An Analysis of the Algebraic Group Model

Jonathan Katz* Cong Zhang† Hong-Sheng Zhou‡

October 1, 2022

Abstract

The algebraic group model (AGM), formalized by Fuchsbauer, Kiltz, and Loss, has recently received significant attention. One of the appealing properties of the AGM is that it is viewed as being (strictly) weaker than the generic group model (GGM), in the sense that hardness results for algebraic algorithms imply hardness results for generic algorithms, and generic reductions in the AGM (namely, between the algebraic formulations of two problems) imply generic reductions in the GGM. We highlight that as the GGM and AGM are currently formalized, this is not true: hardness in the AGM may not imply hardness in the GGM, and a generic reduction in the AGM may not imply a similar reduction in the GGM.

1 Introduction

Computational assumptions in groups. Since the work of Diffie and Hellman [DH76], there have been many elegant cryptographic schemes and protocols whose security can be based on the conjectured hardness of certain computational problems in (cyclic) groups. To prove security in this setting, we begin by formulating an appropriate hardness assumption relative to a group $\mathbb{G}$. It is important to stress that such assumptions are always relative to some *specific encoding* of the elements of $\mathbb{G}$, even though this is not always made explicit. For example, let $\mathbb{G}$ denote the cyclic group of order p, for some large prime p such that $q = 2p + 1$ is also prime. One way to encode elements of $\mathbb{G}$ is to represent them as integers in the order-p subgroup of $\mathbb{Z}_q^*$, with the group operation corresponding to multiplication modulo q. Another way to encode elements of $\mathbb{G}$ is to represent them as integers in $\mathbb{Z}_p$ with the group operation corresponding to addition modulo p. Even though these are both encodings of the same group (or, put differently, these two encodings are isomorphic), it is

*University of Maryland, jkatz@cs.umd.edu.

†Zhejiang University and ZJU-Hangzhou Global Scientific and Technological Innovation Center, congresearch@gmail.com. Work supported in part by Zhejiang University Education Foundation Qizhen Scholar Foundation. Portions of this work were done while at the University of Maryland.

‡Virginia Commonwealth University, hszhou@vcu.edu. Work supported in part by NSF grant CNS-1801470, a Google Faculty Research Award, and a research gift from Ergo Platform.

reasonable to conjecture that the discrete-logarithm problem is hard in the first case even though it is trivial to solve in the second case. Encodings matter.

Beyond understanding the hardness of specific problems in groups, it is also interesting to understand relations between different problems. Here, too, the specific encoding may affect the relations that can be shown.

Unfortunately, the current state-of-the-art in complexity theory does not allow us to prove any unconditional hardness results relative to any concrete group encoding; namely, we do not know how to prove lower bounds on the probability with which arbitrary algorithms can solve some problem relative to any specific encoding of group elements. (On the other hand, we can in some cases show unconditional relations between certain problems, e.g., that—for any encoding—hardness of the decisional Diffie-Hellman assumption implies hardness of the discrete-logarithm problem.) This has motivated researchers to investigate the possibility of proving hardness results for *specific* (restricted) classes of algorithms. Two examples we study in this work are the class of *generic* algorithms, and the class of *algebraic* algorithms. We discuss these in more detail below.

Generic algorithms and the generic group model. Roughly speaking, generic algorithms operate independently of any particular group encoding. That is, they ignore the specific encoding of group elements but instead treat group elements "generically." Studying this class of algorithms is well motivated, since several well-known algorithms such as the baby-step/giant-step algorithm [PH78] and Pollard's rho algorithm [Pol78] are generic in this sense. A generic algorithm has the advantage that it works for *any* encoding of group elements; it cares only about the mathematical structure of the underlying group, but not its encoding. Researchers have proposed different variants of the so-called *generic group model* (GGM) [Nec94, Sho97, Mau05, MPZ20] in an effort to formally define the notion of a generic algorithm. We describe these in Section 2.1.

It is possible to prove unconditional hardness results in the generic group model. While the implications of such results for hardness relative to any specific encoding are unclear, at a minimum a proof of hardness in the GGM serves as a "sanity check" that some assumption is reasonable. Indeed, the GGM is now a canonical tool to establish (some level of) confidence for new hardness assumptions or even security of cryptographic schemes. Moreover, for some specific group encodings (e.g., appropriately defined elliptic-curve groups) and certain problems, the best known algorithms are indeed generic.

