The Semi-Generic Group Model and Applications to Pairing-Based Cryptography*

Tibor Jager[1] and Andy Rupp[2]

[1] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
tibor.jager@rub.de
[2] University of Trier, Germany
andy.rupp@rub.de

Abstract. In pairing-based cryptography the Generic Group Model (GGM) is used frequently to provide evidence towards newly introduced hardness assumptions. Unfortunately, the GGM does not reflect many known properties of bilinear group settings and thus hardness results in this model are of limited significance. This paper proposes a novel computational model for pairing-based cryptography, called the Semi-Generic Group Model (SGGM), that is closer to the standard model and allows to make more meaningful security guarantees. In fact, *the best algorithms currently known for solving pairing-based problems are semi-generic* in nature. We demonstrate the usefulness of our new model by applying it to study several important assumptions (BDDH, Co-DH). Furthermore, we develop master theorems facilitating an easy analysis of other (future) assumptions. These master theorems imply that (unless there are better algorithms than the semi-generic ones) great parts of the zoo of novel assumptions over bilinear groups are reducible to just *two* (more or less) standard assumptions over finite fields. Finally, we examine the appropriateness of the SGGM as a tool for analyzing the security of *practical* cryptosystems without random oracles by applying it to the BLS signature scheme.

Keywords. Restricted models of computation, generic groups, semi-generic group model, cryptographic assumptions, master theorems, provable security, pairing-based cryptography.

1 Introduction

Assuming that certain computational problems, mostly from algebra, number theory, and coding theory, are intractable builds the foundation of public-key cryptography. However, proving the validity of these assumptions in the standard model of computation seems to be impossible with currently available techniques.

Why do we believe in such hardness assumptions, though they are not provable in general? For classic number-theoretic problems, such as integer factorization (IF) or the discrete logarithm (DL) problem, this is certainly due to the absence of efficient algorithms in spite of intensive long-term research by many brilliant people. However,

* This is an extended abstract, see [20] for the full version. This research has been supported by the European Community (FP7/2007-2013) under grant agreement number ICT-2007-216646 - European Network of Excellence in Cryptology II (ECRYPT II).

besides such well-known assumptions, there frequently appear new assumptions building the basis for novel cryptosystems with original properties. What can be done to provide evidence for these assumptions apart from trying to find efficient algorithms over decades? Clearly, we should try to underpin the belief in novel assumptions by searching for reductions to a more mature assumption; but unfortunately finding such a reduction often fails.

An important approach to (nevertheless) gain immediate evidence towards hardness assumptions is to prove them with respect to a restricted but still meaningful class of algorithms. This is the motivation behind the invention of black-box models for algebraic structures like groups, fields, and rings, where algorithms are limited to perform only operations "commonly" available over these structures. Probably, the most famous of these models is the *generic group model (GGM)* introduced by Shoup in his seminal paper [34] from 1997, and refined by Maurer in [28]. In this model one considers algorithms – so-called *generic group algorithms* – that, given a group $\mathbb{G}$ as black-box, may only perform a set of basic operations on the elements of $\mathbb{G}$ such as applying the group law, inversion of group elements and equality testing. Since the group is treated as a black-box, the algorithms cannot exploit any special properties of a concrete group representation. As a consequence, such algorithms are generic in the sense that they can be applied to any concrete instantiation of a group (e.g., $\mathbb{Z}_p^*$ or $E(\mathbb{F}_p)$) in order so solve a problem. Natural examples of this class of algorithms are the Pohlig-Hellman [30] and Pollard's Rho [31] algorithm for computing discrete logarithms.

It should be noted that one has to take care when interpreting results in the GGM like intractability results as evidence in practice, since this model abstracts away from potentially many properties an algorithm might be able to exploit in the real world [15]. On the one hand, there exist cryptographic groups (such as certain elliptic curve groups) for which not many properties beyond the axioms of an algebraic group are known. Hence, modeling such groups as generic can be seen as a reasonable abstraction. On the other hand, there are groups, also used in cryptography, featuring many further properties, which clearly makes the generic model an inappropriate reflection for them. A prime example are multiplicative groups of finite fields or rings. These structures offer many well-understood properties beyond the group axioms, such as additional efficient algebraic operations (e.g., addition in the field or ring), and other properties of the group representation (e.g., the notion of prime integers and irreducible polynomials), that are simply ignored by the generic group model, but give rise to more efficient algorithms for certain problems (e.g., index calculus algorithms for computing discrete logarithms).

But should a minimal requirement on such an idealized model of computation not be that at least all currently known algorithms are captured? There exist some first approaches in the cryptographic literature to tackle this issue: The pseudo-free group model proposed by Hohenberger [19] and Rivest [32] does not treat a group as a black-box. Unfortunately, the definition of pseudo-freeness is very restrictive in the sense that a number of important groups (like all known-order groups) are immediately excluded and important problems, such as Diffie-Hellman-type problems, seem not to be covered. Other approaches due to Leander and Rupp [27] and Aggarwal and Maurer [1] take into account that the RSA group $\mathbb{Z}_n^*$ is embedded in the ring $\mathbb{Z}_n$. They use a generic *ring* model, where algorithms may perform both multiplication and addition

operations on $\mathbb{Z}_n$ to show that breaking RSA is equivalent to factoring. Unfortunately, recent work [21] shows that even computing the *Jacobi symbol* is equivalent to factoring in this model. So this approach has not led to a satisfying abstraction of $\mathbb{Z}_n^*$ yet.

Over the last decade a considerable number of innovative cryptosystems, such as identity-based encryption [7] or short digital signatures with strong security [9, 10], have been proposed over bilinear groups. A bilinear group setting consists of groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$, with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$, called a pairing. Along with these cryptosystems also many new assumptions have been introduced, e.g., Bilinear Diffie-Hellman (BDH) [23, 24], q-Strong Diffie-Hellman [4, 14, 22], Decision Linear Diffie-Hellman (DLIN) [6], Co-Diffie-Hellman (Co-DH) [9, 8], and many more. Unfortunately, for virtually all of them no reduction to a well-analyzed assumption like DL is known. In fact, finding such reductions seems to be a difficult task, since the algebraic settings underlying classic problems (e.g., a single cyclic group for DL) significantly differ from bilinear settings. Hence, given an instance of a classic problem, it appears to be hard to transform this instance to one of the bilinear problem in order to leverage an algorithm for the latter.

Consequently, the only way to provide some immediate evidence for such novel assumptions consists in proofs in restricted models of computation. So far, the only such model for bilinear settings is a straightforward extension of the generic group model, where all three groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$ are modeled as generic groups [33, 11, 25]. In all known instances of bilinear settings the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are elliptic curve groups, thus modeling these groups as generic may be considered as a reasonable abstraction. However, in contrast to that, the group $\mathbb{G}_3$ is usually a subgroup of the multiplicative group of a finite field. So there definitely exist non-generic algorithms for cryptographic problems like BDH, Co-DH, etc. featuring a running time which is *at most* sub-exponential: these sub-exponential algorithms map the inputs over $\mathbb{G}_1$ and $\mathbb{G}_2$ (given as part of a problem instance) to $\mathbb{G}_3$ using the bilinear mapping (MOV reduction [29]) and determine the discrete logarithms of these elements over $\mathbb{G}_3$ using index calculus. Knowledge of these discrete logarithms allows to compute the solution to the problem instance using a few exponentiations. Note that there might be even more efficient algorithms especially for potentially easier problems like decisional or gap problems. Hence, modeling bilinear settings in this way is clearly inappropriate.

OUR CONTRIBUTION We propose the *Semi-Generic Group Model (SGGM)* which leverages this observation as follows: The elliptic curve groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are modeled as generic groups, while $\mathbb{G}_3$ is given in the standard model, i.e., algorithms may perform *any* computation over $\mathbb{G}_3$ that is possible in the subgroup of a finite field. The SGGM is thus closer to the standard model than the GGM and can provide stronger evidence towards hardness assumptions in pairing-based cryptography. In fact, to the best of our knowledge all algorithms currently known for solving pairing-based problems are semi-generic in nature. In particular, the sub-exponential algorithms applying a MOV reduction described above are covered by the SGGM.

