An Analysis of the Algebraic Group Model[*]

Jonathan Katz
University of Maryland

Cong Zhang
Zhejiang University & University of Maryland

Hong-Sheng Zhou
Virginia Commonwealth University

June 19, 2022

Abstract

The algebraic group model (AGM), proposed by Fuchsbauer, Kiltz, and Loss (CRYPTO 2018) has received huge attention. One of the most appealing properties of the AGM, is that, the hardness of security games in the generic group model (GGM) can be transferred via a generic reduction in the AGM. More concretely, for any two security games, **G** and **H**, if there exists a generic reduction from **H** to **G** in the AGM, and **H** is hard in the GGM, then **G** is also hard in the GGM.

This work analyzes the definition of algebraic algorithms, the notion of generic reduction in AGM, and the relationship between the AGM and Shoup's GGM (Eurocrypt 1997). The following lists our contributions:

- the formal definition of algebraic algorithms does *not* capture the intuition of the algebraic algorithms;
- following the definition of generic algorithms in Shoup's GGM, the notion of generic reduction in the AGM is not well-defined;
- to make sure the notion of generic reduction in the AGM is well-defined, two strengthened versions of generic algorithms are introduced;
- hardness of security games in Shoup's GGM cannot be transferred via a generic reduction in the AGM;
- the AGM and Shoup's GGM are incomparable.

1 Introduction

Computational assumptions on groups, and standard model. We have seen many elegant cryptographic schemes and protocols using (cyclic) groups since the breakthrough results by Diffie and Hellman [DH76]. These schemes and protocols are *provably secure*: an adversarial algorithm against the scheme/protocol, can be (efficiently) transformed into another adversarial algorithm for solving a number theoretical hard problem on (cyclic) groups. Good examples of the hard problems include the Discrete Logarithm (DLOG) problem, the Computational Diffie Hellman (CDH)

[*]An early version of this result, with a different title, had been submitted to Eurocrypt 2022. We are grateful to the anonymous reviewers of Eurocrypt 2022 for their many valuable and constructive comments.

problem, and the Decisional Diffie Hellman (DDH) problem. For example, the well-known Diffie-Hellman key-exchange protocol, can be proven passively secure, under the Decisional Diffie-Hellman assumption [KL07]. We remark that, the adversary against the scheme/protocol is computationally bounded, and other than that, there is no any additional restriction; we call such adversarial attacking model as the *standard model*, or the plain model.

From standard model to generic group model. It is fundamentally important to understand the hardness of these hard problems. Certain relations between these hard problems have been demonstrated in the standard model by showing a reduction from one problem to another problem. For example, the DLOG problem is harder to solve than the DDH, and an adversarial algorithm solving the DLOG can be transformed into another adversarial algorithm to solve the DDH. Unfortunately, it is extremely difficult to establish lower bounds for hardness of the assumptions.

Interestingly, initiated by the seminal work of Nechaev [Nec94] and then formalized by Shoup [Sho97] and Maurer [Mau05], the *generic group model (GGM)* has been introduced for studying the security of many group-based intractable assumptions and cryptographic schemes. This line of research has been extended very recently by Maurer et al [MPZ20].

In the GGM, all algorithms are restricted to be "generic" in the sense that, algorithms are not allowed to exploit any structure of the group. Thus, "generic" algorithms can be applied in any group. Well-known examples of generic algorithms include the baby-step giant-step algorithm [PH78], and Pollard's rho algorithm [Pol78]. The GGM, although restricted, has been used for many years as the canonical tool to establish (certain level of) confidence for new intractable assumptions and cryptographic schemes. Note that, for many cryptographic groups (e.g., groups defined over some elliptic curves), the best known algorithms for assumptions on these groups, are generic.

Algebraic group model. In [BV98, PV05], a different type of restricted algorithms have been studied. There, algorithms are restricted to be "algebraic" in the sense that, algorithms given a concrete group description, are only allowed to carry out group operations on group elements. Moreover, to obtain a new group element, an algebraic algorithm must derive it via group operations from already known group elements.

