, g^b \in G$ it is hard to compute the element g^{ab}. While this assumption is not generally true for an arbitrary group, for some groups (e.g., some elliptic curve groups) it is widely believed, but not proven, that it holds.

Because there are many cryptographic schemes that rely on such assumptions, it is important to gain confidence in the belief that the underlying problem is indeed hard. Since a proof in the standard model is unknown, other methods have to be applied.

The *generic group model*, as introduced by Shoup [Sho97], provides one of those methods (similar considerations have been made by Nechaev [Nec94]). Algorithms in the generic group model have oracle access to group operations and group elements are encoded by random bit-strings. Therefore, such *generic algorithms* are effectively prohibited from taking advantage of any special properties of the group or its encoding. In this model, proofs of Diffie-Hellman-related assumptions are feasible. However, those results and their meaning for a concrete group need to be properly interpreted. Also, there are some examples of published proofs in the generic group model that are subtly, but seriously flawed, so careful examination of such proofs is necessary [KM07]. Still, it is an often-used and helpful model to justify assumptions that a scheme relies on.

For security proofs in the generic group model, Boneh, Boyen and Goh [BBG05] provide a useful framework. Their master argument can be used to bound the advantage of generic algorithms in solving decisional problems related to Diffie-Hellman. Their framework is designed for bilinear groups, so it is well-suited for proofs in *pairing-based cryptography*.

However, there are recent assumptions in pairing-based cryptography that the Boneh, Boyen, Goh framework cannot be directly applied to. An example for this is a construction by Waters [Wat11] for a *ciphertext-policy attribute-based* encryption scheme. In conventional public key cryptography, each message is encrypted for a single specific recipient. In contrast, attribute-based encryption schemes allow a ciphertext to be decrypted by a set of people. This set of people is determined by attributes that the sender may specify at encryption time.
Water's construction is proven secure under the new and relatively strong *decisional q-parallel bilinear Diffie-Hellman exponent assumption* (*q-parallel BDHE*). This assumption is not very well studied and again, a proof in the standard model is unknown. Here, a proof in the generic group model is a good way to gain some confidence in their new

assumption. However, the original Boneh, Boyen, Goh framework does not apply to this kind of problem.
To solve this, we provide an extension to the framework that covers a wide range of typical problems in pairing-based cryptography (such as the assumption by Waters) directly.

The thesis is structured as follows:
In Section 3, we discuss the generic group model, the limits of generic algorithms and how proofs in the generic group model need to be interpreted.
Section 4 describes the Boneh, Boyen, Goh framework for security proofs in the generic group model. We provide a detailed proof for their framework.
In Section 5, we provide an extension to the Boneh, Boyen, Goh framework that covers a wider range of problems.
Finally, in Section 5.2, we show how our extension of the framework can be applied to typical assumptions in pairing-based cryptography. Particularly, our extension can be used for the q-parallel BDHE assumption [Wat11] and we prove that this assumption holds generically in Section 5.2.1.

2 Foundations and Notation

This thesis uses the following notation and basic definitions:

2.1 Notation

- $\mathbb{N}$ is the set of natural numbers ($0 \notin \mathbb{N}$) and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.
- For $n \in \mathbb{N}$ we set $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$, that is the ring of integers modulo n. If n is prime, we also write $\mathbb{Z}_n = \mathbb{F}_n$ and in this case, $\mathbb{Z}_n$ is a field.
- If R is a ring, $R^\times$ is the set of units of R.
- For a commutative ring R, $R[X_1, \ldots, X_n]$ is the ring with adjoined elements $X_1, \ldots, X_n$:

  $$R[X_1, \ldots, X_n] = \left\{ \sum_{i=1}^{k} a_i \cdot \prod_{j=1}^{n} X_j^{b_{i,j}} \middle| k \in \mathbb{N}, \forall i \forall j : a_i \in R, b_{i,j} \in \mathbb{N}_0 \right\}$$

  (i.e. a minimal superset of R that is a ring and contains $\{X_1, \ldots, X_n\}$)
  An important example is the (multivariate) polynomial ring over a field. Notice that the adjoined elements may be related. For example, $R[X, X^{-1}] = R[X][X^{-1}]$ is a ring with adjoined X that has a multiplicative inverse for X. However, if not declared otherwise, we assume $X_1, \ldots, X_n$ to be n different variables.
- Let R, R' be commutative rings with ring homomorphism $\phi : R \to R'$ and $(r'_1, \ldots, r'_n) \in (R')^n$.
  For $f \in R[X_1, \ldots, X_n]$ (where the X_i may be related), we define $f(r') = f'(r')$ where f' is the corresponding element in $R'[X_1, \ldots, X_n]$ where the coefficients were projected to R' using ϕ. This means that polynomials in R can be evaluated with elements of R' which yields an element of R'.
- By convention, polynomial variables have upper-case names, ring elements have lower-case names.
- Let R be a ring, $f \in R[X_1, \ldots, X_n]$ a multivariate polynomial.
  f can be written as $f = \sum_{(k_1,\ldots,k_n)\in I} \sigma(k_1, \ldots, k_n) \prod_{i=1}^{n} X_i^{k_i}$ for some finite set $I \subset (\mathbb{N}_0)^n$ and $\sigma : I \to R$.

The *degree* of f is

$$\deg(f) = \max\left\{ \sum_{i=1}^{n} k_i \;\middle|\; (k_1, \ldots, k_n) \in I,\ \sigma(k_1, \ldots, k_n) \neq 0 \right\}$$

- Let $A, B, C \neq \emptyset$ be non-empty sets and $f : A \to B$, $g : B \to C$ two maps.
  - For $a \in A, b \in B$ we write $a \mapsto b$ if $f(a) = b$.
  - $g \circ f : A \to C$ is the composite function of g and f.
  - $\mathrm{im}(f) \subseteq B$ is the image of f.
- For a set S and $n \in \mathbb{N}_0$, S^n is the set of n-tuples over S.
  $S^* = \bigcup_{i=0}^{\infty} S^i$ is the set of (finite) tuples over S.
- For a finite set S we write $s \xleftarrow{R} S$ if s is a random variable that is uniformly distributed over S.
- For a suitable map f and a random variable y, we write $x \leftarrow f(y)$ if x is a random variable that takes on the value of $f(y)$.
- An *instance generator* $\mathcal{G}$ is a probabilistic algorithm with unary input 1^n (a string of length n). Its output depends on the specific problem where it is used. In general, $\mathcal{G}$ is used to randomly generate a group of order at least 2^n. We write $X \leftarrow \mathcal{G}(1^n)$ if X is a random variable for $\mathcal{G}$'s output.
  We assume that for each n, $\mathcal{G}(1^n)$ chooses from a finite set of groups.
- log is the logarithm to base 2.

2.2 Basic Definitions and Problems

In this section, we will introduce some important definitions and problems that are used throughout this thesis.

For security proofs, we often want to show that an event happens with "very small" probability (for example, *an attacker can break the scheme only with negligible probability* with respect to some security parameter). To express this notion of *negligible* quantities formally, we use the following definition.

Definition 1. *A function* $f : \mathbb{N} \to \mathbb{R}$ *is* negligible *if*

$$\forall c \in \mathbb{N}\ \exists n_0 \in \mathbb{N}\ \forall n > n_0 :\ f(n) < 1/n^c$$

Loosely speaking, a negligible function approaches 0 faster than the inverse of any polynomial.

Throughout the thesis, we use the following well-known cryptographic problems.

Problem 2 (Discrete Logarithm)**.** *Let $(G, \cdot)$ be a cyclic group with generator $g \in G$.*
The Discrete Logarithm *problem is:*
Given g and $h \xleftarrow{R} G$, determine $a \in \mathbb{Z}$ such that $g^a = h$.

The discrete logarithm problem is widely believed to be hard in certain groups, for example in $(\mathbb{Z}_p^\times, \cdot)$ for a large prime number $p \in \mathbb{N}$. In other groups however, it is known to be easy.
In $(\mathbb{Z}_n, +)$, $n \in \mathbb{N}$, it is trivial: Given $a \cdot g$ for a generator g (additively written), we can simply multiply the input with g^{-1} to obtain $a \in \mathbb{Z}_n$.

A related problem is the *Diffie-Hellman* problem.

Problem 3 (Diffie-Hellman)**.** *Let $(G, \cdot)$ be a cyclic group with generator $g \in G$.*
The computational Diffie-Hellman *problem is:*
Given g and $g^a, g^b \xleftarrow{R} G$, compute $g^{a \cdot b}$.

The assumption that this problem is hard in certain groups is used in several cryptographic constructions, for example, the Diffie-Hellman key exchange.
It is easy to see that if there is an efficient algorithm for the discrete logarithm problem, then the Diffie-Hellman problem becomes easy: Given g^a, g^b we could efficiently determine b and then raise g^a to the power of b in time that is logarithmic in the group order (using square and multiply) to obtain $g^{a \cdot b}$ as required.

An easier problem is the *decisional* Diffie-Hellman problem. In this variant, the algorithm is asked to distinguish the solution to a computational Diffie-Hellman problem $g^{a \cdot b}$ from a random group element.

Problem 4 (Decisional Diffie-Hellman)**.** *Let $(G, \cdot)$ be a cyclic group with generator $g \in G$.*
The decisional Diffie-Hellman *problem is:*
Given g and $g^a, g^b \xleftarrow{R} G$ and an element $T \in G$, decide whether $T = g^{a \cdot b}$.

Obviously, if an algorithm can efficiently solve computational Diffie-Hellman then the decisional variant is easy as well since one could simply compute the correct group

element and compare it to the one that was supplied.

2.3 Bilinear Groups

In pairing-based cryptography, the following definitions are relevant.

Definition 5 (Bilinear Map)**.** *A* bilinear map *between groups* $(G_0, \cdot), (G_1, \cdot)$ *is a map*

$$e : G_0 \times G_0 \to G_1$$

such that for all $a, b, c \in G_0$

$$e(ab, c) = e(a, c) \cdot e(b, c)$$

and

$$e(a, bc) = e(a, b) \cdot e(a, c)$$

We say e *is* non-degenerate *if* $\mathrm{im}(e) \neq \{1\}$.

Definition 6 (Bilinear Group (cf. Section 2.2 [BBG05]))**.** *Let* $(G_0, \cdot)$ *and* $(G_1, \cdot)$ *be cyclic groups of prime order* p *and let* $g \in G_0$ *be a generator.*
Let $e : G_0 \times G_0 \to G_1$ *be an efficiently computable non-degenerate bilinear map.*

We call $((G_0, \cdot), (G_1, \cdot), e)$ *a* bilinear group *of order* p*. We say that* $g \in G_0$ *is a generator of the bilinear group.*
In this context, e *is often called a* pairing.

Observation 7. For a bilinear group $((G_0, \cdot), (G_1, \cdot), e)$ with generator $g \in G_0$, it holds that for all $a, b \in \mathbb{Z}$

$$e(g^a, g^b) = e(g, g)^{ab}$$

and that

$$e(g, g) \neq 1$$

Particularly, since G_1 has prime order, $e(g, g)$ is a generator of G_1 and $\mathrm{im}(e) = G_1$.

In such bilinear groups (for example, elliptic curve groups with the Weil pairing), the decisional Diffie-Hellman problem (Problem 4) is trivial:
For input $g, g^a, g^b \in G_0$ and $T = g^c \in G_0$, we can base the decision on the values of $e(g^a, g^b) = e(g, g)^{a \cdot b}$ and $e(g, g^c) = e(g, g)^c$.
Since $e(g, g)$ is a generator, $e(g, g)^{a \cdot b}$ and $e(g, g)^c$ are equal if and only if $a \cdot b = c$ modulo

group order. This is equivalent to $g^c = g^{a \cdot b}$, which is what we need to decide.

However, a slightly altered version of the problem is believed to be hard even in (some) bilinear groups.

Problem 8 (Decisional bilinear Diffie-Hellman)**.** *Let* $((G_0, \cdot), (G_1, \cdot), e)$ *be a bilinear group with generator* $g \in G_0$.
The decisional bilinear Diffie-Hellman *problem is:*
Given g *and* $g^a, g^b, g^c \xleftarrow{R} G_0$ *and an element* $T \in G_1$*, decide whether* $T = e(g, g)^{a \cdot b \cdot c}$.

The corresponding computational problem is therefore believed to be hard as well.

Problem 9 (Computational bilinear Diffie-Hellman)**.** *Let* $((G_0, \cdot), (G_1, \cdot), e)$ *be a bilinear group with generator* $g \in G_0$.
The computational bilinear Diffie-Hellman *problem is:*
Given g *and* $g^a, g^b, g^c \xleftarrow{R} G_0$*, compute* $e(g, g)^{a \cdot b \cdot c}$.

2.4 Lemma of Schwartz-Zippel

In this section, we introduce a useful lemma that bounds the probability for multivariate polynomials to vanish when evaluated for random values.

Lemma 10 (Schwartz-Zippel)**.** *Let* F *be a field,* $\emptyset \neq S \subseteq F$ *a finite subset,* $n \in \mathbb{N}_0$, $f \in F[X_1, \ldots, X_n]$, $f \neq 0$.
Then

$$\Pr[f(x_1, \ldots, x_n) = 0] \leq d/|S|$$

where $d = \deg(f)$ *and the probability is over* $x_1, \ldots, x_n \xleftarrow{R} S$.

Proof. We will prove this by induction over the number of variables n.
For $n = 0$, the statement holds trivially, since $f \in F \setminus \{0\}$.
As another base case, let $n = 1$. This means that $f \in F[X_1]$ and consequently $\Pr[f(x) = 0] \leq d/|S|$ because any univariate polynomial f over a field has at most $d = \deg(f)$ roots in F and there are $|S|$ values to choose from for x.

For the inductive step, let $n > 1$ and we assume that the statement holds for polynomials with fewer than n variables.

We write

$$f = \sum_{i=0}^{d} X_1^i f_i$$

where $f_i \in F[X_2, \ldots, X_n]$. Since $f \neq 0$, there is an index k where $f_k \neq 0$ and $f_j = 0$ for all $j > k$.

It holds that $d = \deg(f) \geq \deg(X_1^k \cdot f_k) = k + \deg(f_k)$ and therefore $\deg(f_k) \leq d - k$.

Because f_k is a polynomial in only $n-1$ variables, it follows from the induction hypothesis that $\Pr[f_k(x_2, \ldots, x_n) = 0] \leq \deg(f_k)/|S| \leq (d-k)/|S|$.

Also, for any concrete $x_2, \ldots, x_n \in S$ consider

$$f' = f(X_1, x_2, \ldots, x_n) \in F[X_1]$$

If $0 \neq f_k(x_2, \ldots, x_n) \in F$, then $\deg(f') = k$ as we chose k to be the greatest index in $f = \sum_{i=0}^{d} X_1^i f_i$ and by definition of f'.
Consequently, since f' is a polynomial in one variable, the induction hypothesis implies $\Pr[f'(x_1) = 0 \mid f_k(x_2, \ldots, x_n) \neq 0] \leq \deg(f')/|S| = k/|S|$. By definition of f' it holds that $f'(x_1) = f(x_1, x_2, \ldots, x_n)$ and consequently
$\Pr[f(x_1, \ldots, x_n) = 0 \mid f_k(x_2, \ldots, x_n) \neq 0] \leq k/|S|$

Using the statements above, we have

$$\begin{aligned}
&\Pr[f(\cdot) = 0] \\
&= \Pr[f(\cdot) = 0 \mid f_k(\cdot) = 0] \cdot \Pr[f_k(\cdot) = 0] + \Pr[f(\cdot) = 0 \mid f_k(\cdot) \neq 0] \cdot \Pr[f_k(\cdot) \neq 0] \\
&\leq \Pr[f_k(\cdot) = 0] + \Pr[f(\cdot) = 0 \mid f_k(\cdot) \neq 0] \\
&\leq \frac{(d-k)}{|S|} + \frac{k}{|S|} \\
&= \frac{d}{|S|}
\end{aligned}$$

□

Later in the thesis, we will introduce Lemma 23 which can be applied in a more general situation than the Schwartz-Zippel lemma. Its proof will be similar to the proof of Schwartz-Zippel given here.