We analyzed some of the most important computational and decisional assumptions of pairing-based cryptography in our new model. In this extended abstract we restrict to consider Co-DH and decisional BDH. The full version of the paper [20] covers additional problems, including q-strong DH and DLIN. We are able to reduce the considered

assumptions (with respect to semi-generic algorithms) to fairly standard assumptions over finite fields like Square DH and a slight variation of DL. That means, the bilinear assumptions are at least as hard as certain more standard assumption over $\mathbb{G}_3$ provided that there are no non-semi-generic algorithms. Furthermore, we developed master theorems ensuring the hardness of broad classes of computational and decisional problems in the SGGM. Studying such generalizations is not only important in order to structure and facilitate the analysis of the rapidly growing set of cryptographic assumptions as motivated in [3], but improves our understanding of the properties which need to be satisfied by a problem to be intractable. Results like [12, 33, 11] are in this vein. Boyen [11] (see also [5]) developed master theorems for the hardness of some general classes of decisional problems in the generic group model for bilinear settings. Rupp et al. [33] provide hardness conditions for even broader classes of computational problems and algebraic settings, but still in the GGM. Bresson et al. [12] study a general class of decisional assumptions over a single group in the standard model and show that this class can be reduced to DDH (under certain restrictions). In the scope of the proof of our master theorem for decisional problems we enhance Bresson et al.'s results for the standard model and apply them to the SGGM.

The security of public-key cryptosystems, especially of *practical* cryptosystems, can often only be proven in an idealized model, such as the random oracle model (ROM) [2]. An issue with the ROM is that it idealizes a hash function in a way such that it has *all* properties of a "perfect" hash function (collision resistance, (second) preimage resistance, random output, ...) at the same time. When the cryptosystem (and thus the random oracle) is implemented in practice, one has to choose an adequate hash function instantiating the random oracle. An important question is whether providing *all* properties of the random oracle at the same time is really necessary to provide security.

We examine the useability of the SGGM as a tool complementing the ROM. We are able to prove the security of the Boneh-Lynn-Shacham (BLS) short signature scheme [9, 10] against semi-generic adversaries without random oracles, however, requiring non-standard properties for the employed hash function. It is left as an interesting open problem to study whether these requirements can actually be satisfied by a reasonably efficient practical hash function.

2 The Semi-Generic Group Model

Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$ be groups of prime order p and $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ be corresponding generators. For the sake of simplicity of the subsequent formalizations we use multiplicative notation for all groups.

Definition 1. *A* pairing *is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ with the following properties:*

1. Bilinearity*: $\forall (a, b) \in \mathbb{G}_1 \times \mathbb{G}_2$ and $x_1, x_2 \in \mathbb{Z}_p$ holds that $e(a^{x_1}, b^{x_2}) = e(a, b)^{x_1 x_2}$.*
2. Non-degeneracy*: $g_3 := e(g_1, g_2)$ is a generator of $\mathbb{G}_3$, i.e., $g_3 \neq 1$.*
3. *e is efficiently computable.*

Following [17], we distinguish three different types of bilinear group settings:

- *Type 1: $\mathbb{G}_1 = \mathbb{G}_2$. We will call this the setting with* symmetric *bilinear map.*

– *Type 2:* $\mathbb{G}_1 \neq \mathbb{G}_2$*, there is an* efficiently computable *isomorphism* $\psi : \mathbb{G}_1 \to \mathbb{G}_2$.
– *Type 3:* $\mathbb{G}_1 \neq \mathbb{G}_2$*, there is no efficiently computable isomorphism* $\psi : \mathbb{G}_1 \to \mathbb{G}_2$.

FORMAL DEFINITION OF THE SGGM We base our formal description of the SGGM for bilinear settings on the generic group model introduced by Maurer [28], though our proofs can be adapted to Shoup's GGM [34] as well. The main difference between Maurer's and Shoup's formalization is that in the first model group elements are encoded deterministically whereas in the second model encodings are random.

An algorithm $\mathcal{A}$ in the SGGM interacts with a *semi-generic group oracle* $\mathcal{O}$, which computes the group operations and evaluates the pairing and isomorphism on behalf of $\mathcal{A}$. $\mathcal{O}$ receives as input two vectors of group elements (the problem instance)

$$I_1 = (a_{1,1}, \ldots, a_{1,k_1}) \in \mathbb{G}_1^{k_1} \quad \text{and} \quad I_2 = (a_{2,1}, \ldots, a_{2,k_2}) \in \mathbb{G}_2^{k_2} .$$

It maintains two lists $\mathcal{E}_1 \subseteq \mathbb{G}_1$ and $\mathcal{E}_2 \subseteq \mathbb{G}_2$, with $\mathcal{E}_{i,j}$ denoting the j-th entry of list $\mathcal{E}_i$, which are initialized such that $\mathcal{E}_{i,j} := a_{i,j}$ for all possible (i, j). We denote with $[a]_i$ the smallest index j (also called encoding) such that $\mathcal{E}_{i,j} = a$. Index $[a]_i$ is undefined, if $a \notin \mathcal{E}_i$. We may always assume that semi-generic algorithms only provide defined indices as input to the oracle. During initialization of the lists $\mathcal{E}_1$ and $\mathcal{E}_2$, the corresponding indices pointing to the contained elements are sent to the algorithm.

The oracle implements the following *public procedures*, which may be called by $\mathcal{A}$:

– `GroupOp`$([a]_i, [b]_i, i)$: This procedure takes as input two indices $[a]_i, [b]_i$ and a list index i. It determines the group elements $a, b \in \mathbb{G}_i$ by list lookup, computes $c = a \cdot b \in \mathbb{G}_i$, appends c to $\mathcal{E}_i$, and returns $[c]_i$.
– `BilinearMap`$([a]_1, [b]_2)$: This procedure takes as input two indices $[a]_1, [b]_2$. It determines the corresponding group elements $a \in \mathbb{G}_1, b \in \mathbb{G}_2$ by list lookup and returns $e(a, b)$ in the *standard* representation of $\mathbb{G}_3$ (i.e., as finite field element).

When considering Type 2 settings the algorithm may also query to apply the isomorphism ψ to an element of $\mathbb{G}_1$:

– `Isomorphism`$([a]_1)$: This procedure takes as input an index $[a]_1$, determines the element $a \in \mathbb{G}_1$, computes $b = \psi(a)$, appends b to $\mathcal{E}_2$ and returns $[b]_2$.

Note that a random group element can be efficiently sampled by a semi-generic algorithm by using `GroupOp`$(\cdot)$ to raise the generator (which is always part of a problem instance) to some $r \xleftarrow{\$} \mathbb{Z}_p$.

2.1 Essential Ingredients for Proofs in the SGGM

This section describes a few general observations that will turn out to be the essential ingredients for proofs in the semi-generic model.

Observation 1: Components inside oracle are exchangeable. Semi-generic algorithms due to its nature are "blind" with respect to the internal details of the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ as well as the pairing e and the isomorphism ψ. These components are hidden within a black-box. Hence, we can plug-in "something else" for these components as

long as these replacements behave like cyclic groups with a bilinear map and an isomorphism. We will utilize this observation in a novel way to map inputs given over $\mathbb{G}_3$ back to $\mathbb{G}_1$ and $\mathbb{G}_2$ by setting $\mathbb{G}_1 := \mathbb{G}_2 := \mathbb{G}_3$ internally and simulating a virtual bilinear map $e : \mathbb{G}_3 \times \mathbb{G}_3 \to \mathbb{G}_3$ and isomorphism $\psi : \mathbb{G}_3 \to \mathbb{G}_3$.

Observation 2: Computed elements over $\mathbb{G}_1$ and $\mathbb{G}_2$ are linear polynomials in initial inputs. Let $I_1 \in \mathbb{G}_1^m$ and $I_2 \in \mathbb{G}_2^n$ be inputs given to the semi-generic oracle (as part of a problem instance). We have $I_2 = I_1$ in the case of a Type 1 setting. In the following, we always assume that at least the generators g_1 and g_2 are given (as the first components of these input tuples). So we can write $I_1 = (g_1, g_1^{x_2}, \ldots, g_1^{x_m})$ and $I_2 = (g_2, g_2^{y_2}, \ldots, g_2^{y_n})$ for some *unknown* $x_j, y_k \in \mathbb{Z}_p$ (no assumptions about their distribution are made here) and $\psi(g_1) = g_2$ in the case of a Type 2 setting. Then we define the tuple $I_1' := I_1$ and the tuple $I_2' := I_2$ in the case of a Type 1 and Type 3 setting or $I_2' := (g_2, g_2^{x_2}, \ldots, g_2^{x_m}, g_2^{y_2}, \ldots, g_2^{y_n})$ for a Type 2 setting. These tuples are called the *initial inputs* to semi-generic algorithms. Using this notation, we can describe the following observation: Over $\mathbb{G}_1$ and $\mathbb{G}_2$ a semi-generic algorithm can only perform the group law on the initial inputs. Thus, any element $a \in \mathbb{G}_i$ ($i \in \{1, 2\}$) computed by a semi-generic algorithm is a product of the elements in I_i'. Hence we can represent such an element as $a = g_i^{P(x_2,\ldots,x_m,y_2,\ldots,y_n)}$ for some linear multivariate polynomial $P = \alpha_1 + \sum_{j=2}^{m} \alpha_j X_j + \sum_{j=2}^{n} \beta_j Y_j$, where the β_j are zero in the case $i = 1$ or if we consider a Type 1 setting. It is important to observe that all coefficients of this polynomial are *known* to the oracle.