This line of research has been further extended recently by Fuchsbauer, Kiltz, and Loss [FKL18], formalizing the algebraic group model (AGM), and then presenting multiple findings in the AGM. The AGM by Fuchsbauer et al [FKL18] is well received in cryptographic research community. Since then, many research results in the AGM have been published, and multiple AGM-related research projects have been carried out. We list a few here [MBKM19, MTT19, Lip19, GWC19, ABB$^+$20, AGK20, KLX20a, BDFG20, BFL20, CH20, CHM$^+$20, FPS20, GRWZ20, KLX20b, RS20, GT21, RZ21, ABK$^+$21].

In [FKL18], Fuchsbauer, Kiltz, and Loss, surprisingly find that, AGM and GGM are heavily correlated. They claim that, *the AGM is*, while stronger than the standard model, *weaker than the GGM*. Among many results, they prove that, in the GGM, there is an *alternative way to establish the lower bounds for new number theoretical hard problems based on the known lower bounds for a different hard problem*! For the sake of introducing our main question without touching subtle points, we present here a "fuzzy" version of their alternative approach; the formal version can be found in [FKL18, Lemma 2.2], and also in Section 5 of this writeup.

Let $\mathbf{G}$ and $\mathbf{H}$ be two security games. Assume the following conditions:

(1) $\mathbf{H} \Rightarrow \mathbf{G}$; that is, there exists a reduction R, so that an algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{G}}$ who can win in game $\mathbf{G}$, can be converted into another algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}} = \mathsf{R}^{\mathsf{A}_{\mathsf{alg}}^{\mathbf{G}}}$ who can win in game $\mathbf{H}$.

(2) game $\mathbf{H}$ is hard in the GGM; that is, there exists no generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{H}}$ who can win in game $\mathbf{H}$.

As in [FKL18], we expect to conclude that,

(3) game $\mathbf{G}$ is also hard in the GGM; that is, there exists no generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{G}}$ who can win in game $\mathbf{G}$.

We next follow a "canonical" proof strategy with the goal of raising our doubts, and we emphasize that this "canonical" proof strategy is not the one in [FKL18]. Let's resume our discussions. In order to reach the conclusion (3), a plausible proof strategy is as follows:

Step 1: We prove this by contradiction, and assume the negation of the conclusion (3); that is,

(3') game $\mathbf{G}$ is not hard in the GGM; that is, there exists a generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{G}}$ who can win in game $\mathbf{G}$.

Step 2: Based on the generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{G}}$ as specified in (3') above, ideally, we expect we can transform such generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{G}}$ into an algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{G}}$;

Step 3: Once the algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{G}}$ is defined, we can obtain an algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}}$, based on the condition (1) above, i.e., $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}} = \mathsf{R}^{\mathsf{A}_{\mathsf{alg}}^{\mathbf{G}}}$;

Step 4: Then, based on the algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}}$, ideally, we expect we can transform such algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}}$ into a generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{H}}$;

Step 5: Finally, we expect we complete the proof, since the existence of generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{H}}$ in Step 4, contradicts to condition (2) above.

In the above "canonical" proof strategy, while steps 1, 3, 5 are clear, we must be super cautious about steps 2 and 4. We next focus only on Step 4.

Recall that, the goal of the generic algorithm $\mathsf{A}_{\mathsf{gen}}^{\mathbf{H}}$ is to solve a hard problem (e.g., DDH) in the GGM. Existing techniques [Sho97, Mau05, MPZ20] essentially rely on the random encoding, and the lower bounds of hardness are established with respect to generic algorithms who do not rely on the concrete group description. This means, the generic algorithm $\mathsf{A}_{\mathsf{gen}}^{\mathbf{H}}$ should not use essentially the concrete group description, even such concrete group description is provided.