3 The Generic Group Model

In this section, we will introduce and discuss the generic group model. First, we will introduce the generic group model in its original sense and discuss its properties. In Section 3.2 we will introduce a version of the model for bilinear groups.

3.1 Original Definition

The central definition for the generic group model as introduced by Shoup [Sho97] is the following:

Definition 11 (Generic Algorithms)**.** *Let $(G, +)$ be a finite group and $S \subset \{0,1\}^*$ a finite set with $|G| \leq |S| < \infty$.*
A generic algorithm $\mathcal{A}$ *for G on S is a probabilistic algorithm such that*

- *$\mathcal{A}$'s input is a tuple $I \in \mathrm{im}(\sigma)^*$ for $\sigma \xleftarrow{R} \{\sigma : G \to S \mid \sigma \text{ injective}\}$*
- *$\mathcal{A}$ has access to a dynamic* encoding list *$L = (\sigma(x_1), \ldots, \sigma(x_l)) \in S^*$. This list is initialized with I.*
- *$\mathcal{A}$ may query an* oracle *for group operations: $\mathcal{A}$ specifies two indices $1 \leq i, j \leq l$ of the current encoding list $L = (\sigma(x_1), \ldots, \sigma(x_l))$ and a sign bit. The oracle then computes $x_{l+1} = x_i \pm x_j$ according to the sign-bit and appends $\sigma(x_{l+1})$ to the encoding list.*
- *$\mathcal{A}$'s output is a bit-string.*

We call S the set of (possible) *encodings* and σ an *encoding function*. The group G encoded by a random σ is sometimes called a *generic group*. If the definition of S is omitted, we assume it to be an arbitrary suitable set.

In summary, in the generic group model, algorithms have access to an oracle for group operations on elements that are encoded by unique but random bit-strings. A generic algorithm can gain information about the random encoding function σ only by querying the oracle for group operations and it has no further input other than the encoded group elements.

One way to imagine the way the oracle works internally is that it has access to an encoding function σ (which is randomly chosen before the start of the algorithm) and

to an internal list $((x_1, \sigma(x_1)), \ldots, (x_l, \sigma(x_l)))$ of pairs of group elements with their respective encodings, corresponding to the algorithm's current encoding list. Whenever a query is made for indices i, j on the encoding list, the oracle finds x_i and x_j in its list and calculates $x_{l+1} = x_i \pm x_j$ in G according to the sign bit and evaluates $\sigma(x_{l+1})$, then it appends $(x_{l+1}, \sigma(x_{l+1}))$ to its internal list and $\sigma(x_{l+1})$ to the generic algorithm's encoding list.
Alternatively, the oracle may not create a complete random encoding function σ at the beginning, but rather make random choices whenever it is queried. In this case, it would compute $x_{l+1} = x_i \pm x_j$ as usual, and then check in its list whether there is an index k such that $x_k = x_{l+1}$ (meaning this particular group element x_{l+1} has already been encoded). In this case, the old encoding $\sigma(x_k)$ is used, otherwise a new random encoding for x_{l+1} is chosen from $S \setminus \{\sigma(x_1), \ldots, \sigma(x_l)\}$. Since an oracle cannot be accessed other than by querying it, these two concepts are equivalent. It should be noted that queries to the oracle are typically assumed to take only constant time for $\mathcal{A}$.

The original definition used in [Sho97] only allows the group $(\mathbb{Z}_n, +)$ to be abstracted by the model. This is because it was originally used only to show limitations of generic algorithms for discrete logarithms (Problem 2) and the Diffie-Hellman problem (Problem 3). Since both of these problems require cyclic groups and any cyclic group of order n is isomorphic to $(\mathbb{Z}_n, +)$, this definition is adequate for this purpose.
However, the original definition can be easily generalized for arbitrary finite groups (as seen in Definition 11).

Each generic algorithm is specifically designed for a concrete group G and for a set of possible encodings S and may treat G and S as known constants. However, since group elements are encoded as random bit-strings, generic algorithms are effectively prevented from using the structure of group elements' encodings. As we will see, this also has implications for the ability of generic algorithms to use certain properties of the group G.

3.1.1 Limits of Generic Algorithms

Since a generic algorithm only sees random encodings of group elements, it cannot distinguish between two isomorphic finite groups. In this sense, the concrete group is interchangeable. We present the following lemma which will express this formally.

Lemma 12. *Let $(G, +)$ and $(G', +)$ be two isomorphic finite groups with group isomorphism $\varphi : G \to G'$ and let $\sigma : G \to S$ be a random injective encoding function for $S \subset \{0,1\}^*$, $|G| = |G'| \leq |S| < \infty$. We set $\sigma' = \sigma \circ \varphi^{-1}$. Let $x_1, \ldots, x_k \in G$.*
Then for any generic algorithm $\mathcal{A}$ the following two scenarios are indistinguishable:

1. *$\mathcal{A}$'s input is $(\sigma(x_1), \ldots, \sigma(x_k))$ and the oracle operates on G.*

2. $\mathcal{A}$'s input is $(\sigma'(\varphi(x_1)), \ldots, \sigma'(\varphi(x_k)))$ and the oracle operates on G'.

Proof. First, we note that if σ is randomly chosen (with respect to a uniform distribution), then σ' is also random and occurs with the same probability, since the mapping with $\sigma \mapsto \sigma' = \sigma \circ \varphi^{-1}$ is a bijection between the respective sets of encoding functions.

The input for the algorithm in the first case is by definition $(\sigma(x_1), \ldots, \sigma(x_k))$, in the second case it is $(\sigma'(\varphi(x_1)), \ldots, \sigma'(\varphi(x_k)))$. Because $\sigma = \sigma' \circ \varphi$, the input is the same in both cases.

Consequently, it suffices to show that at any step of the algorithm, the encoding list is the same in scenario 1 as in scenario 2 (because that implies that the oracle behaves exactly the same in each case).

Initially, the encoding list is given by the input to $\mathcal{A}$. We have already seen that the input is the same.
We consider a query to the oracle for an arbitrary current encoding list (with valid entries, i.e. out of $\mathrm{im}(\sigma)$ which is equal to $\mathrm{im}(\sigma')$) and show that the result of the query will be the same for both cases.
Let L be the encoding list before a query to the oracle. Since the list holds valid entries, we can write $L = (\sigma(y_1), \ldots, \sigma(y_l)) = (\sigma'(z_1), \ldots, \sigma'(z_l))$ for some $y_1, \ldots, y_l \in G$ and $z_1, \ldots, z_l \in G'$.
Given a sign bit and two indices $1 \leq i, j \leq l$:

1. The oracle for G appends $\sigma(y_i \pm y_j)$ to the encoding list
2. The oracle for G' appends $\sigma'(z_i \pm z_j)$ to the encoding list

By definition of σ' and y_i, z_i, it holds that $\sigma'(z_i) = \sigma(\varphi^{-1}(z_i)) = \sigma(y_i)$ and therefore $y_i = \varphi^{-1}(z_i)$ (because σ is injective). Analogously, $y_j = \varphi^{-1}(z_j)$.
Therefore, holds that $\sigma(y_i \pm y_j) = \sigma(\varphi^{-1}(z_i) \pm \varphi^{-1}(z_j)) = \sigma(\varphi^{-1}(z_i \pm z_j)) = \sigma'(z_i \pm z_j)$, i.e. the same encoding is appended in both cases.

Since the input and the oracle's behavior is the same and σ and σ' occur with the same probability, the two scenarios are indistinguishable for $\mathcal{A}$. □

This allows an important insight into generic algorithms: Since a generic algorithm has no way to determine which concrete group it is dealing with, it can only exploit group properties that *all* isomorphic groups have in common (e.g., group order, order of individual elements, identity element, properties like commutativity if applicable). Another way to view this is: A generic algorithm works equally well for all isomorphic groups.
In practice however, some problems may be hard in one group but easy in one that is isomorphic to the first. One famous example for this is the discrete logarithm in $(\mathbb{Z}_p^\times, \cdot)$

for a prime number p. $(\mathbb{Z}_p^\times, \cdot)$ is isomorphic to $(\mathbb{Z}_{p-1}, +)$, but as discussed for Problem 2, in $(\mathbb{Z}_p^\times, \cdot)$ the discrete logarithm is widely believed to be hard, whereas in $(\mathbb{Z}_{p-1}, +)$ it is trivial.

3.1.2 Example for a Generic Algorithm

A good example for a generic algorithm is the *baby-step giant-step* algorithm that solves the discrete logarithm problem for a group of order n in $O(\sqrt{n})$ time.

For a cyclic group G of order n, the input to the algorithm is a generator $g \in G$ and some element $g^a \in G$. We set $k = \lceil \sqrt{n} \rceil$.

1. Compute a table of tuples $(j, g^j) \in \mathbb{N} \times G$ for $1 \leq j < k$.
2. For $1 \leq i \leq k$, compute $\text{tmp} = g^a \cdot (g^{-k})^i \in G$ and check whether the table from step 1 contains a tuple (j, tmp).
   If it does, output $i \cdot k + j$.

The algorithm is correct. When it outputs $i \cdot k + j$, it holds that $g^{i \cdot k + j} = g^{i \cdot k} \cdot g^j = g^{i \cdot k} \cdot g^{a - k \cdot i} = g^a$.
The algorithm always outputs a solution. This is because $0 \leq a < n$ can be written as $a = i \cdot k + j$ where $1 \leq j < k$ and $i \leq k$ since $k^2 = \lceil \sqrt{n} \rceil^2 \geq n$. All possible values for j and i are checked in step 2.
The required time for step 1 is $O(\sqrt{n})$ (assuming constant time group operations). For step 2 it is also $O(\sqrt{n})$. This can be achieved by multiplying the constant value of g^{-k} with the previously computed value of $g^a \cdot (g^{-k})^{i-1}$ and by using a hash table for constant time lookup for group elements in the table.

As one can see, the algorithm only uses common properties of cyclic groups $(G, \cdot)$ of order n (or, in the terms of Section 3.1.1, groups that are isomorphic to $(\mathbb{Z}_n, +)$). Namely, the group order is used to compute the necessary number of tuples in the table, and the fact that the group is cyclic is needed for correctness. Also, the only actions that depend on the encodings of group elements are group operations (multiplication, inverse) which can be easily modeled as queries to an oracle. Furthermore, for baby-step giant-step, the fact that group elements have a *unique* encoding can be used for fast lookup (e.g., through hashing) to determine whether or not an element is present in the lookup table. We note that the algorithm works for arbitrary cyclic groups (of some known order n), just like any generic algorithm for $\mathbb{Z}_n$.

3.1.3 Usefulness of the Generic Group Model

It should be stressed that the generic group model is not meant to reflect or even approximate concrete groups that are used in practice. Encodings for concrete groups'

elements are certainly not random but rather explicitly chosen to reflect the represented element's structure. This is necessary since in practice, group operations have to be efficiently computable and this is greatly simplified by choosing a suitable structure for the encodings of elements.

The generic group model is often compared to the random oracle model for hash functions. There are certain similarities: They are both tools that can be used to abstract from concrete mathematical objects to allow proofs of theorems that nobody succeeded to prove in the standard model.
The random oracle models the *intended* (although idealized) behavior of a good hash function. In contrast, generic groups are an idealization of groups only with respect to a lack of structure and special properties. These are attributes not usually desirable in concrete groups that are used for cryptographic constructions. This is a distinct difference in the intent of the respective models. Essentially, a random oracle models an ideal hash function, but a generic group does not model an ideal group in a similar fashion.

Instead, the usefulness of the generic group model lies in the following observation: If a cryptographic construction is proven secure in the generic group model, no *generic* algorithm can efficiently attack that construction with high probability of success. In other words, every efficient attacker with high probability of success needs to exploit some property of the underlying group or its elements' encodings.

Examples for this are the discrete logarithm and Diffie-Hellman. Both of these problems have been intensively studied and they are strongly believed to be hard in certain groups but nobody has been successful in proving this. Using the generic group model, one can at least show that for generic algorithms they are hard. Consequently, to attack these problems one has to employ methods related to the concrete group. (And if such an attack is found, the problem may still be hard in another group.)

If it is suspected that a certain group does not have any properties that would help in solving a particular problem (like the discrete logarithm in elliptic curve groups), the generic group model may be used to abstract from most of the group's properties and thereby make a proof feasible. However, this does not adequately replace a formal proof and should be seen as a chance to at least show security against generic algorithms rather than not being able to give any formal proof at all.

Another application of the generic group model is validating new and untested assumptions. For example, Waters constructs a ciphertext-policy attribute-based encryption scheme in [Wat11]. Their most efficient construction relies on a new, relatively strong assumption that they call the decisional q-parallel bilinear Diffie-Hellman exponent assumption. Because it is similar to many other Diffie-Hellman related problems, this new assumption would currently be difficult to show in the standard model. Here, the generic group model serves as a first check that this new assumption is indeed rea-

sonable. In Section 5.2.1, we will show that their assumption indeed holds generically.

In summary, it is important to understand that a proof of security in the generic group model does not imply security for any concrete group. Nevertheless, in the absence of a proof in the standard model, the generic group model is a useful tool to gain confidence in one's assumptions.

3.2 Generic Bilinear Groups

In this thesis, we are particularly concerned with bilinear groups.

Because the original definition of the generic group model is inadequate for bilinear groups (a generic algorithm in the original sense cannot evaluate the bilinear map), we now introduce generic bilinear groups. This definition is a natural extension of Shoup's generic group model (Definition 11).

Definition 13 (Generic Algorithms for Bilinear Groups)**.** *Let $((G_0, \cdot), (G_1, \cdot), e)$ be a bilinear group of order p with generator $g \in G_0$. Let $S \subset \{0,1\}^*$ be a finite set, $p \leq |S| < \infty$.*
A generic algorithm $\mathcal{A}$ *for $((G_0, \cdot), (G_1, \cdot), e)$ and S is a probabilistic algorithm such that*

- *$\mathcal{A}$'s input consists of $I_0 \in \mathrm{im}(\sigma_0)^*$ and $I_1 \in \mathrm{im}(\sigma_1)^*$ for $\sigma_0 \xleftarrow{R} \{\sigma : G_0 \to S \mid \sigma \text{ injective}\}$ and $\sigma_1 \xleftarrow{R} \{\sigma : G_1 \to S \mid \sigma \text{ injective}\}$.*
- *$\mathcal{A}$ has access to dynamic encodings lists: $L_0 = (\sigma_0(x_1), \ldots, \sigma_0(x_k)) \in S^*$ and $L_1 = (\sigma_1(y_1), \ldots, \sigma_1(y_l)) \in S^*$ (for G_0, G_1 respectively). Initially, $L_0 = I_0$ and $L_1 = I_1$.*
- *$\mathcal{A}$ may query an oracle for the following operations:*
  - *Group operation in G_0: $\mathcal{A}$ specifies two indices $1 \leq i, j \leq k$ of the current encoding list $L_0 = (\sigma_0(x_1), \ldots, \sigma_0(x_k))$ and a sign bit. The oracle then computes $x_{k+1} = x_i \cdot x_j$ or $x_{k+1} = x_i \cdot x_j^{-1}$ according to the sign bit and appends $\sigma_0(x_{k+1})$ to the encoding list L_0.*
  - *Group operation in G_1: $\mathcal{A}$ specifies two indices $1 \leq i, j \leq l$ of the current encoding list $L_1 = (\sigma_1(y_1), \ldots, \sigma_1(y_l))$ and a sign bit. The oracle then computes $y_{l+1} = y_i \cdot y_j$ or $y_{l+1} = y_i \cdot y_j^{-1}$ according to the sign bit and appends $\sigma_1(y_{l+1})$ to the encoding list L_1.*
  - *Bilinear map: $\mathcal{A}$ specifies two indices $1 \leq i, j \leq k$ from the current encoding list $L_0 = (\sigma_0(x_1), \ldots, \sigma_0(x_k))$. The oracle then computes $y_{l+1} = e(x_i, x_j)$ and appends $\sigma_1(y_{l+1})$ to the encoding list L_1.*
- *$\mathcal{A}$'s output is a bit-string.*

A similar, but not completely equivalent model is used in [BBG05]. For an examination of the differences consider Section 3.3.