Observation 3: Pairing is simulatable knowing images of initial inputs. Let $a \in \mathbb{G}_1$ and $b \in \mathbb{G}_2$ be two elements computed by a semi-generic algorithm. Then by using the above observation and setting $x_1 := 1$ it is easy to see that

$$\begin{aligned} e(a,b) &= e(g_1^{\sum_{i=1}^{m} \alpha_i x_i}, g_2^{\sum_{j=1}^{m} \alpha_j' x_j + \sum_{k=2}^{n} \beta_k' y_k}) \\ &= \prod_{i=1}^{m} \prod_{j=1}^{m} e(g_1^{x_i}, g_2^{x_j})^{\alpha_i \alpha_j'} \cdot \prod_{i=1}^{m} \prod_{k=2}^{n} e(g_1^{x_i}, g_2^{y_k})^{\alpha_i \beta_k'} \end{aligned}$$

From this equation it follows that by *knowing the images of the initial inputs* under the pairing, one can *compute the output of e on arbitrary inputs* provided by a semi-generic algorithm without actually evaluating the pairing explicitly. In other words, an oracle equipped with a table containing $e(a, b)$ for all combinations of a in I_1' and b in I_2' would be able to handle all `BilinearMap` queries.

3 Analysis of Selected Problems in the Semi-Generic Model

In this section we exemplarily analyze the hardness of the computational Co-DH and the decisional BDH problem. Certainly, the list of problems we are considering here is by no means complete. Our main purpose is to give concrete analyses of some important problems of bilinear cryptography, thereby illustrating the basic ideas and techniques underlying proofs in this model, before dealing with the more intricate case of general classes of problems in Section 4.

3.1 Reducing 2-DL to Co-DH

The Co-DH problem has been used in [9, 8] for the construction of short and aggregate signatures over bilinear groups. Over a Type 2 setting it can be defined as follows: Given $(g_1, g_1^{x_0}, g_2, g_2^{x_1}, g_3)$, where $(x_0, x_1) \xleftarrow{\$} \mathbb{Z}_p^2$ are secret random choices, output $g_2^{x_0 x_1}$.

It is easy to see that in order to prove something about the hardness of Co-DH, we definitely need to make the assumption that the discrete logarithm problem over $\mathbb{G}_3$ is intractable. But is this enough? Our answer is "not quite": We are going to relate the hardness of Co-DH to the 2-DL problem over $\mathbb{G}_3$, a slightly easier variant of DL. The q-DL problem can be defined as follows: Given $(g_3, g_3^{x^1}, \ldots, g_3^{x^q})$, where $x \xleftarrow{\$} \mathbb{Z}_p$ is a secret random value, output x. The additional input $g_3^{x^2}$ (in comparison to standard DL) is needed in order to be able to simulate the pairing when running the Co-DH algorithm.

Theorem 1. *Suppose there exists a semi-generic group algorithm $\mathcal{A}$ solving* Co-DH *over a Type 2 bilinear group setting in time t with success probability ϵ. Then there exists an algorithm $\mathcal{B}$ solving the* 2-DL *problem over $\mathbb{G}_3$ in time $t' \approx t$ with success probability $\epsilon' \geq \frac{1}{2}\epsilon$.*

Proof. Given an instance of the 2-DL problem, $\mathcal{B}$ sets up an instance of the Co-DH problem in the semi-generic model in a way that it can leverage a solution to Co-DH computed by $\mathcal{A}$ to solve the 2-DL instance. In particular, $\mathcal{B}$ will play the role of the semi-generic oracle. We exploit Observation 1 from Section 2.1 to setup such an useful instance: Since $\mathcal{A}$ is "blind" with respect to the internal details of $\mathbb{G}_1$, $\mathbb{G}_2$, e, and ψ, we set $\mathbb{G}_1 := \mathbb{G}_2 := \mathbb{G}_3$ and try to simulate a virtual bilinear map $e : \mathbb{G}_3 \times \mathbb{G}_3 \to \mathbb{G}_3$.

We are now ready to describe our reduction algorithm $\mathcal{B}$. $\mathcal{B}$ takes as input an instance $a_0 := g_3, a_1 := g_3^x, a_3 := g_3^{x^2}$ of the 2-DL problem over $\mathbb{G}_3$. Then it chooses $i^* \xleftarrow{\$} \{0, 1\}$, $x_{1-i^*} \xleftarrow{\$} \mathbb{Z}_p$ and sets $a_2 := g_3^{x_{1-i^*}}$. The wanted discrete logarithm x is now embedded as the implicit secret choice x_{i^*} in an instance of the Co-DH problem. More precisely, $\mathcal{B}$ sets up a problem instance and simulates the oracle $\mathcal{O}$ as follows:

- The lists $\mathcal{E}_1$ and $\mathcal{E}_2$ are initialized with $g_3, g_3^{x_0}$ and $g_3, g_3^{x_1}$, respectively, where $g_3^{x_{i^*}}$ is set to be a_1. The indices $[g_3]_1, [g_3^{x_0}]_1, [g_3]_2, [g_3^{x_1}]_2$, and g_3 are sent out to $\mathcal{A}$.
- `GroupOp` can be simulated since $\mathcal{B}$ knows how to perform the group law over $\mathbb{G}_3$.
- `Isomorphism`$([a]_1)$ can be simulated by looking up a in $\mathcal{E}_1$, appending it to $\mathcal{E}_2$, and then determining the index $[a]_2$.
- Using Observation 3 from Section 2.1, we can easily see that `BilinearMap` can be simulated: Let $[b]_1, [c]_2$ be the two indices given as input to the procedure by $\mathcal{A}$. Then we can write

$$e(b, c) = e(g_3^{\sum_{j=-1}^{0} z_j x_j}, g_3^{\sum_{k=-1}^{1} z'_k x_k}) = \prod_{j=-1}^{0} \prod_{k=-1}^{1} (g_3^{x_j x_k})^{z_j z'_k}$$

  where $x_{-1} := 1$ and z_j and z'_k are known to $\mathcal{B}$. Since $\mathcal{B}$ is given $a_0, \ldots, a_3$ and knows i^*, x_{1-i^*}, it can compute the required elements $g_3, g_3^{x_0}, g_3^{x_1}, g_3^{x_0^2}, g_3^{x_0 x_1}$ to simulate the pairing: $g_3 = a_0$, $g_3^{x_{i^*}} = a_1$, $g_3^{x_{1-i^*}} = a_2$, $g_3^{x_0^2} = a_3$ if $i^* = 0$ and $g_3^{x_0^2} = a_2^{x_0}$ else, $g_3^{x_0 x_1} = a_1^{x_{1-i^*}}$.

Given some instance of Co-DH, algorithm $\mathcal{A}$ eventually outputs some valid index $[c]_2$. The corresponding element $c \in \mathbb{G}_2$ can be written as $c = g_2^{P(x_0,x_1)}$ for some known polynomial $P = z_0 + z_1X_0 + z_2X_1 \in \mathbb{Z}_p[X_0, X_1]$ (Observation 2, Section 2.1). So alternatively we can say that $\mathcal{A}$ wins if $(P - X_0X_1)(x_0, x_1) \equiv 0 \bmod p$. This success event can be split up into the following disjoint events:

- Event $\mathcal{S}_1$: The univariate polynomial $(P - X_0X_1)(x_0)$, i.e., the polynomial $P - X_0X_1$ where we only evaluate the variable X_0 with x_0, is zero modulo p. Let the probability of this event be denoted by α_1.
- Event $\mathcal{S}_2$: The univariate polynomial $(P - X_0X_1)(x_0)$ is not zero modulo p but $(P - X_0X_1)(x_0, x_1)$ is. Let the probability of this event be denoted by α_2.

Clearly, we have $\epsilon = \alpha_1 + \alpha_2$.