On the other hand, the generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{H}}$, in Step 4, has to use the algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}}$ as the subroutine. To effectively use the algebraic adversary $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}}$ as the subroutine, the generic adversary $\mathsf{A}_{\mathsf{gen}}^{\mathbf{H}}$ must use concrete group description to call the subroutine $\mathsf{A}_{\mathsf{alg}}^{\mathbf{H}}$. We are now in a dilemma!

Main questions. With the doubts, we thus have the following question:

Is it feasible to transform the hardness for one problem in the GGM to that for another problem, via a reduction in the AGM?

We further ask:

Is the AGM weaker than the GGM, or they are incomparable?

We remark that, it is important to answer the above questions since the AGM has served as the foundation for cryptanalysis in many research papers (see the list of citations on page 2).

1.1 Our results

We make a first attempt to answer the questions above and analyze the algebraic group model, including the definition of algebraic algorithms, the notion of generic reduction in the AGM, and the main technical lemma (Lemma 2.2 in [FKL18]).

1.1.1 The formal definition of algebraic algorithms does not capture the intuition.

We first analyze the definition of algebraic algorithms. In [FKL18], the intuition for defining algebraic algorithms has been explicitly stated as: "the only way for an algebraic algorithm to output a new group element is to derive it via group multiplication from known group elements."(Below, we say an algorithm performs properly if it follows this intuition.) To characterize this intuition, Fuchsbauer *et al.* propose the formal definition of algebraic algorithms as follows: briefly, an algorithm A is algebraic if when A outputs a group element, it *must* additionally output a linear representation to indicate how this group element has been derived.

Theorem 1.1 (Informal). *The formal definition of algebraic algorithms does not characterize the intuition of algebraic algorithms described in [FKL18].*

In a high level, this formal definition rules out all algorithms that do not output the linear representation, even those algorithms perform properly. Besides, *extracting the linear representation from a properly performed algorithm is hard in a black-box manner.* To the best of our knowledge, the only known way, discussed in [PV05], to extract the linear representation is via non-black-box access to the algorithm, which might cause inefficiency. On the other hand, requiring an algorithm to output the linear representation is insufficient to make sure that this algorithm performs properly. Specifically, in Section 3.1, we design two algorithms A_1 and A_2, such that

- following the intuition of algebraic algorithms in [FKL18], A_1 is algebraic but A_2 is not;
- following the formal definition of algebraic algorithms in [FKL18], A_2 is algebraic but A_1 is not.

The analysis above makes the formal definition of algebraic algorithms unjustified, as it does not capture the intuition of algebraic algorithms. However, as we mentioned before, in [FKL18], a technical lemma, i.e., their Lemma 2.2, has been introduced, which can justify their formal definition of algebraic algorithms; via this technical lemma, we can have an alternative way to establish the lower bounds for new hard problems in the GGM based on the known lower bounds for an existing hard problem (e.g., DLOG). The technical lemma tells that, for any two security games $\mathbf{H}$ and $\mathbf{G}$, if there is a generic reduction from $\mathbf{H}$ to $\mathbf{G}$ in the AGM, and $\mathbf{H}$ is hard in the GGM, then $\mathbf{G}$ is also hard in the GGM. For instance, if there is a reduction from DLOG to $\mathbf{G}$ in the AGM, then we have that $\mathbf{G}$ yields the same hardness as DLOG in the GGM. In other words, the hardness of security games in the GGM can be transferred via a reduction in the AGM. Again, in our perspective, this technical lemma serves as the justification of the definition of the algebraic algorithms and is essential to the AGM. We next focus on the analysis of this technical lemma.