Algebraic algorithms and the algebraic group model. Other work [BV98, PV05] has proposed a class of so-called *algebraic* algorithms. Roughly speaking, algebraic algorithms are allowed to exploit the concrete encoding of group elements, but they are restricted to only being able to derive (new) group elements via group operations involving elements they have been provided with as input. Fuchsbauer, Kiltz, and Loss [FKL18] recently formalized this idea as the *algebraic group model* (AGM), and showed a number of results in that model. A number of papers have since extended those results [MTT19, KLX20, BFL20, ABK+21, GT21], and have used the AGM to prove security of cryptographic constructions [MBKM19, RS20, KLX22, FPS20, ABB+20, RZ21].

The utility of studying the AGM is not immediately clear, and we are not aware of any

natural group-theoretic algorithms that are algebraic but not generic.[1] We are also not aware of any unconditional hardness results for problems of cryptographic interest in the AGM. (Though lower bounds for some problems are possible in an extension of the AGM [KLX20].) Nevertheless, Fuchsbauer, Kiltz, and Loss argue that the AGM can be useful for studying *reductions* between problems. As an example, for many group encodings the best-known algorithm for solving the computational Diffie-Hellman problem is to first solve the discrete-logarithm problem. In the AGM, one can prove that this is inherent, in the sense that hardness of the latter implies hardness of the former. Such a result is not known to hold in general.

To justify the usefulness of studying reductions in the AGM, Fuchsbauer et al. [FKL18, Lemma 2.2] claim that a generic reduction between two problems in the AGM implies a generic reduction between those problems in the GGM. That is, if there is a generic reduction R showing that the hardness of (algebraic) security game $\mathbf{H}$ implies hardness of (algebraic) security game $\mathbf{G}$, and if $\mathbf{H}$ can be proven unconditionally hard for generic algorithms, then $\mathbf{G}$ is also hard in the GGM. Their proof of this claim uses the following natural steps:

Step 1: Assume toward a contradiction that $\mathbf{G}$ is not hard in the GGM, so there is a generic algorithm $\mathsf{A}^{\mathbf{G}}_{\mathsf{gen}}$ that succeeds in game $\mathbf{G}$ with high probability.

Step 2: Since any generic algorithm is also algebraic, the reduction R can be applied to $\mathsf{A}^{\mathbf{G}}_{\mathsf{gen}}$ to obtain an algebraic algorithm $\mathsf{A}^{\mathbf{H}}_{\mathsf{alg}} := \mathsf{R}^{\mathsf{A}^{\mathbf{G}}_{\mathsf{gen}}}$ that succeeds with high probability in $\mathbf{H}$.

Step 3: Since R is generic, $\mathsf{A}^{\mathbf{H}}_{\mathsf{alg}}$ is in fact a generic algorithm. But this contradicts the fact that $\mathbf{H}$ is unconditionally hard for generic algorithms.

While the above is appealing, some steps are not entirely clear. In particular, it is not obvious that the intuitive conversion of a generic algorithm to an algebraic algorithm (cf. step 2) is applicable in all contexts. And even if it is possible to transform the generic algorithm $\mathsf{A}^{\mathbf{G}}_{\mathsf{gen}}$ to an "equivalent" algebraic algorithm $\mathsf{A}^{\mathbf{G}}_{\mathsf{alg}}$, it is then not clear that the resulting algebraic algorithm $\mathsf{R}^{\mathsf{A}^{\mathbf{G}}_{\mathsf{alg}}}$ can be meaningfully transformed back into a generic algorithm (cf. step 3).

1.1 Our Results

Seeking to better understand the algebraic group model and its relationship to the generic group model, we provide self-contained descriptions of both and then explore their relationship. We first observe that the formal definition of algebraic algorithms proposed by Fuchsbauer et al. may not match the intended intuition. Specifically, Fuchsbauer et al. define an algorithm to be algebraic if it provides a *representation* of any group elements it outputs. (See more details in Section 3.) This is supposed to ensure that "the only way

[1] Fuchsbauer et al. claim that index-calculus algorithms are algebraic, but without any further explanation. It is not clear to us what they mean by this.

for an algebraic algorithm to output a new group element is to derive it via group multiplication from known group elements" [FKL18]. However, we show in Section 3 an algorithm that obtains a new group element using non-group operations but can still output a valid representation of that element.