Let us consider the events $\mathcal{S}_1$ and $\mathcal{S}_2$ when $\mathcal{B}$ runs $\mathcal{A}$ for certain choices of i^*. Note that $\mathcal{B}$ knows the coefficients of P since it responded to $\mathcal{A}$'s queries. With probability $\frac{1}{2}\alpha_1$, we have $i^* = 0$ and $\mathcal{S}_1$. This means $z_0 + z_1x + z_2X_1 - xX_1 \equiv 0$. But in this case z_2 needs to be equal to x. So $\mathcal{B}$ wins by simply returning the known coefficient z_2. Furthermore, with probability $\frac{1}{2}\alpha_2$, we have $i^* = 1$ and $\mathcal{S}_2$. Hence, the wanted DL is the root of the uni-variate non-zero polynomial $z_0 + z_1x_0 + z_2X_1 - x_0X_1$ known to $\mathcal{B}$. It can thus be determined as $x \equiv (z_0 + z_1x_0)(x_0 - z_2)^{-1} \bmod p$. It is easy to verify that the inverse $(x_0 - z_2)^{-1}$ always exists.

To summarize, if i^* happens to be zero, $\mathcal{B}$ outputs z_2, otherwise it outputs $(z_0 + z_1x_0)(x_0 - z_2)^{-1}$. In this way, its success probability is at least $\frac{1}{2}\alpha_1 + \frac{1}{2}\alpha_2 = \frac{1}{2}\epsilon$. □

3.2 Reducing SqDDH to BDDH

The bilinear decisional Diffie-Hellman problem (BDDH) is certainly among the most well-known problems over bilinear groups. It has originally been introduced in a seminal paper by Joux [23] and, e.g., further been used by Boneh and Franklin [7] to construct an identity based encryption scheme. Let us consider BDDH over a Type 1 setting where it can be defined as follows: Given $(g_1, g_1^{x_1}, g_1^{x_2}, g_1^{x_3}, g_3^{r_b})$, where $(x_1, x_2, x_3) \xleftarrow{\$} \mathbb{Z}_p^3$, $b \xleftarrow{\$} \{0, 1\}$, $r_1 = x_1x_2x_3$, and $r_0 \xleftarrow{\$} \mathbb{Z}_p$ are secret choices, output b.

We relate the hardness of BDDH with respect to semi-generic algorithms to the hardness of the well-known decisional Diffie-Hellman (DDH) problem and the square decisional Diffie-Hellman (SqDDH) problem over $\mathbb{G}_3$. SqDDH is a potentially easier variant of DDH: Given $(g_3, g_3^x, g_3^{r_b})$, where $x \xleftarrow{\$} \mathbb{Z}_p$, $b \xleftarrow{\$} \{0, 1\}$, $r_1 = x^2$, and $r_0 \xleftarrow{\$} \mathbb{Z}_p$ are secret choices, output b. Our result is formalized in Theorem 2. It is worth mentioning that in contrast to computational problems (like Co-DH) for decisional problems usually multiple reduction steps are required. In the proof we apply the idea of DDH-steps [12] to the bilinear setting and introduce the new concept of SqDDH-steps. Since the DDH assumption reduces to the SqDDH assumption [38] the hardness of BDDH can be formulated with respect to SqDDH only (Corollary 1).

Theorem 2. *Suppose there exists a semi-generic group algorithm $\mathcal{A}$ solving* BDDH *over a Type 1 setting in time t with advantage ϵ. Then there exists an algorithm $\mathcal{B}_{\mathsf{SqDDH}}$ solving* SqDDH *over $\mathbb{G}_3$ in time $t_{\mathsf{SqDDH}} \approx t$ with advantage $\epsilon_{\mathsf{SqDDH}}$ and an algorithm*

Table 1. Transforming a semi-generic oracle for real BDDH into one for random BDDH using SqDDH and DDH steps.

	Game									
	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}
$g_3^{r_b}$	$g_3^{x_1x_2x_3}$	$g_3^{x_1x_2x_3}$	$g_3^{x_1x_2x_3}$	$g_3^{x_1x_2x_3}$	$g_3^{\mathbf{x_7}x_3}$	$g_3^{\mathbf{x_8}}$	$g_3^{x_8}$	$g_3^{x_8}$	$g_3^{x_8}$	$g_3^{x_8}$
$e(g_1, g_1)$	g_3	g_3	g_3	g_3	g_3	g_3	g_3	g_3	g_3	g_3
$e(g_1, g_1^{x_1})$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_1}$
$e(g_1, g_1^{x_2})$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$	$g_3^{x_2}$
$e(g_1, g_1^{x_3})$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$	$g_3^{x_3}$
$e(g_1^{x_1}, g_1^{x_1})$	$g_3^{x_1^2}$	$g_3^{\mathbf{x_4}}$	$g_3^{x_4}$	$g_3^{x_4}$	$g_3^{x_4}$	$g_3^{x_4}$	$g_3^{x_4}$	$g_3^{x_4}$	$g_3^{x_4}$	$g_3^{\mathbf{x_1^2}}$
$e(g_1^{x_2}, g_1^{x_2})$	$g_3^{x_2^2}$	$g_3^{x_2^2}$	$g_3^{\mathbf{x_5}}$	$g_3^{x_5}$	$g_3^{x_5}$	$g_3^{x_5}$	$g_3^{x_5}$	$g_3^{x_5}$	$g_3^{\mathbf{x_2^2}}$	$g_3^{x_2^2}$
$e(g_1^{x_3}, g_1^{x_3})$	$g_3^{x_3^2}$	$g_3^{x_3^2}$	$g_3^{x_3^2}$	$g_3^{\mathbf{x_6}}$	$g_3^{x_6}$	$g_3^{x_6}$	$g_3^{x_6}$	$g_3^{\mathbf{x_3^2}}$	$g_3^{x_3^2}$	$g_3^{x_3^2}$
$e(g_1^{x_1}, g_1^{x_2})$	$g_3^{x_1x_2}$	$g_3^{x_1x_2}$	$g_3^{x_1x_2}$	$g_3^{x_1x_2}$	$g_3^{\mathbf{x_7}}$	$g_3^{x_7}$	$g_3^{\mathbf{x_1x_2}}$	$g_3^{x_1x_2}$	$g_3^{x_1x_2}$	$g_3^{x_1x_2}$
$e(g_1^{x_1}, g_1^{x_3})$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$	$g_3^{x_1x_3}$
$e(g_1^{x_2}, g_1^{x_3})$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$	$g_3^{x_2x_3}$
Justification		SqDDH	SqDDH	SqDDH	DDH	DDH	DDH	SqDDH	SqDDH	SqDDH

$\mathcal{B}_{\mathsf{DDH}}$ *solving* DDH *over* $\mathbb{G}_3$ *in time* $t_{\mathsf{DDH}} \approx t$ *with advantage* ϵ_{DDH} *such that* $\epsilon \leq 3\epsilon_{\mathsf{DDH}} + 6\epsilon_{\mathsf{SqDDH}}$.

Corollary 1. *If* SqDDH *is* (ϵ, t)*-hard over* $\mathbb{G}_3$*, then* BDDH *is* $(9\epsilon, t)$*-hard for semi-generic algorithms.*

Proof (Theorem 2). In the following we show that a for a semi-generic algorithm a "real" BDDH tuple $(g_1, g_1^{x_1}, g_1^{x_2}, g_1^{x_3}, g_3^{r_1} = g_3^{x_1x_2x_3})$ is computationally indistinguishable from a "random" tuple $(g_1, g_1^{x_1}, g_1^{x_2}, g_1^{x_3}, g_3^{r_0})$, unless SqDDH or DDH are easy over $\mathbb{G}_3$. We do this by considering a series of games played between a semi-generic algorithm $\mathcal{A}$ and an oracle $\mathcal{O}$. We start with $\mathcal{A}$ given oracle access to a real BDDH tuple. We then gradually transform this tuple as well as the output of the oracle until we end up with a random tuple. One can show that if $\mathcal{A}$ can distinguish two consecutive games G_{i-1} and G_i then it can be used to build an algorithm solving SqDDH or DDH.

The games are described by Table 3.2. Each of the columns labeled with G_i specifies the (direct) input over $\mathbb{G}_3$ (see Row 1) or the output of `BilinearMap` in game G_i for all possible inputs over $\mathbb{G}_1$. Bold-printed parts of a value highlight the actual changes in comparison to the previous game. The entry in the last row of a column G_i indicates which assumption (SqDDH or DDH) justifies the indistinguishability of the Games G_{i-1} and G_i. If a new x_j $(j > 3)$ appears in a column, this means that this value has been added to the corresponding game and the oracle chooses x_j uniformly from $\mathbb{Z}_p$.