1.1.2 The notion of generic reduction is not well-defined.

According to the statement of Lemma 2.2 in [FKL18], we see that the reduction R between two security games ($\mathbf{H}$ and $\mathbf{G}$), is required to be *generic*, but both $\mathbf{H}$ and $\mathbf{G}$ are working on concrete group description $\mathcal{G} = (\mathbb{G}, g, p)$, where $\mathbb{G}$ is a cyclic group with prime order p generated by g. We observe that, in Section 3.2, if following the definition of generic algorithms in Shoup's GGM [Sho97], then the notion of generic reduction R between $\mathbf{H}$ and $\mathbf{G}$ is *not well-defined*. For instance, let $\mathbf{H}$ and $\mathbf{G}$ to be DLOG and CDH, respectively. On one hand, being a reduction, R should be capable to "transfer" any algebraic algorithm A for CDH into an algorithm $\mathsf{B} := \mathsf{R}^{\mathsf{A}}$ for DLOG. On the other hand, being a generic algorithm, the challenge terms that R receives are $(\sigma(1), \sigma(x))$, where σ is a random encoding. Note that, when R executes A as a subroutine, R needs to send (g, g^x, g^y) to A as inputs. The main issue is that, R here only receives $\sigma(x)$ which is just a random string, and for an efficient reduction algorithm, it is impossible to output a valid challenging term g^x.[1]

1.1.3 The hardness of security games cannot be transferred via strengthened versions of generic algorithms.

To avoid to trivially invalidate the AGM as above, we then, in Section 4, propose two strengthened versions of generic algorithms, called Type-I generic and Type-II generic, to make sure that the definition of generic reduction between security games is well-defined. Intuitively, a Type-I generic algorithm is a generic algorithm in Shoup's GGM with additional power such that it can execute any algebraic algorithm (working on concrete group description) as a subroutine. Analogously, a Type-II generic algorithm is a generic algorithm in Maurer's GGM [Mau05] with additional power such that it can execute any algebraic algorithm (working on concrete group description) as a subroutine. With these two strengthened versions of generic algorithms, we analyze the main technical lemma in [FKL18].

Theorem 1.2 (Informal)**.** *The hardness of security games in Shoup's GGM cannot be transferred via a Type-I generic reduction in the AGM.*

In Section 5, we design a security game called BRDH, and illustrate a Type-I generic reduction from DLOG to BRDH. Note that DLOG is widely believed to be hard in the AGM, and given the reduction, BRDH can be also assumed to be hard in the AGM. However, we show that BRDH is easy in Shoup's GGM. This implies that, when applying the Type-I generic reduction, the main technical lemma does not hold with respect to Shoup's GGM.

Theorem 1.3 (Informal)**.** *The hardness of security games in Shoup's GGM cannot be transferred via a Type-II generic reduction in the AGM.*

In Section 6, we strengthen our results by considering that the reduction to be Type-II generic. We design a security game, called "variant of Dent's example" (VDLOG in Section 6.2). We then define another game $\mathbf{G}$ such that the adversary wins $\mathbf{G}$ if the adversary outputs $\perp$. Trivial to note that $\mathbf{G}$ is easy in both Shoup's GGM and the AGM. We then illustrate a Type-II generic reduction from VDLOG to $\mathbf{G}$ in the AGM, and prove that VDLOG is easy in the AGM but hard in Shoup's GGM. This implies that, even applying the Type-II generic reduction, the main technical lemma does not hold with respect to Shoup's GGM.

[1] It is easy to see that, this issue also exists when considering Maurer's GGM. In fact, generic algorithms in Maurer's GGM only receive abstract handle, which can be null; as a result, R cannot output a valid g^x.

Moreover, BRDH is easy in Shoup's GGM but believed to be hard in the AGM (assuming DLOG is hard in the AGM), which refers to the AGM is not weaker than Shoup's GGM. On the other hand, VDLOG is easy in the AGM but hard in Shoup's GGM, which means that Shoup's GGM is not weaker than the AGM either. Combing together, we obtain a bonus result.

Theorem 1.4 (Informal)**.** *The AGM and Shoup's GGM are incomparable.*

1.2 Comparing to the Concurrent Result

In a concurrent and independent work [Zha22], Zhandry studies the GGM and the AGM; among many results in [Zha22], Zhandry re-defines the AGM and illustrates an example to show the uninstantiability for the AGM. In this section, we discuss the main difference between our results and Zhandry's.