To give an idea of the limits of generic algorithms for bilinear groups, we note that a generic algorithm for a bilinear group $((G_0, \cdot), (G_1, \cdot), e)$ can only exploit properties of this concrete bilinear group that *all* bilinear groups of the same order have in common. This follows from the next lemma similar to Lemma 12:

Lemma 14. *Let $((G_0, \cdot), (G_1, \cdot), e)$ be a bilinear group of prime order p with generator $g \in G_0$ and let $((G'_0, \cdot), (G'_1, \cdot), e')$ also be a bilinear group of prime order p with generator $g' \in G'_0$. Let $\sigma_0 : G_0 \to S$ and $\sigma_1 : G_1 \to S$ be random injective encoding functions for a set $S \subset \{0,1\}^*$ with $p \leq |S| < \infty$.*
We define isomorphisms $\varphi_0 : G_0 \to G'_0$ and $\varphi_1 : G_1 \to G'_1$ through $\varphi_0(g) = g'$ and $\varphi_1(e(g,g)) = e'(g',g')$. We set $\sigma'_i = \sigma_i \circ \varphi_i^{-1} : G'_i \to S$ for $i \in \{0,1\}$.
Let $x_1, \ldots, x_n \in G_0$, $y_1, \ldots, y_m \in G_1$.

Then for any generic algorithm $\mathcal{A}$ the following two scenarios are indistinguishable:

1. *$\mathcal{A}$'s input is $I_0 = (\sigma_0(x_1), \cdots, \sigma_0(x_n))$, $I_1 = (\sigma_1(y_1), \cdots, \sigma_1(y_m))$ and the oracle operates on $((G_0, \cdot), (G_1, \cdot), e)$*

2. *$\mathcal{A}$'s input is $I'_0 = (\sigma'_0(\varphi_0(x_1)), \cdots, \sigma'_0(\varphi_0(x_n)))$, $I'_1 = (\sigma'_1(\varphi_1(y_1)), \cdots, \sigma'_1(\varphi_1(y_m)))$ and the oracle operates on $((G'_0, \cdot), (G'_1, \cdot), e')$*

Proof. First, we note that G_i, G'_i ($i \in \{0,1\}$) are groups of prime order p and g, g', $e(g,g), e'(g',g')$ are generators, therefore φ_0, φ_1 are well-defined isomorphisms. This also implies that σ'_0, σ'_1 are (uniformly) random encoding functions.

Lemma 12 and its proof imply that for group operation queries, the encodings that the oracle returns are the same in both cases.
It only remains to be shown that evaluation of the bilinear map by the oracle also returns the same encoding as a result in both cases.

Let $L_0 = (\sigma_0(y_1), \ldots, \sigma_0(y_l)) = (\sigma'_0(z_1), \ldots, \sigma'_0(z_l))$ be the current encoding list for G_0. Let $1 \leq i, j \leq l$ be two indices.

1. In the first case, the oracle appends $\sigma_1(e(y_i, y_j))$ to the encoding list L_1.

2. In the second case, the oracle appends $\sigma'_1(e'(z_i, z_j))$ to the encoding list L_1.

Like in the proof of Lemma 12, it holds that $y_i = \varphi_0^{-1}(z_i), y_j = \varphi_0^{-1}(z_j)$ by definition

and injectivity of σ_0'. Consequently:

$$\begin{aligned}
\sigma_1'(e'(z_i, z_j)) &= \sigma_1'(e'((g')^a, (g')^b)) \\
&= \sigma_1'(e'(g', g')^{ab}) \\
&= \sigma_1'(\varphi_1(e(g, g)^{ab})) \\
&= \sigma_1(e(g, g)^{ab}) \\
&= \sigma_1(e(\varphi_0^{-1}((g')^a), \varphi_0^{-1}((g')^b))) \\
&= \sigma_1(e(\varphi_0^{-1}(z_i), \varphi_0^{-1}(z_j))) \\
&= \sigma_1(e(y_i, y_j))
\end{aligned}$$

for some $a, b \in \mathbb{Z}$.

Since the input and the encoding lists are the same at any point in the algorithm and the encoding functions occur with the same probability, a generic algorithm $\mathcal{A}$ cannot distinguish between the two scenarios. □

3.3 Generic Group Model as Understood by Boneh, Boyen and Goh

Boneh, Boyen and Goh implicitly use a slight variation of the generic group model for bilinear groups in their proof framework (Theorem A.2 [BBG05]). There is no formal definition of the model in their paper. Their understanding is implied by their use of the model throughout the proof framework. We will discuss the differences to Definition 13 here briefly.

In the variation of Boneh, Boyen and Goh [BBG05], a generic algorithm does not supply two indices from an encoding list to the oracle for a computation, but rather the encodings themselves. Hence, an algorithm may supply arbitrary elements of S to the oracle (where S is the target set of the encoding function). This includes invalid encodings (i.e. elements of $S \setminus \mathrm{im}(\sigma)$ for the encoding function $\sigma : G \to S$), as well as valid encodings (i.e. in $\mathrm{im}(\sigma)$) that the algorithm does not have in its encoding list.
In this case, an algorithm can express queries that the oracle cannot answer adequately because there is an invalid encoding involved. Of course, this could be implemented by defining a special error value.
Successfully generating a random valid encoding is analogous to generating a random group element. In the original definition, this can also be done (for cyclic groups $(G, \cdot)$) by choosing a random exponent $r \in \{1, \ldots, |G|\}$ and computing g^r, which is possible in $O(\log |G|)$ oracle queries using square and multiply. So in this sense, no significant ability is added to an attacker by this modification of the model.
Also, it can be argued that by increasing the size of S, it can be made arbitrarily hard for

an attacker to guess valid encodings. Indeed, the authors of [BBG05] use the same argument in their proof to omit the case that an algorithm might try to make a query with an encoding not on its encoding list. Nevertheless, this change in the model complicates proofs that rely on simulating an oracle. To be completely accurate when simulating a generic group oracle, it would have to be taken into account that an algorithm may request operations on randomly guessed bit-strings. This might be nontrivial to implement if the encoding functions are not completely known during the course of the algorithm, but rather dynamically generated whenever the attacker queries the simulated oracle. When the attacker specifies a bit-string from S that is not on the encoding list, the simulation would have to decide randomly whether this bit-string is supposed to be a valid encoding or not and which group element it should represent if applicable.

Another difference is that in Shoup's original definition, a generic algorithm is specifically designed for a certain group, whereas in [BBG05], the group order p is passed as an argument to the algorithm. In practice, this should not be a meaningful restriction, since an algorithm for a concrete p could most likely be easily generalized for arbitrary p. Also, their proof does not rely on the fact that the algorithm is not specifically designed for a certain p.

Lastly, as a minor difference, the target set S of the encoding functions σ_0, σ_1 in [BBG05] is $\{0,1\}^m$ for some sufficiently large $m \in \mathbb{N}$, not an arbitrary finite subset of $\{0,1\}^*$ as in Shoup's original definition. Since there are no restrictions on S other than its cardinality, one may indeed assume without loss of generality that $S = \{0,1\}^m$ (i.e. all encodings have the same length).

In conclusion, Boneh, Boyen and Goh use a slight variation of the generic group model that was introduced in Definition 13. For the most part, their changes do not significantly alter the general notion of the model. Their framework and its proof also work for the original model with very minor modifications.

4 The Boneh, Boyen, Goh Framework

Boneh, Boyen and Goh [BBG05] developed a framework for proofs of Diffie-Hellman-related problems in the generic group model. We will review and explain their framework here.

4.1 The (P,Q,f)-Diffie-Hellman Problem

For a universal proof method of Diffie-Hellman-related problems in bilinear groups, we define the (P, Q, f)-Diffie-Hellman problem (originally defined in [BBG05]), which covers many of the Diffie-Hellman-related problems directly.

Problem 15. *Let $((G_0, \cdot), (G_1, \cdot), e)$ be a bilinear group of prime order p with generator $g \in G_0$. Let $s, n \in \mathbb{N}$ and $P, Q \in \mathbb{F}_p[X_1, \ldots, X_n]^s$, i.e. each a sequence of s multivariate polynomials over $\mathbb{F}_p$. We write $P = (p_1, \ldots, p_s)$ and $Q = (q_1, \ldots, q_s)$ and require $p_1 = q_1 = 1$. Let $f \in \mathbb{F}_p[X_1, \ldots, X_n]$ be a single polynomial.*
The corresponding (computational) (P, Q, f)-Diffie-Hellman problem *is defined as follows:*

Given

$$\left(g^{p_1(x_1,\ldots,x_n)}, \ldots, g^{p_s(x_1,\ldots,x_n)}, e(g,g)^{q_1(x_1,\ldots,x_n)}, \ldots, e(g,g)^{q_s(x_1,\ldots,x_n)}\right)$$

for $x_1, \ldots, x_n \xleftarrow{R} \mathbb{F}_p$, compute

$$e(g,g)^{f(x_1,\ldots,x_n)}$$

Notice that we have defined a set of problems. Algorithms for one specific (P, Q, f)-Diffie-Hellman problem may treat P, Q, f and $((G_0, \cdot), (G_1, \cdot), e), p, g$ as known constants.

For an easier understanding, one can observe that P defines what elements of G_0 are supplied to the algorithm and Q analogously for G_1. f defines what the algorithm is supposed to compute.

Example 16 (Computational bilinear Diffie-Hellman)**.** *Suppose we set $P = (1, X, Y, Z)$, $Q = (1, 1, 1, 1) \in \mathbb{F}_p[X, Y, Z]^4$, $f = XYZ \in \mathbb{F}_p[X, Y, Z]$, then the corresponding (P, Q, f)-Diffie-Hellman problem is:*
Given g, g^x, g^y, g^z (and $e(g, g)$) for a generator $g \in G_0$ and some $x, y, z \in \mathbb{F}_p$, com-

pute $e(g,g)^{xyz}$, which is essentially the computational bilinear Diffie-Hellman problem (Problem 9)

For more general results, we consider a decisional variant of the problem.

Problem 17 (cf. Section A.2 [BBG05])**.** *Let $((G_0,\cdot),(G_1,\cdot),e)$ be a bilinear group of prime order p with generator $g \in G_0$. Let $s,n \in \mathbb{N}$, $P=(p_1,\ldots,p_s), Q=(q_1,\ldots,q_s) \in \mathbb{F}_p[X_1,\ldots,X_n]^s$, $p_1 = q_1 = 1$ and $f \in \mathbb{F}_p[X_1,\ldots,X_n]$.*
The corresponding decisional (P,Q,f)-Diffie-Hellman problem *is defined as follows:*

Given

$$\left(g^{p_1(x_1,\ldots,x_n)},\ldots,g^{p_s(x_1,\ldots,x_n)},\, e(g,g)^{q_1(x_1,\ldots,x_n)},\ldots,e(g,g)^{q_s(x_1,\ldots,x_n)}\right)$$

for $x_1,\ldots,x_n \xleftarrow{R} \mathbb{F}_p$,
and an element $T \in G_1$, decide whether or not

$$T = e(g,g)^{f(x_1,\ldots,x_n)}$$

Definition 18 (cf. Section A.2 [BBG05])**.** *In the situation of Problem 17, let $I_g(x_1,\ldots,x_n) = \left(g^{p_1(x_1,\ldots,x_n)},\ldots,g^{p_s(x_1,\ldots,x_n)},\, e(g,g)^{q_1(x_1,\ldots,x_n)},\ldots,e(g,g)^{q_s(x_1,\ldots,x_n)}\right)$.*
We say that a probabilistic algorithm $\mathcal{A}$ has advantage ϵ *in solving the decisional (P,Q,f)-Diffie-Hellman problem, if*

$$\left|\Pr[\mathcal{A}(I_g(x_1,\ldots,x_n),e(g,g)^{f(x_1,\ldots,x_n)})=0] - \Pr[\mathcal{A}(I_g(x_1,\ldots,x_n),T)=0]\right| > \epsilon$$

where the probability is over $x_1,\ldots,x_n \xleftarrow{R} \mathbb{F}_p$, $T \xleftarrow{R} G_1$ and $\mathcal{A}$'s random bits.

Notice that any algorithm that can compute a solution to the (P,Q,f)-Diffie-Hellman problem allows to easily solve the decisional variant. Therefore, any results on lower runtime bounds for the decisional variant translate to the computational variant.

4.2 Independence of Polynomials

For certain P,Q,f, the (P,Q,f)-Diffie-Hellman problem is trivial to solve. For example, consider $P=(1,X,Y), Q=(1,1,1), f=XY$. To solve this, an algorithm can simply use the bilinear map on g^x and g^y, since $e(g^x,g^y)=e(g,g)^{xy}=e(g,g)^{f(x,y)}$. The following definition deals with such cases.

Definition 19 (cf. Definition A.1 [BBG05])**.** *Let* $f \in \mathbb{F}_p[X_1, \ldots, X_n]$, $P = (p_1, \ldots, p_s)$, $Q = (q_1, \ldots, q_s) \in \mathbb{F}_p[X_1, \ldots, X_n]^s$.

We say f *is* dependent *on* (P, Q)*, if there exist constants* $a_{ij}, b_k \in \mathbb{F}_p$ *(for* $i, j, k \in \{1, \ldots, s\}$*) such that*

$$f = \sum_{i=1}^{s} \sum_{j=1}^{s} a_{ij} p_i p_j + \sum_{k=1}^{s} b_k q_k$$

f *is* independent *of* (P, Q) *if it is not dependent on* (P, Q)

Notice that if f is dependent on (P, Q), then the corresponding (P, Q, f)-Diffie-Hellman problem is easy to solve generically, regardless of the actual $x_1, \ldots, x_n \in \mathbb{F}_p$. For notational convenience in this paragraph, let $h' = h(x_1, \ldots, x_n)$ for any polynomial $h \in \mathbb{F}_p[X_1, \ldots, X_n]$ and $x_1, \ldots, x_n \in \mathbb{F}_p$.
Consider a (generic) algorithm that does the following:

- For all pairs $(i, j) \in \{1, \ldots, s\}^2$ compute $e(g^{p'_i}, g^{p'_j}) = e(g, g)^{p'_i p'_j}$ using the given $g^{p'_i}, g^{p'_j}$.
- Raise each element $e(g, g)^{p'_i p'_j} \in G_1$ from the previous step to the power of the corresponding constant a_{ij} (obtaining $e(g, g)^{a_{ij} p'_i p'_j} \in G_1$).
- For all $1 \leq k \leq s$, compute $(e(g, g)^{q'_k})^{b_k} \in G_1$ from the given $e(g, g)^{q'_k} \in G_1$.
- Multiply the previous results (in G_1) to obtain $e(g, g)^{\sum_{i=1}^{s} \sum_{j=1}^{s} a_{ij} p'_i p'_j + \sum_{k=1}^{s} b_k q'_k} = e(g, g)^{f'}$

Using square and multiply, the exponentiation operations can each be done using at most $O(\log p)$ group operations (where p is the order of the bilinear group). Consequently, the algorithm solves the (P, Q, f)-Diffie-Hellman problem in only $O((s^2 + s) \log p)$ time (assuming constant time group operations).