As one can see from the table, by means of the Games G_2 to G_4 we remove all squares x_i^2 $(1 \leq i \leq 3)$ from the output of the pairing oracle. We do this simply by replacing each square with a new value x_j $(4 \leq j \leq 6)$. These transformations are called (bilinear) SqDDH steps and are prerequisites for the subsequent DDH steps performed in Games G_5 to G_6. During these DDH steps we selectively remove all

products $x_i x_j$ that involve variables being part of the challenge. Again, this is done by replacing the products by fresh uniformly chosen values x_j ($j \in \{7, 8\}$). In Game G_6 the challenge $g_3^{r_b} = g_3^{x_8}$ is finally independent of the input since x_8 does not appear anywhere else. After that, in Games G_7 to G_{10} we reverse the changes we did to the input and `BilinearMap` during G_2 to G_6 in reverse order. More, precisely in G_{6+j} we reverse the changes we did in G_{6-j} for $1 \leq j \leq 4$. Finally, in G_{10} we have reversed all changes (except for the one in G_6). This last game corresponds to the situation where $\mathcal{A}$ is given oracle access to a random BDDH tuple. If all intermediate games have been computationally indistinguishable (under the SqDDH and DDH assumption) then certainly also a real BDDH tuple is computationally indistinguishable from a random tuple, with respect to semi-generic algorithms.

For the sake of clarity, let us consider the transition from G_1 to G_2 (SqDDH Step) and G_4 to G_5 (DDH Step) in some more detail and quantify the involved reductions. The oracle $\mathcal{O}_{G_1}$ in Game G_1 corresponds to the original semi-generic oracle for BDDH providing access to a real BDDH tuple. The oracle in $\mathcal{O}_{G_2}$ in G_2 is equal to $\mathcal{O}_{G_1}$ except for the following changes: $\mathcal{O}_{G_2}$ additionally chooses $x_4 \xleftarrow{\$} \mathbb{Z}_p$ and uses a slightly modified table for computing pairing outputs as specified in Table 3.2. Let us assume $\mathcal{A}$ distinguishes the two games in time t with advantage

$$\epsilon_1 = \mathsf{Adv}_{\mathcal{A}}^{G_1,G_2} = \left|\Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_{G_1}}] - \Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_{G_2}}]\right| .$$

Then from $\mathcal{A}$ we can build an algorithm $\mathcal{B}$ for SqDDH. Again, we make use of the observation that semi-generic algorithms are blind with respect to $\mathbb{G}_1$ and e and set $\mathbb{G}_1 := \mathbb{G}_3$ and $e : \mathbb{G}_3 \times \mathbb{G}_3 \to \mathbb{G}_3$. Now let an instance

$$g_3, g_3^{x_1}, g_3^{r'_{b'}} = \begin{cases} g_3^{x_1^2}, & b' = 1 \\ g_3^{x_4}, & b' = 0 \end{cases}$$

of the SqDDH problem over $\mathbb{G}_3$ be given. $\mathcal{B}$ chooses $x_2, x_3 \xleftarrow{\$} \mathbb{Z}_p$. Then it simulates $\mathcal{O}_{G_1}$ and $\mathcal{O}_{G_2}$ as follows (we indicate below how group elements are computed though x_1, x_1^2, x_4, and b' are unknown to $\mathcal{B}$):

- The list $\mathcal{E}_1$ is initialized with $g_3, g_3^{x_1}, g_3^{x_2}, g_3^{x_3}$. Over $\mathbb{G}_3$ $\mathcal{A}$ is given $g_3, (g_3^{x_1})^{x_2 x_3}$.
- For simulating `BilinearMap`, we use the fact that we only need to know the pairing output for all possible initial inputs. These elements can be computed as described by the following table:

a	g_3	g_3	g_3	g_3	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_2}$
b	g_3	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_3}$
$e(a,b)$	g_3	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{r'_{b'}}$	$g_3^{x_2^2}$	$g_3^{x_3^2}$	$(g_3^{x_1})^{x_2}$	$(g_3^{x_1})^{x_3}$	$g_3^{x_2 x_3}$

It is easy to see that if $b' = 1$, algorithm $\mathcal{B}$ exactly simulates $\mathcal{O}_{G_1}$ and $\mathcal{O}_{G_2}$ otherwise. Thus, by simply forwarding the output of $\mathcal{A}$, $\mathcal{B}$ solves the SqDDH problem instance with the same advantage ϵ_1.

Let us now consider the transition from G_4 to G_5. The oracle $\mathcal{O}_{G_5}$ in G_5 coincides with $\mathcal{O}_{G_4}$ except for the following changes: $\mathcal{O}_{G_5}$ additionally chooses $x_7 \xleftarrow{\$} \mathbb{Z}_p$ and

uses a modified table for computing pairing outputs as specified in Table 3.2. Assume $\mathcal{A}$ distinguishes the two games in time t with advantage $\epsilon_4 = \mathsf{Adv}_{\mathcal{A}}^{G_4,G_5}$. Then we can use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ for DDH. Given an instance

$$g_3, g_3^{x_1}, g_3^{x_2}, g_3^{r'_{b'}} = \begin{cases} g_3^{x_1 x_2}, & b' = 1 \\ g_3^{x_7}, & b' = 0 \end{cases}$$

of DDH over $\mathbb{G}_3$, $\mathcal{B}$ chooses $x_3, x_4, x_5, x_6 \xleftarrow{\$} \mathbb{Z}_p$ and simulates $\mathcal{O}_{G_5}$ and $\mathcal{O}_{G_6}$:

- The list $\mathcal{E}_1$ is initialized with $g_3, g_3^{x_1}, g_3^{x_2}, g_3^{x_3}$. Over $\mathbb{G}_3$ $\mathcal{A}$ is given $g_3, (g_3^{r'_{b'}})^{x_3}$.
- For simulating `BilinearMap` we use the following table of pairing outputs:

a	g_3	g_3	g_3	g_3	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_1}$	$g_3^{x_1}$	$g_3^{x_2}$
b	g_3	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_3}$
$e(a,b)$	g_3	$g_3^{x_1}$	$g_3^{x_2}$	$g_3^{x_3}$	$g_3^{x_4}$	$g_3^{x_5}$	$g_3^{x_6}$	$g_3^{r'_{b'}}$	$(g_3^{x_1})^{x_3}$	$(g_3^{x_2})^{x_3}$

If $b' = 1$, $\mathcal{B}$ behaves like $\mathcal{O}_{G_4}$ whereas it behaves like $\mathcal{O}_{G_5}$ if $b' = 0$. By simply forwarding the output of $\mathcal{A}$, $\mathcal{B}$ solves the DDH problem instance with advantage ϵ_4.

The bound on ϵ follows now from $\epsilon \leq \sum_{i=1}^{9} \epsilon_i$, where $\epsilon_i = \mathsf{Adv}_{\mathcal{A}}^{G_i,G_{i+1}}$, and setting $\epsilon_{\mathsf{SqDDH}} = \max_{i\in\{1,2,3,7,8,9\}}(\epsilon_i)$, $\epsilon_{\mathsf{DDH}} = \max_{i\in\{4,5,6\}}(\epsilon_i)$. □

4 Analysis of General Problem Classes

Analyzing general problem classes instead of individual problems is important for at least two reasons: First, it improves our understanding of the properties that need to be satisfied by a problem to be intractable with respect to semi-generic algorithms. Second, master theorems for these classes alleviate the burden of analyzing future problems.

Generalized Pairing-Based Problems. Let a Type 1, 2, or 3 setting according to Definition 1 be given. Furthermore, let $\ell \in \mathbb{N}, d \in \{1,2,3\}$ be positive integers, $\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3} \subset \mathbb{Z}_p[X_1, \ldots, X_\ell]$ be finite sets of (publicly known) polynomials (called *input polynomials*) and $Q \in \mathbb{Z}_p[X_1, \ldots, X_\ell]$ be a single (publicly known) polynomial (called *challenge polynomial*). Then we define a $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ problem as: Given

$$((g_1^{R(\mathbf{x})})_{R\in\mathbf{I_1}}, (g_2^{R(\mathbf{x})})_{R\in\mathbf{I_2}}, (g_3^{R(\mathbf{x})})_{R\in\mathbf{I_3}}),$$

where $\mathbf{x} \xleftarrow{\$} \mathbb{Z}_p^\ell$ are secret random values, output $g_d^{Q(\mathbf{x})}$. A decisional variant of such problems can be defined analogously. In the following we always assume that the polynomial 1 is contained in each $\mathbf{I_i}$ which corresponds to the natural assumption that for each group a generator is given.