We first point out that there is an inconsistency between the AGM defined by Fuchsbauer *et al*. [FKL18] and the one re-defined by Zhandry [Zha22]. Specifically, the AGM re-defined by Zhandry is restricted to the Maurer's GGM; as a result the AGM inherits the limitations of the Maurer's GGM[2]. However, the AGM in [FKL18] does *not* suffer from this restriction. For example, Fuchsbauer *et al*. design a CCA1-secure ElGamal encryption with a very short ciphertext, while Zhandry gives evidence that, in Maurer's GGM, any CPA-secure encryption must have a large ciphertext[3]. We stress that, in this work, we analyze the AGM as the model defined in [FKL18] (rather than the one in [Zha22]).

Comparing to the results in [Zha22], our main result is that, we provide evidences to show that the technical lemma, i.e., Lemma 2.2 of [FKL18] does not hold, with respect to Shoup's GGM. Note that, this technical lemma has not been analyzed in [Zha22]. We remark that, without this technical lemma, it is hard to justify the security of the reduction in the AGM and many findings in [FKL18] will become invalid. In this work, our goal is to provide evidences to show that the technical lemma does not hold; more concretely, we design two examples and illustrate that the hardness of security games in Shoup's GGM cannot be transferred via a reduction in the AGM. Our paper also concludes that the AGM and Shoup's GGM are incomparable as a bonus result, which is also independently identified by Zhandry [Zha22]. As pointed by Zhandry [Zha22], our examples only make sense in Shoup's GGM; we admit that those examples can be trivially ruled out if we restrict that, the AGM only works on the security games that are captured by Maurer's GGM.

However, ruling out our examples is not sufficient to claim the technical lemma is provably sound. In fact, to the best of our knowledge, even with the restrictions proposed by Zhandry, it is still unclear how to prove this technical lemma, if we apply the reduction in the AGM in the black-box manner[4].

1.3 Discussions

In light of our analysis, we now briefly discuss possible research directions.

[2]See the elaboration on page 5 of [Zha22], "... the model inherits the limitations of the TS/Maurer model. ..."

[3]See the elaboration on page 21 of [Zha22], "...any CPA-secure encryption scheme in the TS model, whose domain is bit strings (as opposed to group elements) must have a quite large ciphertexts. In particular, the number of group elements in the ciphertext must approximately equal the bit-length of the message. ..."

[4]If we apply the reduction in the AGM in a non-black-box manner, then more precise and cautious treatments are needed. We emphasize that, even sticking to the non-black-box methodology, it is still unknown how to prove the technical lemma.

It will be great if we can show the hardness of security games in Shoup's GGM can be transferred via a reduction in the AGM, *even with respect to some reasonable restrictions.* We point out that the two examples we design are very artificial, and they are not the security games/assumptions that we are generally interested in at all. An interesting open problem is to characterize a family of security games such that

- this family covers many security games we are interested in, e.g., DLOG, CDH, DDH;
- for any two games belonging to this family, the hardness of these two games in Shoup's GGM can be transferred via a reduction in the AGM.

Organization. In Section 2, we describe the preliminaries, including security games and reductions, algebraic algorithms, generic algorithms and evasive relation. In Section 3, we illustrate the analysis on several notions in [FKL18], including the definition of algebraic algorithms, the notion of generic reduction in AGM. We then, in Section 4, propose two strengthened definitions of generic algorithms, called Type-I generic and Type-II generic, to make sure the notion of generic reduction in AGM is well-defined. In Section 5, we present an example to show that the hardness in Shoup's GGM cannot be transferred via a Type-I generic reduction in the AGM. We then strengthen our results by introducing another example in Section 6, to show that the hardness in Shoup's GGM cannot be transferred via a Type-II generic reduction in the AGM.

2 Preliminaries

In this section, we provide the preliminaries. We note that, the analysis of the remaining of the paper relies on the ideas in (the full version of) Fuchsbauer et al's paper [FKL17], and also the ideas in Shoup's paper [Sho97]. To make our presentation simple and to reduce the confusion, we directly borrow many notations, and definitions from these two papers.