More importantly, we show that a generic algorithm need not be algebraic, and that it might be hard to convert a generic algorithm to an algebraic one with the same behavior. In particular, we show a counterexample to the claim of Fuchsbauer et al. as described above by showing a security game called the "**b**inary **e**ncoding **g**ame (**beg**)" and describing a generic reduction from the discrete-logarithm problem to this game. (Note that the discrete-logarithm problem is unconditionally hard in the GGM, and is conjectured to be hard for certain encodings.) But we show that **beg** is easy in the GGM. Thus:

Theorem 1.1 (Informal)**.** *A generic reduction in the AGM does not imply a generic reduction in the GGM.*

Concurrent work. In concurrent and independent work, Zhandry [Zha22] studies the GGM and the AGM and gives a new definition of the AGM. We consider the AGM as originally defined by Fuchsbauer et al. [FKL18]. Zhandry does not address the relationship between generic reductions in the AGM vs. the GGM, and does not show any analogue of our theorem stated above.

Discussion. Our counterexample to [FKL18, Lemma 2.2] is admittedly contrived, and an important next step is to understand whether there is some subclass of security games for which a version of their lemma might still apply. Any such subclass should of course be broad enough to include security games of cryptographic relevance. More generally, we believe that a more-formal treatment of the GGM and AGM, and the relationship between them, is warranted.

2 Preliminaries

In this section, we provide the required background and preliminaries.

Algorithms. We denote by $s \leftarrow S$ uniform sampling of variable s from the finite set S. Algorithms are written using uppercase letters (e.g., A, B). To indicate that a probabilistic algorithm A runs on some inputs $(x_1, \ldots, x_n)$ and returns y, we write $y \leftarrow \mathsf{A}(x_1, \ldots, x_n)$. If A has oracle access to an algorithm B during its execution, we write $y \leftarrow \mathsf{A}^{\mathsf{B}}(x_1, \ldots, x_n)$.

Group encodings. Throughout this work, we restrict attention to the cyclic group $\mathbb{G}$ of prime order p. For concreteness, we often identify $\mathbb{G}$ with the additive group $\mathbb{Z}_p$. As highlighted in the Introduction, however, we explicitly focus on *encodings* of this group and its impact on algorithms for various problems.

Fix some $\ell \geq \lceil \log p \rceil$. An encoding $\sigma : \mathbb{Z}_p \to \{0,1\}^\ell$ is simply an injective map from $\mathbb{Z}_p$ to $\{0,1\}^\ell$. We let id be the "trivial" encoding in which each element of $\mathbb{Z}_p$ is encoded as a binary integer in the range $\{0, \ldots, p-1\}$ using $\lceil \log p \rceil$ bits and then padded to the left

with 0s to a string of length ℓ, and the group operation is addition modulo p. We often use boldface capital letters (e.g., $\mathbf{X}, \mathbf{Y}$) for encodings of group elements.

As notational shorthand, we will often use standard multiplicative notation for group operations on (encodings of) group elements. Thus, $\sigma(x)\sigma(y)$ refers to computing the group operation on the group elements $\sigma(x), \sigma(y)$; note that $\sigma(x)\sigma(y) = \sigma(x+y \bmod p)$. Similarly, for r an integer, $\sigma(x)^r$ refers to computing the r-fold group operation on $\sigma(x)$; of course, $\sigma(x)^r = \sigma(xr \bmod p)$.

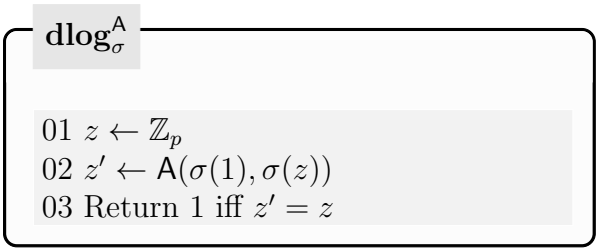

Figure 1: The discrete-logarithm game **dlog**.

Security games. We use a variant of code-based security games [BR06]. A game $\mathbf{G}_\sigma$, parameterized by an encoding σ, has a main procedure and (possibly zero) oracle procedures that describe how oracle queries are answered. Figure 1 shows an example of the discrete-logarithm game. We let $\mathbf{G}^{\mathsf{A}}_\sigma$ be a random variable denoting the boolean output of game $\mathbf{G}_\sigma$ played by algorithm A. Algorithm A is said to *succeed* when $\mathbf{G}^{\mathsf{A}}_\sigma = 1$, and the success probability of A in $\mathbf{G}_\sigma$ is $\mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}_\sigma} \stackrel{\text{def}}{=} \Pr[\mathbf{G}^{\mathsf{A}}_\sigma = 1]$. $\mathbf{Time}^{\mathsf{A}}_{\mathbf{G}_\sigma}$ denotes the running time of $\mathbf{G}^{\mathsf{A}}_\sigma$.