Informally speaking, a $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ problem is *non-trivial* if there is no way to compute Q using only the input polynomials and the operations on them which are implicitly given by the underlying bilinear setting. Let us restrict here to consider the case $d \in \{1,2\}$. Let $\mathbf{I_1} = \{R_1, \ldots, R_t\}$ and $\mathbf{I_2} = \{S_1, \ldots, S_{t'}\}$. Then using

Observation 2 (Section 2.1), one can see that the output $[c]_d$ of a semi-generic algorithm for the considered problem can be written as $g_d^{P(\mathbf{x})}$ for some P of the form

$$P = \begin{cases} \sum_{j=1}^{t} z_j R_j, & d = 1 \\ \sum_{j=1}^{t'} z'_j S_j, & d = 2 \text{ and Type 3 setting} \\ \sum_{j=1}^{t} z_j R_j + \sum_{j=1}^{t'} z'_j S_j, & d = 2 \text{ and Type 2 setting} \end{cases} \quad (1)$$

We call a $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ non-trivial if there is no P of the above form such that $g_d^{P(\mathbf{x})} = g_d^{Q(\mathbf{x})}$ *for all* $\mathbf{x} \in \mathbb{Z}_p^\ell$, i.e., if $P \neq Q \in \mathbb{Z}_p[X_1, \ldots, X_\ell]$. More formal and general definitions can be found in the full version of this paper [20].

Reductions for Generalized Problems. Theorem 3 extends the reduction we have seen for Co-DH in Section 3.1 to the more general class of $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ problems. The crucial difference and novelty lies in the technique for extracting the wanted discrete logarithm given the output of the semi-generic algorithm.

Theorem 3. *Let $d \in \{1, 2\}$ and $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ be a non-trivial problem with challenge and input polynomials in $\mathbb{Z}_p[X_1, \ldots, X_\ell]$. Let $k = \max_i(\deg_{X_i}(\mathbf{I_1} \cup \mathbf{I_2} \cup \mathbf{I_3}))$. Suppose there is a semi-generic algorithm $\mathcal{A}$ solving $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ in time t with success probability ϵ. Then there is an algorithm $\mathcal{B}$ solving $2k$-DL in $\mathbb{G}_3$ in time $t' \approx t + \tilde{O}(k' \log p)$, where $k' = \max(k, \deg(Q))$, with probability $\epsilon' \geq \frac{\epsilon}{\ell}$.*

Proof. Let $k_1 = 2k$. $\mathcal{B}$ takes as input a k_1-DL challenge $a_0 = g_3, a_1 = g_3^{x^1}, \ldots, a_{k_1} = g_3^{x^{k_1}}$. It then chooses $i^* \xleftarrow{\$} \{1, \ldots, \ell\}$ and $x_1, \ldots, x_{i^*-1}, x_{i^*+1}, \ldots, x_\ell \xleftarrow{\$} \mathbb{Z}_p$. The unknown x is treated as the secret choice x_{i^*} in the context of a $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ instance. We only sketch important points in the simulation of the semi-generic oracle: Each internal list $\mathcal{E}_j$ is initialized with the elements $(g_3^{P(\mathbf{x})})_{P \in \mathbf{I_j}}$ where for a polynomial $P = \sum_{e=(e_1,\ldots,e_\ell) \in E} b_e X_1^{e_1} \cdots X_\ell^{e_\ell}$, $E \subset \mathbb{Z}_p^\ell$, the element $g_3^{P(\mathbf{x})}$ can be computed as $g_3^{P(\mathbf{x})} = \prod_e a_{e_{i^*}}^{b_e \prod_{s \neq i^*} x_s^{e_s}}$ using the given instance of the k_1-DL problem. This is possible because the degree in X_{i^*} of the polynomials in each set $\mathbf{I_j}$ is upper bounded by k_1. Similarly, the table for simulating `BilinearMap` can be created since for each entry $g_3^{P(\mathbf{x})}$ in this table, P is again of degree at most k_1 in X_{i^*}.

Given an $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-$\mathsf{BDH}_{\mathbb{G}_d}$ instance, $\mathcal{A}$ eventually outputs an index $[c]_d$. Then c can be written as $g_3^{P(\mathbf{x})}$ for some known polynomial P as described in Equation 1. Thus, $\mathcal{A}$ wins if $Q(\mathbf{x}) \equiv P(\mathbf{x}) \bmod p$. Since $Z := Q - P$ is not zero modulo p (the problem is non-trivial) this success event can be split into disjoint events $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$, where $\mathcal{S}_j$ is defined as:

$$Z(X_1 = x_1, \ldots, X_{j-1} = x_{j-1}) \not\equiv 0 \text{ and } Z(X_1 = x_1, \ldots, X_j = x_j) \equiv 0 \quad (2)$$

Denoting the probability of event $\mathcal{S}_j$ by α_j we obtain $\epsilon = \alpha_1 + \cdots + \alpha_\ell$.

Now assume that event S_{i^*} occurs, which happens with probability ϵ/ℓ. Consider the polynomial $Z_{i^*} = Z(X_1 = x_1, \ldots, X_{i^*-1} = x_{i^*-1}) \bmod p \in \mathbb{Z}_p[X_{i^*}, \ldots, X_\ell]$. This polynomial is of the form $Z_{i^*} = \sum_{e=(e_{i^*},\ldots,e_\ell) \in E} b_e X_{i^*}^{e_{i^*}} \cdots X_\ell^{e_\ell}$, for some $E \subset$

$\mathbb{Z}_p^{\ell-i^*+1}$, where in at least one monomial the variable X_{i^*} appears with a non-zero exponent e_{i^*}. Let $M = b'_e X_{i^*}^{e'_{i^*}} \cdots X_\ell^{e'_\ell}$ be one of these monomials. Then consider the polynomial Z'_{i^*} we obtain by summing up all monomials of Z_{i^*} containing the submonomial $X_{i^*+1}^{e'_{i^*+1}} \cdots X_\ell^{e'_\ell}$:

$$Z'_{i^*} = \sum_{\substack{e=(e_{i^*},\ldots,e_\ell)\in E \\ e_{i^*+1}=e'_{i^*+1},\ldots,e_\ell=e'_\ell}} b_e X_{i^*}^{e_{i^*}} X_{i^*}^{e'_{i^*}} \cdots X_\ell^{e'_\ell}$$

Clearly, we have $Z'_{i^*} \not\equiv 0 \bmod p$ and since $Z_{i^*}(X_{i^*} = x_{i^*}) \equiv 0 \bmod p$ it also holds that $Z'_{i^*}(X_{i^*} = x_{i^*}) \equiv 0 \bmod p$. Hence, $x_{i^*} = x$ is a root of the non-zero uni-variate polynomial

$$Z''_{i^*} = \sum_{\substack{e=(e_{i^*},\ldots,e_\ell)\in E \\ e_{i^*+1}=e'_{i^*+1},\ldots,e_\ell=e'_\ell}} b_e X_{i^*}^{e_{i^*}}$$

Note that Algorithm $\mathcal{B}$ can easily construct the polynomial Z''_{i^*} by picking an *arbitrary* monomial from Z_{i^*} for which X_{i^*} appears with non-zero exponent. The coefficients b_e can also be easily computed since the coefficients of Z are known and $x_1, \ldots, x_{i^*-1}$ have been chosen by $\mathcal{B}$. So by applying an efficient standard algorithm for computing roots of polynomials over $\mathbb{Z}_p$, such as [36, Algorithm 14.15], $\mathcal{B}$ can find the wanted DL $x_{i^*} = x$ by computing all roots of the polynomial Z''_{i^*}. These at most $k' = \max(k, \deg(Q))$ different roots can be computed in time $\tilde{O}(k' \log p)$ [36, Corollary 14.16]. Whether a root x' equals x can be tested by verifying $g^{x'} \stackrel{?}{=} a_1$. □

We have also been able to find a reduction for a general class of decisional problems which is efficient for virtually all problems of this class considered in practice. Essentially, our reduction from the SqDDH problem over $\mathbb{G}_3$ works for all $(\mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3}, Q)$-BDDH$_{\mathbb{G}_3}$ problems where variables in $\mathbf{I_1} \cup \mathbf{I_2}$ and $\mathbf{I_3} \cup \{Q\}$ appear with at most linear and quadratic exponents, respectively. Our approach for this general reduction differs from the one for BDDH we have seen in Section 3.2 in the following way: The BDDH reduction is direct in the sense that *all reduction steps take place directly in the semi-generic model*. As an alternative, one could also first "project" BDDH to the group $\mathbb{G}_3$ by finding an "appropriate" problem which reduces *in a single step* to BDDH (with respect to semi-generic algorithms) and then *apply all* DDH *and* SqDDH *reduction steps to this problem in the standard model*. We follow this latter approach in our proof for general bilinear decisional problems since it has the advantage that we can resort to Bresson et al.'s results for generalized DDH problems [12] in the standard model. However, this is not straightforward. Since their results are quite restricted we need to enhance them to more general problem classes. For more details on our result for bilinear decisional problems we refer to the full version [20].