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

Security games. We use a variant of (code-based) security games [BR04]. In game $\mathbf{G}_{par}$ (defined relative to a set of parameters par), an adversary A interacts with a challenger that answers oracle queries issued by the adversary A. It has a main procedure and (possibly zero) oracle procedures which describe how oracle queries are answered. We denote the output of a game $\mathbf{G}_{par}$ between a challenger and an adversary A via $\mathbf{G}^{\mathsf{A}}_{par}$. The adversary A is said to win if $\mathbf{G}^{\mathsf{A}}_{par} = 1$. See Figure 1 for the example of Discrete Logarithm Game $\mathbf{dlog}^{\mathsf{A}}_{\mathcal{G}}$, where par is set to be a group $\mathcal{G} = (\mathbb{G}, g, p)$. We define the advantage of A in $\mathbf{G}_{par}$ as $\mathbf{Adv}^{\mathbf{G}}_{par,\mathsf{A}} := \Pr[\mathbf{G}^{\mathsf{A}}_{par} = 1]$ and the running time of $\mathbf{G}^{\mathsf{A}}_{par}$ as $\mathbf{Time}^{\mathbf{G}}_{par,\mathsf{A}}$.

$\mathbf{dlog}^{\mathsf{A}}_{\mathcal{G}}$

```
00 $x \twoheadleftarrow \mathbb{Z}_p$
01 $\mathbf{X} := g^x$
02 $z \twoheadleftarrow \mathsf{A}(\mathbf{X})$
03 Return $(z = x)$
```

Figure 1: Discrete Logarithm Game **dlog**, relative to group $\mathcal{G} = (\mathbb{G}, g, p)$ and adversary A.

Security reductions. Let $\mathbf{G}$, $\mathbf{H}$ be security games. We write $\mathbf{H}_{par} \xRightarrow{(\Delta_\epsilon,\Delta_t)} \mathbf{G}_{par}$ if there exists an algorithm R (called $(\Delta_\epsilon, \Delta_t)$-reduction) such that for all algorithms A, algorithm B defined as $\mathsf{B} := \mathsf{R}^{\mathsf{A}}$ satisfies

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

2.1 Generic algorithms

The notion of generic algorithms was first formally defined in [Sho97]. Let $\mathbb{Z}_p$ be the additive group of integers $\pmod p$, and let S be a set of bit strings, where the size $|S| \geq p$. Consider an encoding function σ, which is an injective map from $\mathbb{Z}_p$ into S. We now define generic algorithms.

Definition 2.1 (**Generic Algorithms in Shoup's GGM** [Sho97])**.** *Let $\mathbb{Z}_p$ be the additive group of integers $\bmod p$, and let S be a set of bit strings with size at least p. A probabilistic algorithm $\mathsf{A}_{\mathsf{gen}}$ is called* generic in Shoup's GGM, *if it behaves as follows: algorithm $\mathsf{A}_{\mathsf{gen}}$ may make two types of queries, the labeling queries and the addition queries, to an oracle $\mathcal{O}$. When making a labeling query, $\mathsf{A}_{\mathsf{gen}}$ specifies a value $x \in \mathbb{Z}_p$, and then the oracle $\mathcal{O}$ responds to this query with $\sigma(x)$. When $\mathsf{A}_{\mathsf{gen}}$ makes an addition query, it specifies two values $s_1, s_2 \in S$, the oracle $\mathcal{O}$ responds as follows: if $s_1 = \sigma(x_1)$ and $s_2 = \sigma(x_2)$ for some $x_1, x_2 \in \mathbb{Z}_p$, then $\mathcal{O}$ responds with $\sigma(x_1 + x_2)$; otherwise responds with $\perp$.*

The algorithm $\mathsf{A}_{\mathsf{gen}}$ is based on p and S. However, $\mathsf{A}_{\mathsf{gen}}$ is not based on encoding function σ; information about encoding function σ can be available to $\mathsf{A}_{\mathsf{gen}}$ through the oracle only. We count both the number of bit operations, and the number of group operations (i.e., oracle queries), when we measure the running time of a generic algorithm. In [Mau05], Maurer proposes an alternative generic group model, where the generic algorithms only receive abstract handles (or null). Formally,