Security reductions. Let $\mathbf{G}_\sigma, \mathbf{H}_\sigma$ be security games. We write $\mathbf{H}_\sigma \xrightarrow{(\Delta_t, \Delta_\epsilon)} \mathbf{G}_\sigma$ if there is an algorithm R (a *reduction*) such that for all algorithms A, algorithm $\mathsf{B} := \mathsf{R}^{\mathsf{A}}$ satisfies

$$\mathbf{Succ}^{\mathsf{B}}_{\mathbf{H}_\sigma} \geq \frac{1}{\Delta_\epsilon} \cdot \mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}_\sigma}, \quad \mathbf{Time}^{\mathsf{B}}_{\mathbf{H}_\sigma} \leq \Delta_t \cdot \mathbf{Time}^{\mathsf{A}}_{\mathbf{G}_\sigma}.$$

Note that the reduction may depend on the encoding, and a reduction with some parameters may exist for certain encodings and not others. (For examples of reductions that depend on the encoding, see [Gal12, Section 21.4].)

2.1 Generic Algorithms

In general, an algorithm A in a game $\mathbf{G}_\sigma$ may depend on σ. A *generic* algorithm, however, should be "oblivious" to the encoding used. At least two ways of formalizing this have been considered, one due to Shoup [Sho97] and another due to Maurer [Mau05].

Shoup's approach can be summarized as requiring a generic algorithm A to work for all encodings. Since A cannot depend on the encoding, however, it must be provided with some way to perform group operations. We can provide such capabilities (both to A and possibly the game itself) by giving access to two oracles that we collectively call *encoding oracles*:

- a *labeling oracle* that takes as input $x \in \mathbb{Z}_p$ and returns $\sigma(x)$, and
- a *group-operation oracle* that takes as input strings s_1, s_2 and does the following: if $s_1 = \sigma(x)$ and $s_2 = \sigma(y)$, return $\sigma(x + y \bmod p)$; otherwise, return $\perp$.

Calls to these oracles take unit time by definition. We denote by $\widehat{\mathbf{G}}_\sigma$ the modification of a game $\mathbf{G}_\sigma$ to include the above oracles. We define[2] $\mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}} = \min_\sigma\{\mathbf{Succ}^{\mathsf{A}}_{\widehat{\mathbf{G}}_\sigma}\}$ and $\mathbf{Time}^{\mathsf{A}}_{\mathbf{G}} = \max_\sigma\{\mathbf{Time}^{\mathsf{A}}_{\widehat{\mathbf{G}}_\sigma}\}$.

Maurer's approach to defining the generic group model is similar in spirit, but technically different. Here, roughly speaking, a generic algorithm does not have access to any encodings of group elements at all; instead, the algorithm is able to access group elements only via abstract "handles." One way to formalize this is by initializing a counter ctr to 1, and a table T to empty, at the beginning of an algorithm's execution. The algorithm now has access to three encoding oracles that take the following form:

- the labeling oracle takes as input $x \in \mathbb{Z}_p$. It stores (ctr, x) in T and increments ctr. (It does not return anything.)
- the group-operation oracle takes as input positive integers $i, j < \mathsf{ctr}$. It finds (i, x) and (j, y) in T, stores $(\mathsf{ctr}, x + y \bmod p)$ in T, and increments ctr. (It does not return anything.)
- an *equality oracle* takes as input positive integers $i, j < \mathsf{ctr}$. It finds (i, x) and (j, y) in T and returns 1 if $x = y$ and 0 otherwise.

Note that ctr can also be incremented, and T populated, by actions that occur as part of the game itself rather than due to actions of the algorithm. For example, the discrete-logarithm game of Figure 1 would be modified to store $(1, x)$ in T and increment ctr as part of step 1; it would also provide no input to A in step 2. Moreover, if A is supposed to output a group element in some game, then it should instead output a positive integer $i < \mathsf{ctr}$; this will correspond to an output of $\sigma(x)$, where (i, x) is the record stored in T. If we let $\widetilde{\mathbf{G}}$ denote the appropriate modification of a game $\mathbf{G}$, then we again define $\mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}} = \min_\sigma\{\mathbf{Succ}^{\mathsf{A}}_{\widetilde{\mathbf{G}}_\sigma}\}$ and $\mathbf{Time}^{\mathsf{A}}_{\mathbf{G}} = \max_\sigma\{\mathbf{Time}^{\mathsf{A}}_{\widetilde{\mathbf{G}}_\sigma}\}$.