5 Analyzing Cryptosystems in the Semi-Generic Model

Besides for studying cryptographic hardness assumptions, it would also be interesting to use the SGGM as a tool to analyze the security of practical pairing-based cryptosystems. Similar analyzes have been made in the classical GGM [35, 13]. In this section

we consider the Boneh-Lynn-Shacham (BLS) signature scheme [9, 10] in the SGGM. It turns out that it is possible to prove security of this scheme under the semi-generic groups heuristic, by requiring concrete (but non-standard) properties of the hash function.

The BLS signature scheme over a Type 1 bilinear setting is defined as follows. Let H_1 be a hash function $H_1 : \{0,1\}^\ell \to \mathbb{G}_1$.

- Gen samples a random generator g of $\mathbb{G}_1$, $s \xleftarrow{\$} \mathbb{Z}_p$, and sets $pk = (g, g^s)$, $sk = s$.
- $\mathsf{Sign}(sk, m)$ computes $H_1(m)$ and returns $\sigma = H_1(m)^s$.
- $\mathsf{Verify}(pk, m, \sigma)$ returns 1, if $e(H_1(m), pk) = e(\sigma, g)$, and 0 otherwise.

Let us now describe the EUF-CMA security experiment for the BLS signature scheme in the SGGM. Here we are facing a technical problem: the BLS scheme utilizes a hash function $H_1 : \{0,1\}^\ell \to \mathbb{G}_1$, that is, the output of this map is a group element in some given representation. However, in the SGGM we want to consider algorithms which are *independent* of a particular representation of elements of $\mathbb{G}_1$. Since in our model elements of $\mathbb{G}_1$ are given as list indices, we have no representation of group elements that we could use as the range of the hash function.

One possible solution would be to fall back on the formalization of a generic group by Shoup [34]. In this model, group elements are represented by unique random bit strings. Thus, we could use a hash function that maps to bit strings of appropriate size. However, the fact that group elements are encoded as *random* strings has been subject to much criticism [16, 26, 15]. For instance, the Shoup model can be misused to implement a random oracle, which is of no avail since we want to avoid random oracles in our security proof. Therefore we follow a different approach. We implement H_1 as a generic *group hash function*.

Definition 2. *A group hash function is a pair of algorithms* $\mathbf{H} = (\mathsf{GHGen}, \mathsf{GHEval})$.

- GHGen *takes as input a generator* g *of* $\mathbb{G}_1$*, and returns* $A = (a_1, \ldots, a_\delta) \in \mathbb{G}_1^\delta$*. Vector* A *specifies a function* $H_1 : \{0,1\}^\ell \to \mathbb{G}_1$*.*
- *Algorithm* GHEval *takes as input a vector* $A \in \mathbb{G}_1^\delta$ *and a string* $m \in \{0,1\}^\ell$*, and returns* $H_1(m) \in \mathbb{G}_1$*.*

We say that a group hash function is generic*, if* GHGen *and* GHEval *perform only group operations on elements of* A*.*

Examples of generic group hash functions are the hash function used in Water's IBE scheme [37] and the programmable hash functions of Hofheinz and Kiltz [18].

Generic group hash functions have the useful property that there exist "trapdoor" set-up and evaluation algorithms $(\mathsf{TrapGen}, \mathsf{TrapEval})$ with the following properties.

- $\mathsf{TrapGen}$ takes as input a generator $g \in \mathbb{G}_1$. It returns a vector $A \in \mathbb{G}_1^\delta$, distributed identically to the output of GHGen for all g, and some trapdoor information td.
- Algorithm $\mathsf{TrapEval}$ takes as input a vector $A \in \mathbb{G}_1^\delta$ and a string $m \in \{0,1\}^\ell$, and returns h such that $g^h = H_1(m)$.

For the security proof we need to demand a strong form of collision resistance.

Definition 3. *A group hash function is* (ϵ, t, q)-algebraic collision resistant*, if*

$$\Pr\left[\mathcal{A}(A) = (m_0, ..., m_q, i_0, ..., i_q) : H_1(m_0) = g^{i_0} \prod_{j=1}^{q} (H_1(m_j))^{i_j}\right] \leq \epsilon$$

for all algorithms $\mathcal{A}$ running in time t.

By employing techniques from [18] it is possible to construct hash functions satisfying this property under weak assumptions, like the hardness of computing discrete logarithms in $\mathbb{G}_1$, for any *constant* q. A major drawback is, however, that for these constructions the size δ of vector A grows at least linearly with q. We leave it as an open problem to study whether there exists a (possibly probabilistic) trapdoor group hash function such that δ is constant and $q = q(\kappa)$ is a polynomial.

We formalize the EUF-CMA experiment in the SGGM as follows. At the beginning of the game, the challenger samples a random generator g and a secret key x. Then it runs $(a_1, \ldots, a_\delta) \xleftarrow{\$} \mathsf{GHGen}(g)$, sets $I_1 := (g, g^x, a_1, \ldots, a_\delta)$, and implements a semi-generic oracle with input I_1 as described in Section 2. This provides the adversary with the public key, and the ability to perform group operations on elements of $\mathbb{G}_1$.

When the adversary queries a signature for some chosen message m_i, the challenger computes $H(m_i)^x$ and appends it to the list $\mathcal{E}_1$.

We say that the adversary *wins* the game, if it outputs a message m and index $[s]_1$ such that $s = H(m)^x$, that is, the adversary has computed a valid signature for m. We say that a semi-generic adversary $\mathcal{A}$ (ϵ, t)-*breaks* the EUF-CMA security of a signature scheme if $\mathcal{A}$ runs in time t and $\Pr[\mathcal{A} \text{ wins}] \geq \epsilon$.

Theorem 4. *Suppose there exists an adversary $\mathcal{A}$ (ϵ, t)-breaking the EUF-CMA security of the BLS signature scheme in the semi-generic model by making q chosen-message signature queries. Then there exists an algorithm $\mathcal{B}_{\mathsf{coll}}$ $(\epsilon_{\mathsf{dl}}, t_{\mathsf{dl}}, q)$-breaking the algebraic collision resistance of H_1 and an algorithm $\mathcal{B}_{\mathsf{dl}}$ $(\epsilon_{\mathsf{dl}}, t_{\mathsf{dl}})$-solving the discrete logarithm problem in $\mathbb{G}_1$, such that $t \approx t_{\mathsf{coll}} \approx t_{\mathsf{dl}}$ and $\epsilon \leq \epsilon_{\mathsf{coll}} + \epsilon_{\mathsf{dl}}$.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that outputs a message m and an index $[s]_1$ such that $s = H(m)^x$. In the SGGM, an adversary has to compute a group element of $\mathbb{G}_1$ by applying a sequence of group operations to the initial values $(g, g^x, a_1, \ldots, a_\delta)$ stored in $\mathcal{E}_1$ and to group elements added to the list by the challenger oracle in response to chosen-message signature queries. Thus, when $\mathcal{A}$ outputs $(m, [s]_1)$ such that $s = H(m)^x$, then the oracle obtains an equation

$$H(m)^x = g^{\alpha_1} \cdot (g^x)^{\alpha_2} \cdot \prod_{i=1}^{\delta} a_i^{\beta_i} \cdot \prod_{i=1}^{q} (H(m_i)^x)^{\gamma_i}\,, \tag{3}$$

or equivalently $x \cdot \big(\log_g H(m) - \sum_{i=1}^{q} \gamma_i \log_g H(m_i) - \alpha_2\big) = \alpha_1 + \sum_{i=1}^{\delta} \beta_i \log_g a_i$, for integers $\alpha_i, \beta_i, \gamma_i$ known to the oracle. We consider two types of forgers:

1. A Type-A forger performs a sequence of operations such that

$$\log_g H(m) - \sum_{i=1}^{q} \gamma_i \log_g H(m_i) - \alpha_2 \equiv 0 \bmod p. \tag{4}$$

2. A Type-B forger performs a sequence of operations such that

$$\log_g H(m) - \sum_{i=1}^{q} \gamma_i \log_g H(m_i) - \alpha_2 \not\equiv 0 \bmod p. \tag{5}$$

Lemma 1. *Suppose there exists a Type-A forger $\mathcal{A}$ (ϵ, t)-breaking the EUF-CMA security of the BLS signature scheme by making at most q chosen-message queries. Then there exists an algorithm $\mathcal{B}_{\mathsf{coll}}$ $(\epsilon_{\mathsf{dl}}, t_{\mathsf{dl}}, q)$-breaking the algebraic collision resistance of $(\mathsf{GHGen}, \mathsf{GHEval})$ in time $t' \approx t$ with success probability $\epsilon_{\mathsf{coll}} \geq \epsilon$.*

PROOF. Algorithm $\mathcal{B}_{\mathsf{coll}}$ receives as input a vector $A' = (g', a'_1, \ldots, a'_\delta)$. It proceeds exactly like the semi-generic EUF-CMA challenger, except that it sets $g := g'$ and $a_i := a'_i$ instead of sampling g at random and generating A by running $\mathsf{GHGen}(g)$. Thus, in particular $\mathcal{B}_{\mathsf{coll}}$ chooses the secret key $x \xleftarrow{\$} \mathbb{Z}_p$ and thus is able to simulate the original challenger perfectly.

When $\mathcal{A}$ outputs $(m, [s]_1)$ such that $s = H(m)^x$, then $\mathcal{B}_{\mathsf{coll}}$ computes and returns integers $(\alpha_2, \gamma_1, \ldots, \gamma_q)$ as in Equation 4. Observe that if Equation 4 is satisfied, then we have $H(m) = g^{\alpha_2} \cdot \prod_{i=1}^{q} H(m_i)^{\gamma_i}$. ▲

Lemma 2. *Suppose there exists a Type-B forger $\mathcal{A}$ (ϵ, t)-breaking the EUF-CMA security of the BLS signature scheme. Then there exists an algorithm $\mathcal{B}_{\mathsf{dl}}$ solving the discrete logarithm problem in $\mathbb{G}_1$ in time $t_{\mathsf{dl}} \approx t$ with success probability $\epsilon_{\mathsf{dl}} \geq \epsilon$.*

PROOF. Algorithm $\mathcal{B}_{\mathsf{dl}}$ receives as input a tuple (g', y). It sets $g := g'$, $g^x := y$, and runs $(A, td) \xleftarrow{\$} \mathsf{TrapGen}(g)$ to generate the public parameters of the hash function. Recall that A is distributed identically to some A' generated by GHGen. It sets $I_1 := (g, g^x, a_1, \ldots, a_\delta)$, and implements a semi-generic oracle with initial list state I_1.

Since $\mathcal{B}_{\mathsf{dl}}$ does not know the secret-key exponent x, it answers chosen-message signature queries of $\mathcal{A}$ differently. $\mathcal{B}_{\mathsf{dl}}$ makes use of the trapdoor information td generated along with A. Whenever $\mathcal{A}$ submits a chosen-message m_i, $\mathcal{B}_{\mathsf{dl}}$ computes $h_i = \mathsf{TrapEval}(m_i)$ and appends y^{h_i} to $\mathcal{E}_1$. Note that $y^{h_i} = g^{x \log_g H(m_i)} = H(m_i)^x$, thus this is a valid signature.

When $\mathcal{A}$ outputs $(m, [s]_1)$ such that $s = H(m)^x$, then $\mathcal{B}_{\mathsf{dl}}$ computes integers $(\alpha_i, \beta_i, \gamma_i)$ as in Equation 3, and returns

$$x = \log_{g'} y = \frac{\alpha_1 + \sum_{i=1}^{\delta} \beta_i \log_g a_i}{\log_g H(m) - \sum_{i=1}^{q} \gamma_i \log_g H(m_i) - \alpha_2} \bmod p,$$

which is possible since $\log_g H(m) - \sum_{i=1}^{q} \gamma_i \log_g H(m_i) - \alpha_2 \neq 0 \bmod p$. ▲

□

Acknowledgements. We would like to thank Dennis Hofheinz, Jesper Buus Nielsen, and Dominique Unruh for valuable discussions and the anonymous reviewers of Asiacrypt 2010 for their detailed and helpful comments.

References

1. D. Aggarwal and U. Maurer. Breaking RSA generically is equivalent to factoring. In Antoine Joux, editor, *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 36–53. Springer, 2009.
2. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
3. D. Boneh. Number-theoretic assumptions. Invited Talk at TCC's Special Session on Assumptions for Cryptography, 2007.
4. D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *LNCS*, pages 223–238. Springer, 2004.
5. D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext (full paper). Cryptology ePrint Archive, Report 2005/015, 2005. `http://eprint.iacr.org/`.
6. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 41–55. Springer, 2004.
7. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.
8. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.
9. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *LNCS*, pages 514–532. Springer, 2001.
10. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, 2004.
11. X. Boyen. The Uber-Assumption family. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing*, volume 5209 of *LNCS*, pages 39–56. Springer, 2008.
12. E. Bresson, Y. Lakhnech, L. Mazaré, and B. Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *LNCS*, pages 482–499. Springer, 2007.
13. D. R. L. Brown. Generic groups, collision resistance, and ECDSA. *Des. Codes Cryptography*, 35(1):119–152, 2005.
14. J. Cheon. Security analysis of the Strong Diffie-Hellman problem. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *LNCS*, pages 1–11. Springer, 2006.
15. A. W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *LNCS*, pages 100–109. Springer, 2002.
16. M. Fischlin. A note on security proofs in the generic model. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *LNCS*, pages 458–469. Springer, 2000.
17. S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
18. D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *CRYPTO*, volume 5157 of *LNCS*, pages 21–38. Springer, 2008.
19. S. Hohenberger. The cryptographic impact of groups with infeasible inversion. Master's thesis, Massachusetts Institute of Technology, 2003.
20. T. Jager and A. Rupp. The semi-generic group model and applications to pairing-based cryptography (full paper), 2010. `http://www.nds.rub.de/chair/publications/`.

21. T. Jager and J. Schwenk. On the analysis of cryptographic assumptions in the generic ring model. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 399–416. Springer, 2009.
22. D. Jao and K. Yoshida. Boneh-Boyen signatures and the Strong Diffie-Hellman problem. Cryptology ePrint Archive, Report 2009/221, 2009. `http://eprint.iacr.org/`.
23. A. Joux. A one round protocol for tripartite Diffie-Hellman. In Wieb Bosma, editor, *ANTS*, volume 1838 of *LNCS*, pages 385–394. Springer, 2000.
24. A. Joux. A one round protocol for tripartite Diffie-Hellman. *J. Cryptology*, 17(4):263–276, 2004.
25. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *LNCS*, pages 146–162. Springer, 2008.
26. N. Koblitz and A. Menezes. Another look at generic groups. *Advances in Mathematics of Communications*, 1:13–28, 2007.
27. G. Leander and A. Rupp. On the equivalence of RSA and factoring regarding generic ring algorithms. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 241–251. Springer, 2006.
28. U. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *LNCS*, pages 1–12. Springer, 2005.
29. A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.
30. S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.
31. J. M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.
32. R. L. Rivest. On the notion of pseudo-free groups. In Moni Naor, editor, *TCC*, volume 2951 of *LNCS*, pages 505–521. Springer, 2004.
33. A. Rupp, G. Leander, E. Bangerter, A. W. Dent, and A. Sadeghi. Sufficient conditions for intractability over black-box groups: Generic lower bounds for generalized DL and DH problems. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *LNCS*, pages 489–505. Springer, 2008.
34. V. Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *LNCS*, pages 256–266. Springer, 1997.
35. N. P. Smart. The exact security of ECIES in the generic group model. In B. Honary, editor, *IMA Int. Conf.*, volume 2260 of *LNCS*, pages 73–84. Springer, 2001.
36. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, second edition, 2003.
37. B. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 114–127. Springer, 2005.
38. S. Wolf. *Information-theoretically and computationally secure key agreement in cryptography*. PhD thesis, ETH Zurich, 1999. ETH dissertation No. 13138.