As we will see in the following proof, f's independence of (P, Q) is not only necessary, but also sufficient for generic security of the corresponding (P, Q, f)-Diffie-Hellman problem.

4.3 Generic Security of the (P,Q,f)-Diffie-Hellman Problem

For the generic security of the (P, Q, f)-Diffie-Hellman problem we show the following theorem that bounds the probability for success of a generic algorithm for a special decision variant of the (P, Q, f)-Diffie-Hellman problem: The algorithm is given *two* group elements, one of which is the correct solution (this variant differs from Problem 17. Corollary 21 shows that this theorem indeed implies generic security for the decisional (P, Q, f)-Diffie-Hellman problem). We are using Definition 13 for generic bilinear groups instead of the understanding of the model of Boneh, Boyen and Goh. The differences

are listed in Section 3.3. Essentially, the proof is still the same as the one in [BBG05], but slightly adapted to our definition of generic groups and considerably more detailed.

Theorem 20 (cf. Theorem A.2 [BBG05])**.** *Let $p \in \mathbb{N}$ be a prime number, $((G_0, \cdot), (G_1, \cdot), e)$ a bilinear group of order p with generator $g \in G_0$. Let $S \subset \{0,1\}^*$ with $p \leq |S| < \infty$. Further, let $s, n \in \mathbb{N}$ and $P = (p_1, \ldots, p_s), Q = (q_1, \ldots, q_s) \in \mathbb{F}_p[X_1, \ldots, X_n]^s$ with $p_1 = q_1 = 1$, $f \in \mathbb{F}_p[X_1, \ldots, X_n]$. Let $d = \max\{2\deg(p_i), \deg(q_i), \deg(f) \mid i \in \{1, \ldots, s\}\}$.*
We set

$$I(\sigma_0, \sigma_1; x_1, \ldots, x_n, t_0, t_1) = \left(\begin{array}{c} \sigma_0(g^{p_1(x_1,\ldots,x_n)}), \ldots, \sigma_0(g^{p_s(x_1,\ldots,x_n)}), \\ \sigma_1(e(g,g)^{q_1(x_1,\ldots,x_n)}), \ldots, \sigma_1(e(g,g)^{q_s(x_1,\ldots,x_n)}), \\ \sigma_1(e(g,g)^{t_0}), \sigma_1(e(g,g)^{t_1}) \end{array} \right)$$

and call it the input vector

If f is independent of (P, Q), then for any generic algorithm $\mathcal{A}$ for $((G_0, \cdot), (G_1, \cdot), e)$ and S that makes at most q queries to the oracle it holds that

$$\left| \Pr\left[\mathcal{A}\left(I(\sigma_0, \sigma_1; x_1, \ldots, x_n, t_0, t_1)\right) = b : \begin{array}{c} x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, \\ b \xleftarrow{R} \{0,1\}, \\ t_b \leftarrow f(x_1, \ldots, x_n), \\ t_{1-b} \leftarrow y \end{array} \right] - \frac{1}{2} \right| \leq \frac{(q + 2s + 2)^2 \cdot d}{4p}$$

where the probability is over $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, b \xleftarrow{R} \{0,1\}, \sigma_0 \xleftarrow{R} \{\sigma : G_0 \to S \mid \sigma \text{ injective}\}, \sigma_1 \xleftarrow{R} \{\sigma : G_1 \to S \mid \sigma \text{ injective}\}$ and the random bits of $\mathcal{A}$.

For this theorem, $\mathcal{A}$ is given the usual input and two (encoded) group elements $e(g,g)^{t_0}, e(g,g)^{t_1} \in G_1$, one of which is the element $e(g,g)^{f(x_1,\ldots,x_n)}$ and the other is randomly chosen. $\mathcal{A}$ must then distinguish the correct answer $e(g,g)^{f(x_1,\ldots,x_n)}$ from the random group element. Notice that if $\mathcal{A}$ could actually solve the computational version of the (P, Q, f)-Diffie-Hellman problem, it could simply compute $e(g,g)^{f(x_1,\ldots,x_n)}$ and compare it to $e(g,g)^{t_0}$ and $e(g,g)^{t_1}$ for the decision.
The proposition is that the probability for an arbitrary generic algorithm $\mathcal{A}$ to output the correct choice is close to $\frac{1}{2}$ and therefore close to the probability one would achieve by simply choosing $b \xleftarrow{R} \{0,1\}$ at random without computing anything.
As a slight change from [BBG05], we show a bound that is more tight by a factor of $1/2$.

Proof. The idea of the proof is to consider an Algorithm $\mathcal{B}$ that simulates a generic group oracle and answers $\mathcal{A}$'s oracle queries. Analysis of the output of $\mathcal{A}$ when interacting with $\mathcal{B}$ will provide the information needed to prove this theorem.
Unlike a proper oracle, $\mathcal{B}$ will not depend on the random values of $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, b \xleftarrow{R} \{0,1\}$ and do its computations on group elements. Instead, $\mathcal{B}$ will try to pro-

vide an accurate simulation by playing along with polynomials where the polynomial variables correspond to the unknown random values. Specifically, $\mathcal{B}$ will associate encodings with polynomials. When $\mathcal{A}$ requests an oracle operation, $\mathcal{B}$ will do a computation on polynomials corresponding to the requested operation and will obtain another polynomial as a result. If $\mathcal{B}$ never encountered this polynomial before, a random unique encoding is generated for it, otherwise the previously issued encoding is returned.
The idea is that with high probability, $\mathcal{B}$ can provide an accurate simulation for $\mathcal{A}$ without having to know the concrete random values for the input variables. If $\mathcal{B}$'s simulation is successful, then $\mathcal{A}$'s guess $b' \in \{0,1\}$ is independent of the actual solution $b \xleftarrow{R} \{0,1\}$ (since b is not known to either $\mathcal{A}$ nor $\mathcal{B}$) and in these cases, $\mathcal{A}$ only succeeds with probability $1/2$. The main concern of the proof will be to bound the probability that $\mathcal{B}$'s simulation fails.
The proof is structured as follows: First, we will describe the initial setup and define how $\mathcal{B}$ answers oracle queries. Then we will analyze under what circumstances $\mathcal{B}$'s encoding responses deviate from an oracle's for a concrete input, then bound the probability for this to happen. Finally, we will bound $\mathcal{A}$'s success probability when it's interacting with a proper oracle.

Setup

First, we define two lists for $\mathcal{B}$ to maintain. Let $L_0 \in (\mathbb{F}_p[X_1,\ldots,X_n] \times S)^*$ and $L_1 \in (\mathbb{F}_p[X_1,\ldots,X_n,Y_0,Y_1] \times S)^*$ be two lists where each entry holds a polynomial and an encoding. Initially, we set

$$L_0 = ((p_1,\xi_{0,1}),\ldots,(p_s,\xi_{0,s}))$$

and

$$L_1 = ((q_1,\xi_{1,1}),\ldots,(q_s,\xi_{1,s}),(q_{s+1},\xi_{1,s+1}),(q_{s+2},\xi_{1,s+2}))$$

with $p_1,\ldots,p_s$ as defined in P, $q_1,\ldots,q_s$ as defined in Q, $q_{s+1} = Y_0$ and $q_{s+2} = Y_1 \in \mathbb{F}_p[X_1,\ldots,X_n,Y_0,Y_1]$.
The $\xi_{*,*} \in S$ values are random encodings subject to $p_k = p_l \Leftrightarrow \xi_{0,k} = \xi_{0,l}$ for all $k,l \in \{1,\ldots,s\}$ and $q_k = q_l \Leftrightarrow \xi_{1,k} = \xi_{1,l}$ for all $k,l \in \{1,\ldots,s+2\}$.
Over the course of $\mathcal{B}$'s execution, the lists will be updated by appending new pairs. We always write $(p_i,\xi_{0,i})$ for the ith entry of L_0 and $(q_i,\xi_{1,i})$ for the ith entry of L_1. Furthermore, let τ_0 be the length of L_0 and τ_1 the length of L_1.

After this initial setup, $\mathcal{B}$ supplies the vector $(\xi_{0,1},\ldots,\xi_{0,s},\xi_{1,1},\ldots,\xi_{1,s+2})$ to $\mathcal{A}$ as its input.
Therefore, for $i \in \{1,\ldots,s\}$, $\mathcal{A}$ associates the encoding $\xi_{0,i}$ with $g^{p_i(x_1,\ldots,x_n)} \in G_0$ and $\xi_{1,i}$ with $e(g,g)^{q_i(x_1,\ldots,x_n)} \in G_1$, as well as $\xi_{1,s+1}$ with $e(g,g)^{t_0} \in G_1$ and $\xi_{1,s+2}$ with $e(g,g)^{t_1} \in G_1$.

Our simulation $\mathcal{B}$ will associate the encodings with polynomials. The polynomial variables $X_1,\ldots,X_n$ correspond to the random $x_1,\ldots,x_n \in \mathbb{F}_p$. Y_0 and Y_1 correspond to t_0

and t_1 respectively. Notice that the concrete values for $x_1, \ldots, x_n, t_0, t_1$ are not known to $\mathcal{B}$.

Consider this overview:

Encoding	$\xi_{0,i}$	$\xi_{1,i}$
Corresponding for $\mathcal{B}$	$p_i \in \mathbb{F}_p[X_1, \ldots, X_n]$	$q_i \in \mathbb{F}_p[X_1, \ldots, X_n, Y_0, Y_1]$
Corresponding for $\mathcal{A}$	$g^{p_i(x_1,\ldots,x_n)} \in G_0$	$e(g,g)^{q_i(x_1,\ldots,x_n,t_0,t_1)} \in G_1$

Handling of Queries

$\mathcal{A}$ can query the simulated oracle for group operations in G_0 and G_1 and for the bilinear map e. We will now define how $\mathcal{B}$ will respond to such requests.

- *Group operation* in G_0: The algorithm $\mathcal{A}$ specifies two indices $1 \leq i, j \leq \tau_0$ from its G_0 encoding list and a sign bit. Depending on the sign bit, $\mathcal{A}$ then expects $\sigma_0(g^{p_i(x_1,\ldots,x_n) \pm p_j(x_1,\ldots,x_n)})$ to be appended to its encoding list.
  $\mathcal{B}$ computes $p_{\tau_0+1} = p_i \pm p_j \in \mathbb{F}_p[X_1, \ldots, X_n]$ according to the sign bit.
  If $p_{\tau_0+1} = p_l$ for an $l \in \{1, \ldots, \tau_0\}$, we set $\xi_{0,\tau_0+1} = \xi_{0,l}$. Otherwise, we set ξ_{0,τ_0+1} to a new random encoding out of $S \setminus \{\xi_{0,1}, \ldots, \xi_{0,\tau_0}\}$.
  Finally, we append ξ_{0,τ_0+1} to $\mathcal{A}$'s encoding list for G_0, and $(p_{\tau_0+1}, \xi_{0,\tau_0+1})$ to L_0 (then τ_0 is incremented).

- *Group operation* in G_1: This is analogous to the group operation in G_0:
  Given two indices $1 \leq i, j \leq \tau_1$ and a sign bit, $\mathcal{B}$ computes $q_{\tau_1+1} = q_i \pm q_j \in \mathbb{F}_p[X_1, \ldots, X_n, Y_0, Y_1]$ according to the sign bit.
  If $q_{\tau_1+1} = q_l$ for an $l \in \{1, \ldots, \tau_1\}$, we set $\xi_{1,\tau_1+1} = \xi_{1,l}$, otherwise we set ξ_{1,τ_1+1} to a new random bit-string out of $S \setminus \{\xi_{1,1}, \ldots, \xi_{1,\tau_1}\}$
  Finally, we append ξ_{1,τ_1+1} to $\mathcal{A}$'s encoding list for G_1, and $(q_{\tau_1+1}, \xi_{1,\tau_1+1})$ to L_1 (then τ_1 is incremented).

- *Bilinear map*: $\mathcal{A}$ specifies two indices $1 \leq i, j \leq \tau_0$ from its encoding list for G_0. It then expects $\sigma_1(e(g^{p_i(x_1,\ldots,x_n)}, g^{p_j(x_1,\ldots,x_n)}))$, which is equal to $\sigma_1(e(g,g)^{p_i(x_1,\ldots,x_n) \cdot p_j(x_1,\ldots,x_n)})$, to be appended to its encoding list.
  $\mathcal{B}$ computes $q_{\tau_1+1} = p_i \cdot p_j \in \mathbb{F}_p[X_1, \ldots, X_n] \subseteq \mathbb{F}_p[X_1, \ldots, X_n, Y_0, Y_1]$.
  If $q_{\tau_1+1} = q_l$ for some $l \in \{1, \ldots, \tau_1\}$, we set $\xi_{1,\tau_1+1} = \xi_{1,l}$, otherwise we set ξ_{1,τ_1+1} to a new random encoding out of $S \setminus \{\xi_{1,1}, \ldots, \xi_{1,\tau_1}\}$.
  Finally, we append ξ_{1,τ_1+1} to $\mathcal{A}$'s encoding list for G_1, and $(q_{\tau_1+1}, \xi_{1,\tau_1+1})$ to L_1 (then τ_1 is incremented).

Without loss of generality, we assume that $q+2s+2 \leq |S|$ since otherwise $q+2s+2 \geq p$ in which case the inequality stated in the theorem holds trivially. Consequently, the operations above are well-defined and there are enough encodings in S for the $2s+2$ initial list entries and at most q queries by $\mathcal{A}$.

We will use the following invariants about the lists L_0 and L_1: Let $1 \leq i, i' \leq \tau_0$ and $1 \leq j, j' \leq \tau_1$, then

- $\xi_{0,i}$ corresponds to $g^{p_i(x_1,\ldots,x_n)} \in G_0$ for $\mathcal{A}$
- $\xi_{1,j}$ corresponds to $e(g,g)^{q_j(x_1,\ldots,x_n,t_0,t_1)} \in G_1$ for $\mathcal{A}$
- $\xi_{0,i} = \xi_{0,i'} \Leftrightarrow p_i = p_{i'}$ and $\xi_{1,j} = \xi_{1,j'} \Leftrightarrow q_j = q_{j'}$
- p_i can be written as $\sum_{k=1}^{s} a_k p_k$ for some $a_k \in \mathbb{F}_p$ for $k \in \{1,\ldots,s\}$.
- q_j can be written as $\sum_{k=1}^{s}\sum_{l=1}^{s} a_{kl}p_k p_l + \sum_{u=1}^{s} b_u q_u + c_0 Y_0 + c_1 Y_1$ for some $a_{kl}, b_u, c_0, c_1 \in \mathbb{F}_p$ for $k,l,u \in \{1,\ldots,s\}$

Initially, these invariants hold by definition of the initial values of L_0, L_1. It is easy to see that the operations above preserve the invariant.

After at most q queries, $\mathcal{A}$ terminates and returns a guess $b' \in \{0,1\}$.

Analysis of Simulation Failure

Now we analyze the circumstances under which the simulation provided by $\mathcal{B}$ deviates from the behavior of an oracle for concrete values of $x_1,\ldots,x_n,y \xleftarrow{R} \mathbb{F}_p$ and $b \xleftarrow{R} \{0,1\}$. Essentially, such a deviation might occur if $\mathcal{B}$ chooses to generate a new encoding for a query because a polynomial is unequal to the others in the list. But when the polynomial variables are substituted with their concrete $x_1,\ldots,x_n,y$ counterparts, this decision might turn out wrong and may have caused $\mathcal{B}$ to supply two different encodings for the same group element. The following considerations will express this formally and show that this is the only source of errors in the simulation. We want to know for a concrete execution of $\mathcal{B}$ whether there exist encoding functions such that an oracle with those encoding functions would have supplied the same encodings as $\mathcal{B}$ has.