We refer to *Shoup-generic* and *Maurer-generic* algorithms depending on the model under consideration. With respect to either model, we say a game $\mathbf{G}$ is (t, ϵ)-hard in the generic group model if for every generic algorithm A it holds that $\mathbf{Time}^{\mathsf{A}}_{\mathbf{G}} \leq t \Rightarrow \mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}} \leq \epsilon$.

A generic algorithm A (in either model) with success probability $\epsilon = \mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}}$ may fail to run in a "standard" game $\mathbf{G}_\sigma$ where the encoding oracles are not present. However, for any σ it is possible to modify a generic algorithm A (of either type) in a black-box way (by simulating the encoding oracles) to obtain an algorithm A_σ where $\mathbf{Succ}^{\mathsf{A}_\sigma}_{\mathbf{G}_\sigma} = \epsilon$ and the time complexity of A_σ relative to A reflects only the time required to perform group operations for the encoding σ.

[2] While one might expect $\mathbf{Succ}^{\mathsf{A}}_{\widehat{\mathbf{G}}_\sigma}$ and $\mathbf{Time}^{\mathsf{A}}_{\widehat{\mathbf{G}}_\sigma}$ to be independent of σ (and that is the case for "natural" generic algorithms), that may not be the case in general.

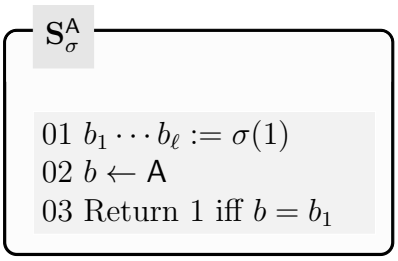

Figure 2: Game $\mathbf{S}$.

For completeness, we remark that there can be games where the optimal success probabilities for generic algorithms differ depending on which generic group model is used. Consider, for example, game $\mathbf{S}$ in Figure 2. With respect to Shoup's notion of generic algorithms, there exists a trivial algorithm A that has success probability 1 for any encoding. (A simply asks its encoding oracle for $\sigma(1)$ and outputs the first bit.) On the other hand, with respect to Maurer's notion of generic algorithms it is not possible to have an algorithm that achieves success probability better than $1/2$ for all encodings.

Generic reductions. For games $\mathbf{G}$, $\mathbf{H}$, we write $\mathbf{H} \xRightarrow[\text{S-GGM}]{(\Delta_t,\Delta_\epsilon)} \mathbf{G}$ if there is a generic *reduction* R (where generic is defined relative to Shoup's model) such that for all generic algorithms A, algorithm $\mathsf{B} := \mathsf{R}^{\mathsf{A}}$ (which is generic) satisfies

$$\mathbf{Succ}^{\mathsf{B}}_{\mathbf{H}} \geq \frac{1}{\Delta_\epsilon} \cdot \mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}}, \quad \mathbf{Time}^{\mathsf{B}}_{\mathbf{H}} \leq \Delta_t \cdot \mathbf{Time}^{\mathsf{A}}_{\mathbf{G}}.$$

We define $\mathbf{H} \xRightarrow[\text{M-GGM}]{(\Delta_\epsilon,\Delta_t)} \mathbf{G}$ analogously with respect to Maurer's model.

3 Algebraic Algorithms

Algebraic algorithms are another example of a class of algorithms that has been considered in the context of group-theoretic problems. The main idea, which seems to have originated in work of Paillier and Vergnaud [PV05], is to try to capture the notion of an algorithm that, on the one hand, only performs group operations on group elements (as in the generic group model) but, on the other hand, can depend on a specific encoding σ rather than being "encoding-agnostic." As one might expect, formalizing this intuition is not straightforward. The main difficulty is that, with an encoding σ fixed, it is no longer clear how to differentiate between arbitrary computations on group elements done by an algorithm and group operations on group elements (that may depend on σ).

Fuchsbauer et al. [FKL18] suggest one way to resolve the above dilemma. Roughly speaking, they do not attempt to place any restrictions on intermediate computations done by an algorithm, but instead require that any group elements output by an algorithm must[3] be accompanied by a *representation* relative to the ordered set S of group elements (the *base*

[3]Formally, if an algorithm violates these requirements in some game, then by definition it does not succeed.

set) provided to that algorithm as input. (A representation of a group element $\sigma(y)$ relative to an ordered set of group elements $S = (\sigma(x_1), \ldots, \sigma(x_k))$ is a vector $\vec{r} = (r_1, \ldots, r_k) \in \mathbb{Z}_p^k$ such that $\sigma(y) = \prod_i \sigma(x_i)^{r_i}$. Note this implies $y = \sum_i r_i x_i \bmod p$.) To ensure nontriviality, we assume the set S always includes $\sigma(1)$ (i.e., $\sigma(1)$ is always provided to the algorithm as input). To be clear: (1) group elements received by the algorithm as a result of an oracle call are added to the base set (in particular, the base set can expand during the course of executing the algorithm; a valid representation must always be relative to the current set), and (2) an algebraic algorithm must also provide a representation for any group elements it provides as input to some oracle call. This is intended to capture the intuitive idea that the only way for an algebraic algorithm to generate a new group element is to derive it via group operations from known group elements.

We note a number of unsatisfactory aspects of this definition:

1. The definition does not constrain algorithms that do not output group elements. In particular, for the discrete-logarithm game the class of algebraic algorithms is the class of all algorithms. Thus, the AGM is useless for analyzing games where the algorithm's output is not a group element.

$\underline{\mathsf{A}(1)}$
$r_1, r_2 \leftarrow \mathbb{Z}_p$
$s \leftarrow r_1 \cdot r_2 \bmod p$
Output (s, s)

Figure 3: Algorithm A with respect to the identity encoding id.

2. The formalization considers some algorithms to be algebraic even though they may not match one's intuition regarding what operations an algebraic algorithm should be allowed to perform. For example, consider algorithm A in Figure 3 with respect to the identity encoding $\sigma = \mathsf{id}$. This algorithm samples two group elements r_1, r_2 and then *multiplies* them modulo p. The group operation here, however, is *addition* modulo p. Nevertheless, A is able to output a representation of the resulting group element s with respect to its base set $\{1\}$. More generally, whenever the encoding is such that the discrete-logarithm problem can be solved efficiently relative to that encoding, any algorithm can be made algebraic by simply computing a representation of any group elements it outputs.

3. Perhaps more problematic is that, once a particular encoding σ is fixed, it is not immediately well-defined what it means for an algorithm to "be provided with a group element as input" or to "output a group element." To get a sense of the problem, consider a game involving an oracle that, on input i, returns the ith bit of $\sigma(x)$. At no point in time does an algorithm in that game ever receive a group element from an oracle; nevertheless, it is clearly trivial to construct an algorithm that outputs the

group element $\sigma(x)$. Fuchsbauer et al. attempt to address this issue by requiring that "other elements" (i.e., non-group elements) "must not depend on any group elements," but it is not clear how such a requirement can be formalized.

It seems intuitive, and one would like to claim, that algebraic algorithms are at least as strong as generic algorithms, in the sense that for any game $\mathbf{G}$, any generic algorithm A with $\epsilon = \mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}}$, and any encoding σ, it is possible to construct an algebraic algorithm A_σ achieving the same success probability by simply simulating the encoding oracles for A and keeping track of the representations of any group elements generated during the execution of A. As already noted by Fuchsbauer et al., however, thus is not necessarily true (at least for Shoup's version of the GGM). Specifically, in Shoup's GGM it may be possible to obliviously sample group elements (i.e., without knowledge of their discrete logarithm), something that is ruled out by definition in the AGM.

We show in Section 4, in the context of reductions, that it is also not the case that all generic algorithms can be made algebraic.

Although Fuchsbauer et al. conjecture that any Maurer-generic algorithm can be made algebraic, we are not aware of a proof of that conjecture.

Generic reductions for algebraic adversaries. Fuchsbauer et al. [FKL18] consider generic reductions for algebraic adversaries; we map their definition to our syntax. For games $\mathbf{G}$, $\mathbf{H}$, write $\mathbf{H} \xRightarrow[\mathsf{alg}]{(\Delta_t,\Delta_\epsilon)} \mathbf{G}$ if there is a generic reduction R such that for all algebraic algorithms A and encodings σ, algorithm $\mathsf{B} := \mathsf{R}^{\mathsf{A}}$ satisfies

$$\mathbf{Succ}^{\mathsf{B}}_{\mathbf{H}_\sigma} \geq \frac{1}{\Delta_\epsilon} \cdot \mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}_\sigma}, \quad \mathbf{Time}^{\mathsf{B}}_{\mathbf{H}_\sigma} \leq \Delta_t \cdot \mathbf{Time}^{\mathsf{A}}_{\mathbf{G}_\sigma}. \tag{1}$$

The reduction is deliberately restricted to be generic (rather than algebraic) so that, as explained by Fuchsbauer et al., if A is algebraic then B will be algebraic, and if A is generic then B will be generic. We remark that the above notion seems to be useful only for Shoup-generic reductions; it is not clear how a Maurer-generic reduction would be able to provide A with encodings of group elements that A expects.