For this, we consider the encodings that $\mathcal{B}$ returned for the queries. Our invariants state that for $1 \leq i \leq \tau_0$ and $1 \leq j \leq \tau_1$, $\mathcal{A}$ associates each encoding $\xi_{0,i}$ with $g^{p_i(x_1,\ldots,x_n)}$ and $\xi_{1,j}$ with $e(g,g)^{q_j(x_1,\ldots,x_n,t_0,t_1)}$.
Therefore, the simulation was successful if and only if injective encoding functions $\sigma_0 : G_0 \to S$ and $\sigma_1 : G_1 \to S$ exist with

$$\sigma_0(g^{p_i(x_1,\ldots,x_n)}) = \xi_{0,i}$$

and

$$\sigma_1(e(g,g)^{q_j(x_1,\ldots,x_n,t_0,t_1)}) = \xi_{1,j}$$

for all $1 \leq i \leq \tau_0$ and $1 \leq j \leq \tau_1$, because then a proper oracle with those encoding functions would have supplied the same encodings to the algorithm for the input vector $I(\sigma_0, \sigma_1; x_1, \ldots, x_n, t_0, t_1)$ as $\mathcal{B}$ did.
We can find such injective encoding functions if and only if

$$\xi_{0,i} = \xi_{0,i'} \Leftrightarrow g^{p_i(x_1,\ldots,x_n)} = g^{p_{i'}(x_1,\ldots,x_n)}$$

$$\text{and} \tag{4.1}$$

$$\xi_{1,j} = \xi_{1,j'} \Leftrightarrow e(g,g)^{q_j(x_1,\ldots,x_n,t_0,t_1)} = e(g,g)^{q_{j'}(x_1,\ldots,x_n,t_0,t_1)}$$

for all $1 \leq i, i' \leq \tau_0$ and $1 \leq j, j' \leq \tau_1$.
This is because for such functions to exist, a group element must not have two different encodings (i.e. two different images in S). Also, two different group elements must not have the same encodings in order to satisfy injectivity. If these conditions are met, then by setting the images of the group elements in L_0, L_1 to their encodings as required and setting the other group elements to arbitrary values, preserving injectivity (which is possible since we required $|S|$ to be large enough), such encoding functions can be found.

Let $1 \leq i, i' \leq \tau_0$ and $1 \leq j, j' \leq \tau_1$. Because g and $e(g,g)$ are generators, $\varphi_0 : \mathbb{F}_p \to G_0,\ a \mapsto g^a$ and $\varphi_1 : \mathbb{F}_p \to G_1,\ a \mapsto e(g,g)^a$ are well-defined bijective maps. Therefore, $g^{p_i(x_1,\ldots,x_n)} = g^{p_{i'}(x_1,\ldots,x_n)} \Leftrightarrow p_i(x_1,\ldots,x_n) = p_{i'}(x_1,\ldots,x_n)$ and $e(g,g)^{q_j(x_1,\ldots,x_n,t_0,t_1)} = e(g,g)^{q_{j'}(x_1,\ldots,x_n,t_0,t_1)} \Leftrightarrow q_j(x_1,\ldots,x_n,t_0,t_1) = q_{j'}(x_1,\ldots,x_n,t_0,t_1)$.
Throughout the algorithm we have maintained the invariant that $\xi_{0,i} = \xi_{0,i'} \Leftrightarrow p_i = p_{i'}$ and $\xi_{0,j} = \xi_{0,j'} \Leftrightarrow q_j = q_{j'}$. It follows that condition (4.1) is equivalent to the following:

$$p_i = p_{i'} \Leftrightarrow p_i(x_1,\ldots,x_n) = p_{i'}(x_1,\ldots,x_n)$$

$$\text{and} \tag{4.2}$$

$$q_j = q_{j'} \Leftrightarrow q_j(x_1,\ldots,x_n,t_0,t_1) = q_{j'}(x_1,\ldots,x_n,t_0,t_1)$$

for all $1 \leq i, i' \leq \tau_0$ and $1 \leq j, j' \leq \tau_1$.

Now we substitute Y_b with $f(X_1,\ldots,X_n)$ in the polynomials $q_j \in \mathbb{F}_p[X_1,\ldots,X_n,Y_0,Y_1]$, $1 \leq j \leq \tau_1$ and call the results $q'_j \in \mathbb{F}_p[X_1,\ldots,X_n,Y_{1-b}]$. Since by definition, $t_b = f(x_1,\ldots,x_n)$, we have that $q_j(x_1,\ldots,x_n,t_0,t_1) = q'_j(x_1,\ldots,x_n,t_{1-b})$.
The intuition is that the substitution of Y_b is needed for probability analysis later, since the value of $t_b = f(x_1,\ldots,x_n)$ is not uniformly distributed in $\mathbb{F}_p$ but determined by the random $x_1,\ldots,x_n$.

We will now show that $q_i = q_j \Leftrightarrow q'_i = q'_j$, i.e. the substitution does not introduce any new equalities between polynomials.
Let $1 \leq i, j \leq \tau_1$ be arbitrary indices.

$q_i = q_j \Rightarrow q'_i = q'_j$ follows immediately, as q'_i and q'_j are the results of a substitution. We are left to show that $q'_i = q'_j \Rightarrow q_i = q_j$ or, equivalently, $q_i \neq q_j \Rightarrow q'_i \neq q'_j$.

Let $q_i \neq q_j$. Because of our invariant, $q_i - q_j$ can be written as

$$q_i - q_j = \sum_{k=1}^{s}\sum_{l=1}^{s} a_{kl}p_k p_l + \sum_{u=1}^{s} b_u q_u + c_0 Y_0 + c_1 Y_1$$

for some $a_{kl}, b_u, c_0, c_1 \in \mathbb{F}_p$ for $k, l, u \in \{1, \ldots, s\}$.

If $c_b = 0$, then clearly $q_i - q_j = q'_i - q'_j$ and since $q_i - q_j \neq 0$, it follows that $q'_i - q'_j \neq 0$. Therefore, $q'_i \neq q'_j$.
Otherwise, we consider $c_b \neq 0$. Assume $q'_i - q'_j = 0$, then we have

$$f = -c_b^{-1}\left(\sum_{k=1}^{s}\sum_{l=1}^{s} a_{kl}p_k p_l + \sum_{u=1}^{s} b_u q_u + c_{1-b}Y_{1-b}\right)$$

Since Y_{1-b} is not a variable in $f \in \mathbb{F}[X_1, \ldots, X_n]$, it follows that $c_{1-b} = 0$ and therefore

$$f = \left(\sum_{k=1}^{s}\sum_{l=1}^{s} (-c_b^{-1}a_{kl})p_k p_l + \sum_{u=1}^{s} (-c_b^{-1}b_u)q_u\right)$$

This would violate our assumption that f is independent of (P, Q). Consequently, our assumption that $q'_i - q'_j = 0$ leads to a contradiction. Thus, in this case $q'_i \neq q'_j$ holds as well.

In conclusion, we have that $q_i = q_j \Leftrightarrow q'_i = q'_j$ and that $q_j(x_1, \ldots, x_n, t_0, t_1) = q'_j(x_1, \ldots, x_n, t_{1-b}) = q'_j(x_1, \ldots, x_n, y)$ (by definition of t_{1-b}). We can now reformulate our previous condition:
The simulation was successful if and only if

$$p_i = p_{i'} \Leftrightarrow p_i(x_1, \ldots, x_n) = p_{i'}(x_1, \ldots, x_n)$$

and

$$q'_j = q'_{j'} \Leftrightarrow q'_j(x_1, \ldots, x_n, y) = q'_{j'}(x_1, \ldots, x_n, y)$$

for all $1 \leq i, i' \leq \tau_0$ and $1 \leq j, j' \leq \tau_1$.

Notice that $p_i = p_{i'} \Rightarrow p_i(x_1, \ldots, x_n) = p_{i'}(x_1, \ldots, x_n)$ and $q'_j = q'_{j'} \Rightarrow q'_j(x_1, \ldots, x_n, y) = q'_{j'}(x_1, \ldots, x_n, y)$ hold trivially.

As a result, the simulation *failed*, if and only if

$$p_i(x_1, \ldots, x_n) = p_{i'}(x_1, \ldots, x_n), \text{ but } p_i \neq p_{i'}$$

or

$$q'_j(x_1,\ldots,x_n,y) = q'_{j'}(x_1,\ldots,x_n,y), \text{ but } q'_j \neq q'_{j'}$$

for any $1 \leq i, i' \leq \tau_0,\ \ 1 \leq j, j' \leq \tau_1$.

Probability of Simulation Failure

We now bound the probability for $\mathcal{B}$'s simulation to fail, i.e. for two unequal polynomials in the lists L_0, L_1 to evaluate to the same value for $x_1,\ldots,x_n,y \xleftarrow{R} \mathbb{F}_p$.
First, for $1 \leq i \leq \tau_0$ and $1 \leq j \leq \tau_1$, the definition of $d = \max\{2\deg(p_k), \deg(q_k), \deg(f) \mid k \in \{1,\ldots,s\}\}$ and our invariants imply that

$$\deg(p_i) = \deg\left(\sum_{k=1}^{s} a_k p_k\right) \leq \max\{\deg(p_k) \mid k \in \{1,\ldots,s\}\} \leq d$$

(for appropriate $a_k \in \mathbb{F}_p$ where $k \in \{1,\ldots,s\}$) and that

$$\deg(q'_j) = \deg\left(\sum_{k=1}^{s}\sum_{l=1}^{s} a_{kl} p_k p_l + \sum_{u=1}^{s} b_u q_u + c_{1-b} Y_{1-b} + c_b f\right)$$

$$\leq \max\{2\deg(p_k), \deg(q_k), \deg(f) \mid k \in \{1,\ldots,s\}\} = d$$

(for appropriate $a_{kl}, b_u, c_0, c_1 \in \mathbb{F}_p$ where $k, l, u \in \{1,\ldots,s\}$), using that $\deg(f) \geq 1$ as f would be dependent on (P, Q) otherwise.

From this we can conclude that for $1 \leq i, i' \leq \tau_0,\ \ 1 \leq j, j' \leq \tau_1$ with $p_i \neq p_{i'}$ and $q'_j \neq q'_{j'}$

$$\Pr[p_i(x_1,\ldots,x_n) = p_j(x_1,\ldots,x_n)] = \Pr[(p_i - p_j)(x_1,\ldots,x_n) = 0] \leq d/p$$

$$\Pr[q'_j(x_1,\ldots,x_n,y) = q'_{j'}(x_1,\ldots,x_n,y)] = \Pr[(q'_j - q'_{j'})(x_1,\ldots,x_n,y) = 0] \leq d/p$$

(for random $x_1,\ldots,x_n,y \xleftarrow{R} \mathbb{F}_p$) using that the polynomials' degrees are at most d and the lemma of Schwartz–Zippel (Lemma 10).

Initially, L_0, L_1 contained $2s+2$ entries combined. After at most q queries, each of which adds exactly one entry to one of the lists, there are now at most $q+2s+2$ entries in L_0, L_1 combined. This implies that there are less than $\binom{q+2s+2}{2}$ pairs $\{p_i, p_{i'}\}$ with $p_i \neq p_{i'}$ and $\{q'_j, q'_{j'}\}$ with $q'_j \neq q'_{j'}$ $(1 \leq i, i' \leq \tau_0,\ \ 1 \leq j, j' \leq \tau_1)$. Each such pair has a chance of at most d/p that the polynomials evaluate to the same value for the random choice of $x_1,\ldots,x_n,y$, which must happen at least once for $\mathcal{B}$'s simulation to fail. Therefore, let 'fail' be the event that the simulation was not successful, then

$$\Pr[\text{fail}] \leq \binom{q+2s+2}{2}\frac{d}{p} = \frac{(q+2s+2)!\ d}{(q+2s)!\ 2!\ p} \leq \frac{(q+2s+2)^2 d}{2p}$$

where the probability is over the random bits of $\mathcal{A}$ and of $\mathcal{B}$ and $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, b \xleftarrow{R} \{0,1\}$.

Analysis of $\mathcal{A}$'s Success Probability

Up until now, we only analyzed the experiment run by $\mathcal{B}$ and bounded the probability that $\mathcal{B}$ fails to provide a simulation that conforms to an oracle's behavior for random values of $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, b \xleftarrow{R} \{0,1\}$.
The last step will be to use these findings to draw conclusions about $\mathcal{A}$'s probability of success when it interacts with an oracle.

First, we define a probability space that contains the random bits of $\mathcal{A}$ and $\mathcal{B}$ as well as random variables $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p$, $b \xleftarrow{R} \{0,1\}$ and $t_b \leftarrow f(x_1, \ldots, x_n), t_{1-b} \leftarrow y$.

Furthermore, let σ_0 be a random encoding function such that $\sigma_0(g^{p_i(x_1,\ldots,x_n)}) = \xi_{0,i}$ for $1 \leq i \leq k_0$ where $1 \leq k_0 \leq \tau_0$ is the maximal index in the encoding list L_0 such that for all $k, k' \leq k_0$: $p_k = p_{k'} \Leftrightarrow p_k(x_1, \ldots, x_n) = p_{k'}(x_1, \ldots, x_n)$ (that is the greatest index where the simulation did not fail for $x_1, \ldots, x_n, y, b$).
Similarly, let σ_1 be a random encoding function such that $\sigma_1(e(g,g)^{q_j(x_1,\ldots,x_n,t_0,t_1)}) = \xi_{1,j}$ for $1 \leq j \leq k_0$ where $1 \leq k_0 \leq \tau_1$ is the maximal index in the encoding list L_1 such that for all $k, k' \leq k_0$: $q_k = q_{k'} \Leftrightarrow q_k(x_1, \ldots, x_n, t_0, t_1) = q_{k'}(x_1, \ldots, x_n, t_0, t_1)$. (cf. condition (4.2) above).
Since $\mathcal{B}$ chooses the encodings $\xi_{*,*}$ randomly during the simulation, σ_0 and σ_1 are uniformly distributed over the set of injective functions $G_0 \to S$ and $G_1 \to S$ respectively.

Let fail be the event that $\mathcal{B}$'s simulation is not successful (as described in the *analysis of simulation failure*).
We are interested in the values of two random variables:
Let b' be the bit that $\mathcal{A}$ returns after interacting with $\mathcal{B}$.
Let b'' be the bit that $\mathcal{A}$ returns for input $I(\sigma_0, \sigma_1; x_1, \ldots, x_n, t_0, t_1)$ after interacting with an oracle that issues encodings according to σ_0, σ_1 as chosen above.
Notice that b' is completely determined by the random bits of $\mathcal{A}$ and $\mathcal{B}$. b'' is completely determined by $x_1, \ldots, x_n, y, b, \sigma_0, \sigma_1$ and $\mathcal{A}$'s random bits.

It holds that $1/2 = \Pr[b = b'] = \Pr[b = b' \mid \neg\text{fail}]$, because we choose $b \xleftarrow{R} \{0,1\}$ independently of b' and the event fail.
If $\mathcal{B}$'s simulation is successful, i.e. event fail does not occur, then (and only then) the encodings issued by $\mathcal{B}$ conform to some encoding functions for the input that is determined by $x_1, \ldots, x_n, y, b$. In this case, σ_0, σ_1 as defined above are such encoding functions.
It follows that whenever event $\neg$fail occurs, $\mathcal{B}$ supplies the same encodings to $\mathcal{A}$ as an oracle that issues encodings according to σ_0, σ_1. Since $\mathcal{A}$'s output is only determined by the supplied encodings (which are the same in both cases) and its random bits, we have

that $\neg\text{fail}$ implies $b' = b''$.