We observe several technical issues with the above definition:

- It is not true that when R is generic and A is algebraic, the composed algorithm $\mathsf{B} = \mathsf{R}^{\mathsf{A}}$ is algebraic. Indeed, a simple counterexample is a generic algorithm R that obliviously samples a group element and outputs it.

- Even if (1) holds for all algebraic algorithms A, it is not clear whether it holds for all generic algorithms A. Again, this is because a generic algorithm is not necessarily algebraic (nor is it necessarily possible to construct an algebraic algorithm with the same behavior).

4 A Counterexample

In this section, we give an example showing that a generic reduction in the AGM does not imply a reduction in the GGM. Concretely, we show two games $\mathbf{G}$ and $\mathbf{H}$ such that: (1) there

is a Shoup-generic reduction from $\mathbf{H}$ to $\mathbf{G}$; (2) $\mathbf{H}$ is hard for Shoup-generic algorithms; but (3) $\mathbf{G}$ is easy for Shoup-generic algorithms. Formally,

Theorem 4.1. *There are security games* $\mathbf{G}$ *and* $\mathbf{H}$ *such that*

- $\mathbf{H} \xRightarrow[\mathsf{alg}]{(2,1)} \mathbf{G}$;
- $\mathbf{H}$ *is* $(t, O(t^2/p))$*-hard with respect to Shoup-generic algorithms;*
- *There is a Shoup-generic algorithm* A *running in time* $O(\ell)$ *with* $\mathbf{Succ}^{\mathsf{A}}_{\mathbf{G}} = 1$.

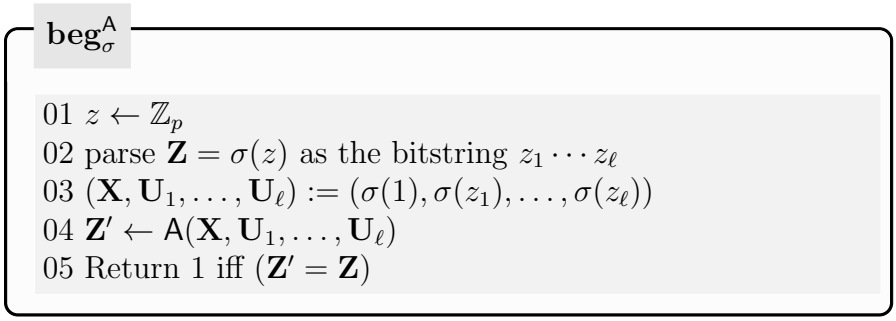

Figure 4: The binary encoding game.

Proof. Take $\mathbf{H}$ as the discrete-logarithm game from Figure 1. Security game $\mathbf{G}$ is one we introduce called the *binary encoding game* (**beg**); see Figure 4. Hardness of $\mathbf{H}$ for Shoup-generic algorithms was shown in [Sho97]. It is easy to see that there is a Shoup-generic algorithm A with $\mathbf{Succ}^{\mathsf{A}}_{\mathbf{beg}} = 1$: for each i, the algorithm sets $z'_i := 1$ iff $\mathbf{U}_i = \mathbf{X}$ and then outputs $\mathbf{Z}' := z'_1 \cdots z'_\ell$. Thus, it only remains to prove that $\mathbf{dlog} \xRightarrow[\mathsf{alg}]{(2,1)} \mathbf{beg}$.

Fix an encoding σ. Generic reduction R is given $(\mathbf{X}, \mathbf{Z}) := (\sigma(1), \sigma(z))$ as input along with oracle access to an algebraic algorithm A; it proceeds as follows:

1. Parse $\mathbf{Z}$ as the bitstring $z_1 \cdots z_\ell$. Set $z_0 := 1$.
2. Request $\mathbf{I} = \sigma(0)$ from the labeling oracle.
3. For $i = 1, \ldots, \ell$ do: if $z_i = 0$ then set $\mathbf{U}_i := \mathbf{I}$; else set $\mathbf{U}_i := \mathbf{X}$.
4. Run $\mathsf{A}(\mathbf{X}, \mathbf{U}_1, \ldots, \mathbf{U}_\ell)$ to obtain output $\mathbf{Z}'$ along with a representation $(x_0, x_1, \ldots, x_\ell)$ such that $\mathbf{Z}' = \mathbf{X}^{x_0} \cdot \mathbf{U}_1^{x_1} \cdots \mathbf{U}_\ell^{x_\ell}$.
5. Output $\sum_{i=0}^{\ell} z_i \cdot x_i \bmod p$.