Formally, this further implies:

$$\Pr[b = b'' \mid \neg\text{fail}] = \Pr[b = b' \mid \neg\text{fail}] = 1/2$$

Hence,

$$\begin{aligned}\Pr[b = b''] &= \Pr[b = b'' \mid \neg\text{fail}]\Pr[\neg\text{fail}] + \Pr[b = b'' \mid \text{fail}]\Pr[\text{fail}] \\ &\leq \Pr[b = b'' \mid \neg\text{fail}](1 - \Pr[\text{fail}]) + \Pr[\text{fail}] = 1/2(1 - \Pr[\text{fail}]) + \Pr[\text{fail}] \\ &= 1/2 + \Pr[\text{fail}]/2\end{aligned}$$

and

$$\begin{aligned}\Pr[b = b''] &= \Pr[b = b'' \mid \neg\text{fail}]\Pr[\neg\text{fail}] + \Pr[b = b'' \mid \text{fail}]\Pr[\text{fail}] \\ &\geq \Pr[b = b'' \mid \neg\text{fail}]\Pr[\neg\text{fail}] = 1/2(1 - \Pr[\text{fail}]) \\ &= 1/2 - \Pr[\text{fail}]/2\end{aligned}$$

Finally, this implies

$$-\Pr[\text{fail}]/2 \leq \Pr[b = b''] - 1/2 \leq \Pr[\text{fail}]/2$$

$$\Rightarrow |\Pr[b = b''] - 1/2| \leq \Pr[\text{fail}]/2 \leq \frac{(q + 2s + 2)^2 d}{4p}$$

Where b'' is the value that $\mathcal{A}$ returns for input $I(\sigma_0, \sigma_1; x_1, \ldots, x_n, t_0, t_1)$ after interacting with an oracle that uses the encoding functions σ_0, σ_1. The probability is over $\sigma_0 \xleftarrow{R} \{\sigma : G_0 \to S \mid \sigma \text{ injective}\}, \sigma_1 \xleftarrow{R} \{\sigma : G_1 \to S \mid \sigma \text{ injective}\}$ and $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p$, $b \xleftarrow{R} \{0, 1\}$ and the random bits of $\mathcal{A}$ as required. □

The following corollary uses Theorem 20 to bound the advantage of generic algorithms for the decisional (P, Q, f)-Diffie-Hellman problem (Definition 18).

Corollary 21. *Let $p \in \mathbb{N}$ be a prime number, $((G_0, \cdot), (G_1, \cdot), e)$ a bilinear group of order p with generator $g \in G_0$. Let $S \subset \{0, 1\}^*$ with $p \leq |S| < \infty$*
Let $s, n \in \mathbb{N}$ and $P = (p_1, \ldots, p_s), Q = (q_1, \ldots, q_s) \in \mathbb{F}_p[X_1, \ldots, X_n]^s$ with $p_1 = q_1 = 1$, $f \in \mathbb{F}_p[X_1, \ldots, X_n]$. Let $d = \max\{2\deg(p_i), \deg(q_i), \deg(f) \mid i \in \{1, \ldots, s\}\}$.

If f is independent of (P, Q), then any generic algorithm $\mathcal{A}$ for $((G_0, \cdot), (G_1, \cdot), e)$ and S that makes at most q oracle queries has at most advantage $\frac{(q+2s+2)^2 \cdot d}{2p}$ in solving the (P, Q, f)-Diffie-Hellman problem (Definition 18).

Proof. For the proof, we construct an algorithm $\mathcal{B}$ that solves the problem described in

Theorem 20 by using an algorithm $\mathcal{A}$ for the (P, Q, f)-Diffie-Hellman problem. Theorem 20 will then imply bounds that apply for $\mathcal{A}$.

Let f be independent of (P, Q).
For notational convenience, let

$$I(\sigma_0, \sigma_1; x_1, \ldots, x_n) = \begin{pmatrix} \sigma_0(g^{p_1(x_1,\ldots,x_n)}), \ldots, \sigma_0(g^{p_s(x_1,\ldots,x_n)}), \\ \sigma_1(e(g,g)^{q_1(x_1,\ldots,x_n)}), \ldots, \sigma_1(e(g,g)^{q_s(x_1,\ldots,x_n)}) \end{pmatrix}$$

Let $\mathcal{A}$ be a generic algorithm for the decisional (P, Q, f)-Diffie-Hellman problem.

We write $\mathcal{A}(\text{correct}) = \mathcal{A}(I(\sigma_0, \sigma_1; x_1, \ldots, x_n), \sigma_1(e(g,g)^{f(x_1,\ldots,x_n)}))$ and
$\mathcal{A}(\text{random}) = \mathcal{A}(I(\sigma_0, \sigma_1; x_1, \ldots, x_n), \sigma_1(e(g,g)^{y}))$.

Consider an algorithm $\mathcal{B}$ that takes input $I(\sigma_0, \sigma_1; x_1, \ldots, x_n), \sigma_1(e(g,g)^{t_0}), \sigma_1(e(g,g)^{t_1})$ as in Theorem 20. $\mathcal{B}$ runs $\mathcal{A}$ with input $I(\sigma_0, \sigma_1; x_1, \ldots, x_n), \sigma_1(e(g,g)^{t_0})$ and returns $\mathcal{A}$'s output b'.

We bound the probability for $\mathcal{B}$'s success:
Let $b \xleftarrow{R} \{0,1\}$, $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p$, $t_b \leftarrow f(x_1, \ldots, x_n)$, $t_{1-b} \leftarrow y$.

$$\begin{aligned} \Pr[b = b'] &= \Pr[b = b' \mid b = 0]\Pr[b = 0] + \Pr[b = b' \mid b = 1]\Pr[b = 1] \\ &= 1/2(\Pr[b = b' \mid b = 0] + \Pr[b = b' \mid b = 1]) \end{aligned}$$

If $b = 0$, then $t_0 = f(x_1, \ldots, x_n)$. $\mathcal{B}$ supplies the correct solution to $\mathcal{A}$. Here, $\mathcal{B}$ is successful if and only if $\mathcal{A}$ returns 0. Therefore

$$\Pr[b = b' \mid b = 0] = \Pr[\mathcal{A}(\text{correct}) = 0]$$

If $b = 1$, then $\mathcal{B}$ supplies a random (encoded) group element to $\mathcal{A}$. In this case, $\mathcal{B}$ is successful if and only if $\mathcal{A}$ returns 1:

$$\begin{aligned} \Pr[b = b' \mid b = 1] &= \Pr[\mathcal{A}(\text{random}) = 1] \\ &= 1 - \Pr[\mathcal{A}(\text{random}) = 0] \end{aligned}$$

Consequently,

$$\begin{aligned} &\Pr[b = b'] = 1/2 * (\Pr[\mathcal{A}(\text{correct}) = 0] + 1 - \Pr[\mathcal{A}(\text{random}) = 0]) \\ \Leftrightarrow &\Pr[b = b'] - 1/2 = 1/2(\Pr[\mathcal{A}(\text{correct}) = 0] - \Pr[\mathcal{A}(\text{random}) = 0]) \\ \Rightarrow &|\Pr[b = b'] - 1/2| = 1/2 \ |\Pr[\mathcal{A}(\text{correct}) = 0] - \Pr[\mathcal{A}(\text{random}) = 0]| \end{aligned}$$

Theorem 20 implies that $|\Pr[b = b'] - 1/2| \leq \frac{(q+2s+2)^2 \cdot d}{4p}$. It follows that

$$|\Pr[\mathcal{A}(\text{correct}) = 0] - \Pr[\mathcal{A}(\text{random}) = 0]| \leq \frac{(q + 2s + 2)^2 \cdot d}{2p}$$

where the probability is over $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{Z}_p, \sigma_0, \sigma_1$ and $\mathcal{A}$'s random bits. □

4.4 Applications of the Framework

As discussed in [BBG05], the framework can be applied to many standard assumptions. We shortly suggest how the framework can be used for the problems introduced in Section 2 to show that they are hard for generic algorithms.

- Decisional Diffie-Hellman (Problem 4) using $P = (1, 1, 1), Q = (1, A, B), f = A \cdot B$.
- Decisional bilinear Diffie-Hellman (Problem 8) using $P = (1, A, B, C), Q = (1, 1, 1, 1)$, $f = A \cdot B \cdot C$.

5 Extending the Boneh, Boyen, Goh Framework for Polynomials with Negative Exponents

Some cryptographic assumptions cannot be proven directly using the framework of [BBG05] (Section 4). One simple example for this is the *decisional modified bilinear Diffie-Hellman assumption* (Definition 34). Loosely speaking, the assumption states that given elements g^a, g^b, g^c from a bilinear group it must be hard to distinguish $e(g,g)^{a \cdot b/c}$ from a random group element. To translate this problem into a (P, Q, f)-Diffie-Hellman problem, f would have to be in the form $A \cdot B \cdot C^{-1}$ which is not a proper polynomial because of the negative exponent. Consequently, the Boneh, Boyen, Goh framework cannot be applied.
Another example is the previously mentioned *decisional q-parallel bilinear Diffie-Hellman exponent problem* (Problem 30). In this section, we extend the Boneh, Boyen, Goh framework so that it can be applied in such cases where negative exponents are involved.

After that, we will show how the extended framework can be used to prove assumptions such as the decisional q-parallel BDHE assumption in the generic group model (Section 5.2).

5.1 The Extension

First, we formally describe what is meant by "polynomials with negative exponents" and define a degree function specifically devised for our purposes.

Definition 22. *Let R be a commutative ring, $n \in \mathbb{N}$, $f \in R[X_1, \ldots, X_n][X_1^{-1}, \ldots, X_n^{-1}]$. f can be written as $f = \sum_{(k_1,\ldots,k_n) \in I} \sigma(k_1, \ldots, k_n) \prod_{i=1}^{n} X_i^{k_i}$ for some finite set $I \subset \mathbb{Z}^n$ and $\sigma : I \to R$*

If R is a field, f is called a Laurent polynomial *in n variables. We define the* degree *of $f \neq 0$ as*

$$\deg(f) = \max \left\{ \sum_{i=1}^{n} |k_i| \;\middle|\; (k_1, \ldots, k_n) \in I,\ \sigma(k_1, \ldots, k_n) \neq 0 \right\}$$

Notice that for $f \in R[X_1, \ldots, X_n] \subset R[X_1, \ldots, X_n][X_1^{-1}, \ldots, X_n^{-1}]$, this is consistent

with the usual definition of polynomial degrees. If there are variables with negative exponents, they are counted according to their absolute value. Also notice that for any Laurent polynomials g, h, it holds that $\deg(g \cdot h) \leq \deg(g) + \deg(h)$.

The extension of the Boneh, Boyen, Goh framework is possible because of the observation that the proof of Theorem 20 essentially only relies on the fact that the (P, Q, f)-Diffie-Hellman problem description consists of proper polynomials when the lemma of Schwartz-Zippel (Lemma 10) is invoked. As a replacement for the Schwartz-Zippel lemma we present a new lemma that applies to Laurent polynomials.

Lemma 23. *Let F be a field, $S \subseteq F$ a finite subset with $|S| > 1$, $n, m \in \mathbb{N}_0$, $f \in F[X_1, \ldots, X_n, Y_1, \ldots, Y_m][X_1^{-1}, \ldots, X_n^{-1}]$, $f \neq 0$.*
Then

$$\Pr[f(x_1, \ldots, x_n, y_1, \ldots, y_m) = 0] \leq 2d/(|S| - 1)$$

where $d = \deg(f)$ and the probability is over $x_1, \ldots, x_n \xleftarrow{R} S \setminus \{0\}$, $y_1, \ldots, y_m \xleftarrow{R} S$.

Proof. Similar to Lemma 10, will prove this by induction over the number of variables $n + m$.
For $n + m = 0$, the statement holds trivially, since $f \in F \setminus \{0\}$.
As another base case, let $n + m = 1$. If $n = 0$, then Lemma 10 implies $\Pr[f(y) = 0] \leq d/|S| \leq 2d/(|S| - 1)$.
If $n = 1$, then $f \in F[X][X^{-1}]$. In this case, $f \cdot X^d \in F[X]$ is a polynomial of degree at most $2d$ and therefore has at most $2d$ roots. Because every root of f is also a root of $f \cdot X^d$, there are at most $2d$ roots for f. It follow that $\Pr[f(x) = 0] \leq 2d/(|S| - 1)$ (there are at least $|S \setminus \{0\}| \geq |S| - 1$ values to choose from for x).

For the inductive step, let $n + m > 1$ and we assume that the statement holds for any Laurent polynomial with fewer than $n + m$ variables.
If $n = 0$, then $f \in F[Y_1, \ldots, Y_m]$ and in this case, the lemma of Schwartz-Zippel (Lemma 10) directly implies the necessary statement: $\Pr[f(y_1, \ldots, y_m) = 0] \leq d/|S| < 2d/(|S| - 1)$.
If $n > 0$, we write

$$f = \sum_{i=-d}^{d} X_1^i f_i$$

where $f_i \in F[X_2, \ldots, X_n, Y_1, \ldots, Y_m][X_2^{-1}, \ldots, X_n^{-1}]$. Since $f \neq 0$, there is an index k where $f_k \neq 0$ and $f_j = 0$ for all j with $|j| > |k|$.

According to our definition of the degree, $d = \deg(f) \geq \deg(X_1^k \cdot f_k) = |k| + \deg(f_k)$ and therefore $\deg(f_k) \leq d - |k|$.

Because f_k is a Laurent polynomial in only $n+m-1$ variables, it follows from the induction hypothesis that $\Pr[f_k(x_2,\ldots,x_n,y_1,\ldots,y_m) = 0] \leq 2\deg(f_k)/(|S|-1) \leq 2(d-|k|)/(|S|-1)$.

Also, for any concrete $x_2,\ldots,x_n \in S \setminus \{0\},\ y_1,\ldots,y_m \in S$ consider

$$f' = f(X_1, x_2, \ldots, x_n, y_1, \ldots, y_m) \in F[X_1, X_1^{-1}]$$

If $0 \neq f_k(x_2,\ldots,x_n,y_1,\ldots,y_m) \in F$, then $\deg(f') = |k|$ as we chose k to be the (absolute) greatest index in $f = \sum_{i=-d}^{d} X_1^i f_i$ and by definition of f'.
Consequently, since f' is a Laurent polynomial in one variable, the induction hypothesis implies $\Pr[f'(x_1) = 0 \mid f_k(x_2,\ldots,x_n,y_1,\ldots,y_m) \neq 0] \leq 2\deg(f')/(|S|-1) = 2k/(|S|-1)$. By definition of f' it follows that
$\Pr[f(x_1,x_2,\ldots,x_n,y_1,\ldots,y_m) = 0 \mid f_k(x_2,\ldots,x_n,y_1,\ldots,y_m) \neq 0] \leq 2k/(|S|-1)$

Using the statements above, we have

$$\begin{aligned}
&\Pr[f(\cdot) = 0] \\
&= \Pr[f(\cdot) = 0 \mid f_k(\cdot) = 0] \cdot \Pr[f_k(\cdot) = 0] + \Pr[f(\cdot) = 0 \mid f_k(\cdot) \neq 0] \cdot \Pr[f_k(\cdot) \neq 0] \\
&\leq \Pr[f_k(\cdot) = 0] + \Pr[f(\cdot) = 0 \mid f_k(\cdot) \neq 0] \\
&\leq \frac{2(d-k)}{|S|-1} + \frac{2k}{|S|-1} \\
&= \frac{2d}{|S|-1}
\end{aligned}$$

□

We note that this bound is tight. For example, consider $F = S = \mathbb{F}_3$, $f = X - X^{-1} \in \mathbb{F}_3[X, X^{-1}]$. Since $f(1) = 0$ and $f(2) = 0$, we have that $\Pr[f(x) = 0] = 2/2 = 2\deg(f)/(|S|-1)$ (probability over $x \in S \setminus \{0\}$).

Before we present the central theorem for our extension, we define the decisional $(P,Q,f)_L$-Diffie-Hellman problem which is similar to Problem 17 but allows Laurent polynomials to be used in P, Q and for f.