We now analyze the behavior of R. Let A be an algebraic adversary with $\epsilon = \mathbf{Succ}^{\mathsf{A}}_{\mathbf{beg}_\sigma}$. Observe that when A is run as a subroutine by R in game $\mathbf{dlog}_\sigma$, the input provided to A is distributed identically as in $\mathbf{beg}_\sigma$. Moreover, whenever A succeeds it holds that (1) $\mathbf{Z}' = \mathbf{Z}$ and (2) $z = \sum z_i \cdot x_i \bmod p$. It follows that $\mathbf{Succ}^{\mathsf{R}^{\mathsf{A}}}_{\mathbf{dlog}_\sigma} = \epsilon$. This completes the proof. □

In light of our counterexample, we highlight where the proof of the result by Fuchsbauer et al. [FKL18, Lemma 2.2] fails. Note that the generic algorithm A with $\mathbf{Succ}^{\mathsf{A}}_{\mathbf{beg}} = 1$ that we construct as part of the proof cannot be converted to an algebraic algorithm. (More formally: the "trivial" attempt to convert A to an algebraic algorithm by monitoring its encoding oracles does not work, nor do we see another way to convert A to an algebraic algorithm. Moreover, as long as the discrete-logarithm problem is hard for some particular encoding σ, there is no efficient way to convert A into an algebraic algorithm with similar behavior relative to that encoding.)

5 Concluding Thoughts

Our work raises several issues related to the AGM. For starters, it is unclear whether the AGM is a meaningful class of algorithms to study; on the one hand because we are not aware of any (natural) algebraic algorithms that are not generic, and on the other hand because it is not clear whether the class of algebraic algorithms contains the class of generic algorithms. This may be related to the issue of whether the current formalization of the AGM adequately captures one's intuition about what "algebraic" algorithms can do, as well as whether it is possible to formally define what it means for certain objects not to "depend on" encodings of group elements. One argument in favor of the AGM is that it provides a meaningful way to analyze reductions; our work shows, however, that the main justification for studying reductions in the AGM does not hold in certain settings.

Our work raises several interesting directions for future work, including the question of developing other formalism for the algebraic group model, as well as formally resolving the question as to whether the class of algebraic algorithms strictly includes the class of Maurer-generic algorithms.

Acknowledgments

We thank Steven Galbraith for interesting discussions about the AGM and helpful comments on an earlier draft of this work.

References

[ABB+20] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In Daniele Micciancio and Thomas Ristenpart, editors, *Crypto 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, August 2020.

[ABK+21] Michel Abdalla, Manuel Barbosa, Jonathan Katz, Julian Loss, and Jiayu Xu. Algebraic adversaries in the universal composability framework. In Mehdi Ti-

bouchi and Huaxiong Wang, editors, *Asiacrypt 2021, Part III*, volume 13092 of *LNCS*, pages 311–341. Springer, Heidelberg, December 2021.

[BFL20] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *Crypto 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, Heidelberg, August 2020.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Eurocrypt 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

[BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *Eurocrypt '98*, volume 1403 of *LNCS*, pages 59–71. Springer, Heidelberg, May / June 1998.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Crypto 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *Eurocrypt 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg, May 2020.

[Gal12] Steven D Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[GT21] Ashrujit Ghoshal and Stefano Tessaro. Tight state-restoration soundness in the algebraic group model. In Tal Malkin and Chris Peikert, editors, *Crypto 2021, Part III*, volume 12827 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.

[KLX20] Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 390–413. Springer, Heidelberg, November 2020.

[KLX22] Julia Kastner, Julian Loss, and Jiayu Xu. On pairing-free blind signature schemes in the algebraic group model. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 468–497. Springer, 2022.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[MPZ20] Ueli Maurer, Christopher Portmann, and Jiamin Zhu. Unifying generic group models. Cryptology ePrint Archive, Report 2020/996, 2020. https://eprint.iacr.org/2020/996.

[MTT19] Taiga Mizuide, Atsushi Takayasu, and Tsuyoshi Takagi. Tight reductions for Diffie-Hellman variants in the algebraic group model. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 169–188. Springer, Heidelberg, March 2019.

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

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (Corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Pol78] John M Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *Asiacrypt 2005*, volume 3788 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2005.

[RS20] Lior Rotem and Gil Segev. Algebraic distinguishers: From discrete logarithms to decisional uber assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 366–389. Springer, Heidelberg, November 2020.

[RZ21] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *Crypto 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Eurocrypt '97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

[Zha22] Mark Zhandry. To label, or not to label (in generic groups). 2022. To appear at *Crypto 2022*. Full version available at https://eprint.iacr.org/2022/226.