Problem 24. *Let $((G_0,\cdot),(G_1,\cdot),e)$ be a bilinear group of prime order p with generator $g \in G_0$. Let $s,n \in \mathbb{N}$,*
$P = (p_1,\ldots,p_s), Q = (q_1,\ldots,q_s) \in (\mathbb{F}_p[X_1,\ldots,X_n,Z_1,\ldots,Z_n][Z_1^{-1},\ldots,Z_n^{-1}])^s$, $p_1 = q_1 = 1$ and $f \in \mathbb{F}_p[X_1,\ldots,X_n,Z_1,\ldots,Z_n][Z_1^{-1},\ldots,Z_n^{-1}]$.
The corresponding decisional $(P,Q,f)_L$-Diffie-Hellman problem *is defined as follows:*

Given

$$\left(g^{p_1(x,z)}, \ldots, g^{p_s(x,z)}, e(g,g)^{q_1(x,z)}, \ldots, e(g,g)^{q_s(x,z)}\right)$$

for $x = (x_1, \ldots, x_n) \xleftarrow{R} \mathbb{F}_p^n, z = (z_1, \ldots, z_n) \xleftarrow{R} (\mathbb{F}_p \setminus \{0\})^n$,
and an element $T \in G_1$, *decide whether or not*

$$T = e(g,g)^{f(x_1,\ldots,x_n)}$$

The Laurent polynomial variables in this problem are divided into $X_1, \ldots, X_n$ and $Z_1, \ldots, Z_n$. Variables that may appear inverted and therefore must not be evaluated with 0 are represented by $Z_1, \ldots, Z_n$. The variables that may be randomly picked from complete $\mathbb{F}_p$ are represented by $X_1, \ldots, X_n$ and those Laurent polynomial variables only appear with nonnegative exponents.

Definition 25. *In the situation of Problem 24, let*
$I_g(x,z) = \left(g^{p_1(x,z)}, \ldots, g^{p_s(x,z)}, e(g,g)^{q_1(x,z)}, \ldots, e(g,g)^{q_s(x,z)}\right)$
We say that a probabilistic algorithm $\mathcal{A}$ *has* advantage ϵ *in solving the decisional* $(P,Q,f)_L$*-Diffie-Hellman problem, if*

$$\left|\Pr[\mathcal{A}(I_g(x,z), e(g,g)^{f(x,z)}) = 0] - \Pr[\mathcal{A}(I_g(x,z), T) = 0]\right| > \epsilon$$

where the probability is over $x = (x_1, \ldots, x_n) \xleftarrow{R} \mathbb{F}_p^n$, $z = (z_1, \ldots, z_n) \xleftarrow{R} (\mathbb{F}_p \setminus \{0\})^n$, $T \xleftarrow{R} G_1$ *and* $\mathcal{A}$*'s random bits.*

We now present the central theorem of the extended framework based on Theorem 20.

Theorem 26. *Let* $p \in \mathbb{N}$ *be a prime number,* $((G_0,\cdot),(G_1,\cdot),e)$ *a bilinear group of order* p *with generator* $g \in G_0$. *Let* $S \subset \{0,1\}^*$ *with* $p \leq |S| < \infty$.
Further, let $s, n \in \mathbb{N}$ *and*
$P = (p_1, \ldots, p_s), Q = (q_1, \ldots, q_s) \in (\mathbb{F}_p[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}])^s$ *with* $p_1 = q_1 = 1$, $f \in \mathbb{F}_p[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}]$.
Let $d = \max\{2\deg(p_i), \deg(q_i), \deg(f) \mid i \in \{1, \ldots, s\}\}$ *(*deg *as in Definition 22).*
We set

$$I\left(\ \sigma_0, \sigma_1; x, z, t_0, t_1\ \right) = \begin{pmatrix} \sigma_0(g^{p_1(x,z)}), \ldots, \sigma_0(g^{p_s(x,z)}), \\ \sigma_1(e(g,g)^{q_1(x,z)}), \ldots, \sigma_1(e(g,g)^{q_s(x,z)}), \\ \sigma_1(e(g,g)^{t_0}), \sigma_1(e(g,g)^{t_1}) \end{pmatrix}$$

and call it the input vector

If f is independent of (P, Q) (analogous to Definition 19), then for any probabilistic generic algorithm $\mathcal{A}$ for $((G_0, \cdot), (G_1, \cdot), e)$ and S, that makes at most q queries to the oracle it holds that

$$\left| \Pr\left[\mathcal{A}\left(\begin{array}{c} I(\sigma_0, \sigma_1; \\ (x_1, \ldots, x_n), \\ (z_1, \ldots, z_n), \\ t_0, t_1) \end{array} \right) = b : \begin{array}{c} x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, \\ z_1, \ldots, z_n \xleftarrow{R} \mathbb{F}_p \setminus \{0\}, \\ b \xleftarrow{R} \{0,1\}, \\ t_b \leftarrow f(x_1, \ldots, x_n), \\ t_{1-b} \leftarrow y \end{array} \right] - \frac{1}{2} \right| \leq \frac{(q + 2s + 2)^2 \cdot d}{2(p-1)}$$

where the probability is over $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, z_1, \ldots, z_n \xleftarrow{R} \mathbb{F}_p \setminus \{0\}, b \xleftarrow{R} \{0,1\}, \sigma_0 \xleftarrow{R} \{\sigma : G_0 \to S \mid \sigma \text{ injective}\}, \sigma_1 \xleftarrow{R} \{\sigma : G_1 \to S \mid \sigma \text{ injective}\}$ and the random bits of $\mathcal{A}$.

Proof. With the help of Lemma 23, the proof for Theorem 20 is easily adapted to Laurent polynomials.
The setup of $\mathcal{B}$ and its handling of queries are essentially the same (the variables $Z_1, \ldots, Z_n$ need to be accounted for but can be treated exactly the same as $X_1, \ldots, X_n$ in the original proof). The algorithm $\mathcal{B}$ is still well-defined when using Laurent polynomials as the only operations are multiplication, addition and comparison of (Laurent) polynomials. The set of Laurent polynomials is a ring and this suffices for this stage of the proof.

The analysis of the circumstances under which $\mathcal{B}$'s simulation fails does not cause any problems when exercised with Laurent polynomials. It should be noted that the substitution of Y_b with f works as intended since Y_b still only occurs with exponent 1.

When analyzing the probability for $\mathcal{B}$'s simulation to fail, some statements need to be reconsidered. First, for the degree of Laurent polynomials f, g (Definition 22) it does not necessarily hold that $\deg(f \cdot g) = \deg(f) \cdot \deg(g)$. However, it is easy to see that $\deg(f \cdot g) \leq \deg(f) \cdot \deg(g)$. Consequently, the degrees of the Laurent polynomials p_i, q'_j in the proof (from the encoding lists maintained by $\mathcal{B}$) are still at most d when the new definition of the degree function is applied.
Instead of the lemma of Schwartz-Zippel (which does not apply to Laurent polynomials), we now need to apply Lemma 23 to bound the probability of two unequal Laurent polynomials to evaluate to the same value:

For $1 \leq i, i' \leq \tau_0,\ 1 \leq j, j' \leq \tau_1$ with $p_i \neq p_{i'}$ and $q'_j \neq q'_{j'}$

$$\Pr[(p_i - p_j)(x_1, \ldots, x_n, z_1, \ldots, z_n) = 0] \leq 2d/(p-1)$$

and

$$\Pr[(q'_j - q'_{j'})(x_1, \ldots, x_n, z_1, \ldots, z_n, y) = 0] \leq 2d/(p-1)$$

(for random $x_1, \ldots, x_n, y \xleftarrow{R} \mathbb{F}_p, z_1, \ldots, z_n \xleftarrow{R} \mathbb{F}_p \setminus \{0\}$) using that the polynomials' degrees are at most d and Lemma 23. (Also using that the set of Laurent polynomials forms a ring and therefore for Laurent polynomials g, h: $g = h \Leftrightarrow g - h = 0$ (so that Lemma 23 applies) and that evaluation of Laurent polynomials is still a ring homomorphism. Otherwise the probabilities above would not necessarily reflect the event that two unequal polynomials evaluate to the same value.)

Since the bound provided by Lemma 23 is not the same as the one by Schwartz-Zippel, the rest of the bounds need to be adapted. This results in

$$\Pr[\text{fail}] \leq \frac{(q+2s+2)^2 \cdot 2d}{2(p-1)}$$

and finally

$$|\Pr[b = b''] - 1/2| \leq \Pr[\text{fail}]/2 \leq \frac{(q+2s+2)^2 d}{2(p-1)}$$

□

This allows a corollary similar to Corollary 21.

Corollary 27. *Let $p \in \mathbb{N}$ be a prime number, $((G_0, \cdot), (G_1, \cdot), e)$ a bilinear group of order p with generator $g \in G_0$. Let $S \subset \{0,1\}^*$ with $p \leq |S| < \infty$*
Let $s, n \in \mathbb{N}$ and
$P = (p_1, \ldots, p_s), Q = (q_1, \ldots, q_s) \in (\mathbb{F}_p[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}])^s$ with $p_1 = q_1 = 1$, $f \in \mathbb{F}_p[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}]$.
Let $d = \max\{2\deg(p_i), \deg(q_i), \deg(f) \mid i \in \{1, \ldots, s\}\}$ (deg as in Definition 22) and $\epsilon > 0$.

If f is independent of (P, Q), then any generic algorithm $\mathcal{A}$ for $((G_0, \cdot), (G_1, \cdot), e)$ and S that makes at most q oracle queries has at most advantage $\frac{(q+2s+2)^2 \cdot d}{p-1}$ in solving the $(P, Q, f)_L$-Diffie-Hellman problem (Definition 25).

Proof. Analogous to Corollary 21. □

5.2 Applying the Extended Framework

We present a general argument for applying our findings above to typical assumptions.

For this, we first present a sufficient condition that in many (typical) cases allows to show that f is independent of (P, Q).

Lemma 28. *Let $p \in \mathbb{N}$ be a prime number, $s \in \mathbb{N}$ and $A \subset \mathbb{F}_p[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}]$ a set of (Laurent) monomials with coefficient 1 (i.e. terms in the form $X_1^{i_1} \cdot \ \ldots \ \cdot X_n^{i_n} \cdot Z_1^{j_1} \cdot \ \ldots \ \cdot Z_n^{j_n}$ where $i_1, \ldots, i_n \in \mathbb{N}_0, j_1, \ldots, j_n \in \mathbb{Z}$).*

If $P = (p_1, \ldots, p_s), Q = (q_1, \ldots, q_s) \in A^s$ and f is a (Laurent) monomial with coefficient 1, but $f \notin \{p \cdot p' \mid p, p' \in A \cup \{1\}\}$, then f is independent of (P, Q).

Proof. Let $B = \{p \cdot p' \mid p, p' \in A \cup \{1\}\}$.
Then $B \cup \{f\}$ is a set of (Laurent) monomials with coefficient 1. Consequently, $B \cup \{f\}$ is a set of linearly independent vectors over the F vector space of Laurent polynomials over F.
Particularly, f cannot be written as a linear combination of the elements in B.

If f were dependent on (P, Q), then there would be $a_{ij}, b_k \in F$ (for $i, j, k \in \{1, \ldots, s\}$) such that

$$f = \sum_{i=1}^{s} \sum_{j=1}^{s} a_{ij} p_i p_j + \sum_{k=1}^{s} b_k q_k$$

Since $p_i \cdot p_j, q_k \in B$, this is a linear combination of f, which contradicts our assumptions. Therefore, f must be independent of (P, Q). □

We now present a convenient theorem to show (decisional) assumptions that are defined asymptotically using the notion of negligible functions and a group generator $\mathcal{G}$ (such as the q-parallel BDHE assumption (Definition 31)).
The $(P, Q, f)_L$-Diffie-Hellman problem is defined for specific groups. In contrast, the following theorem is concerned with asymptotic security where groups are generated by an instance generator $\mathcal{G}$ according to a security parameter k.
Since the $(P, Q, f)_L$-Diffie-Hellman problem is defined through Laurent polynomials over $\mathbb{F}_p$, we cannot base the problem description on $(P, Q, f)_L$-Diffie-Hellman (as p is not constant but randomly generated). We will base the problem description on terms over $\mathbb{Z}$ instead and project them to Laurent polynomials over $\mathbb{F}_p$ using a homomorphism $\pi_p : \mathbb{Z}[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}] \to \mathbb{F}_p[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}]$ such that $\pi_p(a) = (a + p\mathbb{Z}) \in \mathbb{Z}_p$ for $a \in \mathbb{Z}$ and $\pi_p(X_i) = X_i, \pi_p(Z_i) = Z_i$ for $i \in \{1, \ldots, n\}$.

Theorem 29. *Let $s, n \in \mathbb{N}$ and $P = (p_1, \ldots, p_s), Q = (q_1, \ldots, q_s) \in (\mathbb{Z}[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}])^s$ with $p_1 = q_1 = 1$, $f \in \mathbb{Z}[X_1, \ldots, X_n, Z_1, \ldots, Z_n][Z_1^{-1}, \ldots, Z_n^{-1}]$ such that for all prime numbers $p \in \mathbb{N}$: $\pi_p(f)$ is independent of $((\pi_p(p_1), \ldots, \pi_p(p_n)), (\pi_p(q_1), \ldots, \pi_p(q_n)))$ (where π_p is the projection to $\mathbb{F}_p$ as discussed above).*

Let

$$I_g(\sigma_0, \sigma_1; x, z) = \begin{pmatrix} \sigma_0(g^{p_1(x,z)}), \ldots, \sigma_0(g^{p_s(x,z)}), \\ \sigma_1(e(g,g)^{q_1(x,z)}), \ldots, \sigma_1(e(g,g)^{q_s(x,z)}) \end{pmatrix}$$

For all generic polynomial time algorithms $\mathcal{A}$ (in k) that output 0 or 1 there is a negligible function negl *such that for all $k \in \mathbb{N}$:*

$$\left|\Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; x, z), \sigma_1(e(g,g)^{f(x,z)})) = 0] - \Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; x, z), \sigma_1(T)) = 0]\right| < \text{negl}(k)$$

where the probabilities are over $(((G_0, \cdot), (G_1, \cdot), e), p, g) \leftarrow \mathcal{G}(1^k)$ where $((G_0, \cdot), (G_1, \cdot), e)$ is a bilinear group of prime order $p > 2^k$ with generator g and $\sigma_0 \xleftarrow{R} \{\sigma : G_0 \to S \mid \sigma \text{ injective}\}, \sigma_1 \xleftarrow{R} \{\sigma : G_1 \to S \mid \sigma \text{ injective}\}, x = (x_1, \ldots, x_n) \xleftarrow{R} \mathbb{Z}_p^n, z = (z_1, \ldots, z_n) \xleftarrow{R} (\mathbb{Z}_p \setminus \{0\})^n, T \xleftarrow{R} G_1$ and the random bits of $\mathcal{A}$.

Proof. Let $d = \max\{2\deg(p_i), \deg(q_i), \deg(f) \mid i \in \{1, \ldots, s\}\}$ (degree as in Definition 22).
Let $(q_k)_{k\in\mathbb{N}}$ be a series where q_k is an upper bound for the number of oracle queries that $\mathcal{A}$ makes for security parameter k. Since $\mathcal{A}$ is a polynomial time algorithm in k, we assume $(q_k)_{k\in\mathbb{N}}$ to be polynomial in k.

We set $\text{negl} : \mathbb{N} \to \mathbb{R}$, $k \mapsto \frac{(q_k+2s+2)^2 \cdot d}{2^k}$ which is a negligible function in k.

Let $k \in \mathbb{N}$.

$$\begin{aligned}
&\left|\Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; x, z), \sigma_1(e(g,g)^{f(x,z)})) = 0] - \Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; x, z), \sigma_1(T)) = 0]\right| \\
&\leq \sum_{G=(((G_0,\cdot),(G_1,\cdot),e),p,g)} \left| \begin{array}{l} \Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; x, z), \sigma_1(e(g,g)^{f(x,z)})) = 0 \mid G] \\ -\Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; x, z), \sigma_1(T)) = 0 \mid G] \end{array} \right| \cdot \Pr[G]
\end{aligned}$$

(using the law of total probability and the triangle inequality)

$$< \sum_{G=(((G_0,\cdot),(G_1,\cdot),e),p,g)} \frac{(q_k + 2s + 2)^2 \cdot d}{p - 1} \cdot \Pr[G]$$

(using Corollary 27)

$$\begin{aligned}
&\leq \sum_{G=(((G_0,\cdot),(G_1,\cdot),e),p,g)} \frac{(q_k + 2s + 2)^2 \cdot d}{2^k} \cdot \Pr[G] \\
&= \frac{(q_k + 2s + 2)^2 \cdot d}{2^k} \\
&= \text{negl}(k)
\end{aligned}$$

(where $G = (((G_0, \cdot), (G_1, \cdot), e), p, g)$ is the event that $\mathcal{G}(1^k)$ generates the corresponding group of order p and generator g) □

5.2.1 Generic Proof of the Decisional q-Parallel Bilinear Diffie-Hellman Exponent Assumption

Using the extended framework and Theorem 29 we can now easily prove that the decisional q-parallel BDHE assumption holds generically.

First, we formally state the problem and the assumption.

Problem 30 (cf. Section 2.4.1 [Wat11])**.** *Let $q \in \mathbb{N}$ be a natural number. The (computational)* q-parallel bilinear Diffie-Hellman exponent problem *is:*
Let $(((G_0, \cdot), (G_1, \cdot), e), p, g) \leftarrow \mathcal{G}(1^n)$ where $((G_0, \cdot), (G_1, \cdot), e)$ is a bilinear group of prime order p with generator $g \in G_0$.
Given $((G_0, \cdot), (G_1, \cdot), e), p$ and

$$\{g, g^s, g^{a^i}, g^{s \cdot b_j}, g^{a^i/b_j}, g^{a^l \cdot s \cdot b_k/b_j} \mid 1 \leq i \leq 2q,\ i \neq q+1,\ 1 \leq j, k, l \leq q,\ j \neq k\}$$

for $a, s \xleftarrow{R} \mathbb{F}_p, b_1, \ldots, b_q \xleftarrow{R} \mathbb{F}_p \setminus \{0\}$, compute

$$e(g, g)^{a^{q+1}s}$$

We also call this problem the (computational) *q-parallel BDHE problem.*
In [Wat11] there is a slight error in Appendix E where the definition of P does not match the original problem definition. Problem 30 is based on the description in Section 2.4.1 in [Wat11].

The assumption used by Waters in their construction [Wat11] is based on a decision variant of Problem 30.

Definition 31 (cf. Definition 2.1 [Wat11])**.** *In the situation of Problem 30 let*
$I_g(a, s, b_1, \ldots, b_q) = \{g, g^s, g^{a^i}, g^{s \cdot b_j}, g^{a^i/b_j}, g^{a^l \cdot s \cdot b_k/b_j} \mid 1 \leq i \leq 2q,\ i \neq q+1,\ 1 \leq j, k, l \leq q,\ j \neq k\}$.
The decisional q-parallel bilinear Diffie-Hellman exponent assumption *for $\mathcal{G}$ states:*

For all polynomial time algorithms $\mathcal{A}$ that output 0 or 1 there is a negligible function negl *such that for all $n \in \mathbb{N}$:*

$$\left|\Pr[\mathcal{A}(I_g(a, s, b_1, \ldots, b_q), e(g, g)^{a^{q+1}s}) = 0] - \Pr[\mathcal{A}(I_g(a, s, b_1, \ldots, b_q), T) = 0]\right| < \mathrm{negl}(n)$$

where the probabilities are over $(((G_0, \cdot), (G_1, \cdot), e), p, g) \leftarrow \mathcal{G}(1^n), a, s \xleftarrow{R} \mathbb{Z}_p, b_1, \ldots, b_q \xleftarrow{R}$

$\mathbb{Z}_p \setminus \{0\}, T \xleftarrow{R} G_1$ *and the random bits of* $\mathcal{A}$.

If Definition 31 holds then we say that no polynomial time algorithm (in n) has non-negligible advantage in solving the decisional q-parallel BDHE problem.

In order to be able to apply the original framework, Waters [Wat11] (Appendix E) suggests substituting the generator g with $g^{\prod_{j\in[1,q]} b_j}$. Roughly speaking, all of the terms in P are multiplied with $\prod_j B_j$ which results in proper polynomials. The original Boneh, Boyen, Goh framework can then be applied to the modified problem.
Specifically, we would consider (P', Q', f') corresponding to the problem with substituted generator $g^{\prod_{j\in[1,q]} b_j}$ (we shortly write $B = \prod_j B_j$):
P' is a tuple with components from

$$P'_{\text{set}} = \left\{ B, B\cdot S, B\cdot A^i, B\cdot S\cdot B_j, A^i\cdot B/B_j, A^l\cdot B_k\cdot B/B_j \;\middle|\; \begin{array}{l} 1\le i\le 2q,\ i\neq q+1, \\ 1\le j,k,l\le q,\ j\neq k \end{array} \right\}$$

and $Q' = (B^2, \ldots, B^2)$ and $f' = B^2\cdot A^{q+1}\cdot S$

However, their assumption (Definition 31) can be easily shown directly using our extension to the framework.

For this, we first show that for $(P, Q, f)_L$ corresponding to the q-parallel BDHE assumption, f is independent of (P, Q). This is greatly simplified by using Lemma 28.

Corollary 32. *Let $p \in \mathbb{N}$ be a prime number and let $q \in \mathbb{N}$.*
Let $P \in \mathbb{F}_p[A, S, B_1, \ldots, B_q][B_1^{-1}, \ldots, B_q^{-1}]^{q^3+q^2+2q+1}$ be a tuple with components from

$$P_{set} = \{1, S, A^i, S\cdot B_j, A^i\cdot B_j^{-1}, A^l\cdot B_k\cdot B_j^{-1} \mid 1\le i\le 2q,\ i\neq q+1,\ 1\le j,k,l\le q,\ j\neq k\}$$

and let $Q = (1, \ldots, 1) \in \mathbb{F}_p[A, S, B_1, \ldots, B_q][B_1^{-1}, \ldots, B_q^{-1}]^{q^3+q^2+2q+1}$

Then $f = A^{q+1}S$ is independent of (P, Q).

Proof. To apply Lemma 28 we only need to show that f is not the product of two terms in P_{set} (since $Q = (1, \ldots, 1)$).
Suppose $p, p' \in P_{\text{set}}$ with $f = p\cdot p'$.
Then either p or p' must contain the polynomial variable S. Without loss of generality, $p \in \{S, S\cdot B_j \mid 1\le j\le q\}$. Therefore, $p' = f/p \in \{A^{q+1}, A^{q+1}B_j^{-1} \mid 1\le j\le q\}$. But $\{A^{q+1}, A^{q+1}B_j^{-1} \mid 1\le j\le q\} \cap P_{\text{set}} = \emptyset$, therefore $p' \notin P_{\text{set}}$ which contradicts our assumption.
Lemma 28 implies that f is independent of (P, Q). □

Finally, we use Theorem 29 to show that the decisional q-parallel BDHE assumption (Definition 31) holds generically.

Corollary 33 (decisional q-parallel BDHE assumption)**.** *Let $q \in \mathbb{N}$ be a natural number and $\mathcal{G}$ an instance generator that generates bilinear groups of prime order p with $p > 2^n$ when invoked with $\mathcal{G}(1^n)$.*

No generic polynomial time algorithm has non-negligible advantage in solving the q-parallel BDHE problem with respect to $\mathcal{G}$.

Proof. Let $P \in \mathbb{Z}[A, S, B_1, \ldots, B_q][B_1^{-1}, \ldots, B_q^{-1}]^{q^3+q^2+2q+1}$ be a tuple with components from

$$P_{\text{set}} = \{1, S, A^i, S \cdot B_j, A^i \cdot B_j^{-1}, A^l \cdot B_k \cdot B_j^{-1} \mid 1 \leq i \leq 2q,\ i \neq q+1,\ 1 \leq j, k, l \leq q,\ j \neq k\}$$

and let $Q = (1, \ldots, 1) \in \mathbb{Z}[A, S, B_1, \ldots, B_q][B_1^{-1}, \ldots, B_q^{-1}]^{q^3+q^2+2q+1}$.

Corollary 32 implies that for all prime numbers $p \in \mathbb{N}$, f is independent of (P, Q) when the polynomials are projected into $\mathbb{F}_p$.
Therefore, Theorem 29 supplies the necessary bound:

For all generic polynomial time algorithms $\mathcal{A}$ (in n) that output 0 or 1 there is a negligible function negl such that for all $n \in \mathbb{N}$:

$$\left| \begin{array}{l} \Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; a, s, b_1, \ldots, b_q), \sigma_1(e(g,g)^{a^{q+1}s})) = 0] \\ -\Pr[\mathcal{A}(I_g(\sigma_0, \sigma_1; a, s, b_1, \ldots, b_q), \sigma_1(T)) = 0] \end{array} \right| < \text{negl}(n)$$

where the probabilities are over $(((G_0, \cdot), (G_1, \cdot), e), p, g) \leftarrow \mathcal{G}(1^n)$ and $\sigma_0 \xleftarrow{R} \{\sigma : G_0 \to S \mid \sigma \text{ injective}\}, \sigma_1 \xleftarrow{R} \{\sigma : G_1 \to S \mid \sigma \text{ injective}\}, a, s \xleftarrow{R} \mathbb{Z}_p, b_1, \ldots, b_q \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}, T \xleftarrow{R} G_1$ and the random bits of $\mathcal{A}$. □

5.2.2 Generic Proof of the Decisional Modified Bilinear Diffie-Hellman Assumption

Another application of our extended framework is the decisional modified bilinear Diffie-Hellman (MBDH) assumption [SW05].

Definition 34 (cf. Definition 3 [SW05])**.** *The* decisional modified bilinear Diffie-Hellman assumption *for an instance generator $\mathcal{G}$ states:*

For all polynomial time algorithms $\mathcal{A}$ that output 0 or 1 there is a negligible function

negl *such that for all* $n \in \mathbb{N}$*:*

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^c, e(g,g)^{a \cdot b/c}) = 0] - \Pr[\mathcal{A}(g, g^a, g^b, g^c, T) = 0]\right| < \text{negl}(n)$$

where the probabilities are over $(((G_0, \cdot), (G_1, \cdot), e), p, g) \leftarrow \mathcal{G}(1^n), a, b \xleftarrow{R} \mathbb{Z}_p, c \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}, T \xleftarrow{R} G_1$ *and the random bits of* $\mathcal{A}$*.*

If this definition holds we say that no polynomial time algorithm has non-negligible advantage in solving the decisional MBDH problem.

In the generic group model, the assumption follows directly from Theorem 29.

Corollary 35. *Let* $\mathcal{G}$ *be an instance generator that generates bilinear groups of prime order* p *with* $p > 2^n$ *when invoked with* $\mathcal{G}(1^n)$*.*

No generic polynomial time algorithm has non-negligible advantage in solving the decisional MBDH problem with respect to $\mathcal{G}$*.*

Proof. We set $P = (1, A, B, C), Q = (1, 1, 1, 1) \in (\mathbb{Z}[A, B, C][C^{-1}])^4$ and $f = (A \cdot B \cdot C^{-1}) \in \mathbb{Z}[A, B, C][C^{-1}]$.
Using Lemma 28 it is trivial to see that f is independent of (P, Q) when projected to $\mathbb{F}_p$ for any prime number p (since $f = A \cdot B \cdot C^{-1}$ cannot be written as a product of any two terms in P).
This allows us to apply Theorem 29 which directly implies the necessary statement:

For all generic polynomial time algorithms $\mathcal{A}$ (in n) that output 0 or 1 there is a negligible function negl such that for all $n \in \mathbb{N}$:

$$\left| \begin{array}{c} \Pr[\mathcal{A}(\sigma_0(g), \sigma_0(g^a), \sigma_0(g^b), \sigma_0(g^c), \sigma_1(e(g,g)^{a \cdot b/c})) = 0] \\ -\Pr[\mathcal{A}(\sigma_0(g), \sigma_0(g^a), \sigma_0(g^b), \sigma_0(g^c), \sigma_1(T)) = 0] \end{array} \right| < \text{negl}(n)$$

where the probabilities are over $(((G_0, \cdot), (G_1, \cdot), e), p, g) \leftarrow \mathcal{G}(1^n)$ where $((G_0, \cdot), (G_1, \cdot), e)$ is a bilinear group of prime order $p > 2^n$ with generator g and $\sigma_0 \xleftarrow{R} \{\sigma : G_0 \to S \mid \sigma \text{ injective}\}, \sigma_1 \xleftarrow{R} \{\sigma : G_1 \to S \mid \sigma \text{ injective}\}, a, b \xleftarrow{R} \mathbb{Z}_p, c \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}, T \xleftarrow{R} G_1$ and the random bits of $\mathcal{A}$. □

6 Conclusion

In this thesis, we discussed the generic group model as a formal way to ensure that algorithms cannot exploit group specific or encoding specific properties. A central result in the discussion was the fact that generic algorithms cannot distinguish isomorphic groups. This closely describes the limits of generic algorithms: They can only exploit properties that all isomorphic groups have in common.
The Boneh, Boyen, Goh framework [BBG05] allows bounding the advantage of generic algorithms for Diffie-Hellman related problems in bilinear groups. We provided a thorough proof for their framework.
However, while this framework covers many typical assumptions directly, it does not apply to Diffie-Hellman problems where some exponents may appear inverted. An example for this is the q-parallel bilinear Diffie-Hellman exponent assumption by Waters [Wat11]. We presented an extension to the original framework that can be applied to a wider range of typical problems in pairing-based cryptography.
Finally, we used the extended framework to show that the assumption by Waters holds generically.

As a result of these findings, we note that essentially, a decisional $(P, Q, f)_L$-Diffie-Hellman problem is hard for generic algorithms as soon as the element described by f cannot be computed trivially using the supplied elements from the input (i.e. f is independent of (P, Q)).
In other words: Diffie-Hellman exponent type problems (possibly in bilinear groups) are either trivially and visibly insecure or they are immediately generically secure.

There are two ways to view this:
On the one hand, it seems that generic algorithms are quite restricted in their possibilities and those restrictions are possibly far too severe to have any real-world implications. Formally, this is true: Proofs in the generic group model explicitly do not imply security for any concrete group. However, for many groups and problems, the best algorithms that are currently known are actually generic algorithms. This especially holds for suitable elliptic curve groups. In this sense, the model allows to draw some conclusions about real-world security.
On the other hand, this strong result allows researchers to postulate various Diffie-Hellman related assumptions and immediately have some basic certainty about the assumption's validity. This means that they can more confidently create cryptographic constructions that are not based on standard assumptions but on some (almost) arbitrary variation of Diffie-Hellman. Of course, those assumptions may still be proven insecure. But if an attack is found then an attacker must be using some property related to a specific group. It is therefore likely that an attack could be averted by simply

changing the group that the system is based on.

For future improvements upon our findings we note that Theorem 26 can be easily generalized for situations with a bilinear map $G_0 \times G_1 \to G_2$, where G_0 is not necessarily the same group as G_1, using the same idea that Boneh, Boyen, Goh propose (Definition A.4 [BBG05]). Our extension of the framework for Laurent polynomials is compatible with this generalization.

Bibliography

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology–EUROCRYPT 2005*, pages 440–456. Springer, 2005.

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

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

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology–EUROCRYPT 1997*, pages 256–266. Springer, 1997.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473. Springer, 2005.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography–PKC 2011*, pages 53–70. Springer, 2011.