Definition 2.2 (**Generic Algorithms in Maurer's GGM** [Mau05])**.** *Let $\mathbb{Z}_p$ be the additive group of integers mod p. A probabilistic algorithm $\mathsf{A}_{\mathsf{gen}}$ is called* generic in Maurer's GGM, *if it behaves as follows: algorithm $\mathsf{A}_{\mathsf{gen}}$ may make three types of queries, the labeling queries, the addition queries and equality test queries, to an oracle $\mathcal{O}$. The oracle keeps a table T which is initially empty. When making a labeling query, $\mathsf{A}_{\mathsf{gen}}$ specifies a value $x \in \mathbb{Z}_p$, and then the oracle $\mathcal{O}$ responds to this query with a handle h and stores the tuple (x, h) into T. Algorithm $\mathsf{A}_{\mathsf{gen}}$ is only allowed to performs addition query and equality test query with the handle. When $\mathsf{A}_{\mathsf{gen}}$ makes an addition query, it specifies two handles h_1, h_2, the oracle $\mathcal{O}$ responds as follows: if $\exists (x_1, h_1), (x_2, h_2) \in T$, then responds to this query with a handle h' and stores $(x_1 + x_2, h')$ into T, otherwise it responds with $\perp$. When making an equality test query, algorithm $\mathsf{A}_{\mathsf{gen}}$ specifies two handles h_1, h_2, the oracle $\mathcal{O}$ responds as follows: if $\exists (x_1, h_1), (x_2, h_2) \in T$ such that $x_1 = x_2$, then responds to the query with 1, otherwise responds with 0.*

According to the definitions above, we observe that the generic algorithms in Maurer's GGM is much more restrictive than the ones in Shoup's GGM. More specifically, the generic algorithms in Maurer's GGM are only allowed to perform addition and equality test queries with handles. In contrast, the generic algorithms in Shoup's GGM can make use of the representation of the group elements, e.g., outputting the first bit of $\sigma(1)$.

2.2 Algebraic algorithms

We consider algebraic security games $\mathbf{G}_{\mathcal{G}}$ for which we set *par* to a fixed group description $\mathcal{G} = (\mathbb{G}, g, p)$, where $\mathbb{G}$ is a cyclic group of prime order p generated by g. In algebraic security games, we syntactically distinguish between elements of group $\mathbb{G}$ (written in bold, uppercase letters, e.g., $\mathbf{A}$) and all other elements, which must not depend on any group elements.

We now define algebraic algorithms. Intuitively, the only way for an algebraic algorithm to output a new group element $\mathbf{Z}$ is to derive it via group multiplications from known group elements.

Definition 2.3 (Algebraic Algorithms)**.** *An algorithm $\mathsf{A}_{\mathsf{alg}}$ executed in an algebraic game $\mathbf{G}_{\mathcal{G}}$ is called* algebraic *if for all group elements $\mathbf{Z}$ that $\mathsf{A}_{\mathsf{alg}}$ outputs (i.e., the elements in bold uppercase letters), it additionally provides the representation of $\mathbf{Z}$ relative to all previously received group elements. That is, if $\vec{\mathbf{L}}$ is the list of group elements $\mathbf{L}_0, \ldots, \mathbf{L}_m \in \mathbb{G}$ that $\mathsf{A}_{\mathsf{alg}}$ has received so far (w.l.o.g. $\mathbf{L}_0 = g$), then $\mathsf{A}_{\mathsf{alg}}$ must also provide a vector $\vec{z}$ such that $\mathbf{Z} = \Pi_i \mathbf{L}_i^{z_i}$. We denote such an output as $[\mathbf{Z}]_{\vec{z}}$.*

2.3 Generic reductions between algebraic security games

Generic algorithms $\mathsf{A}_{\mathsf{gen}}$ are only allowed to use generic properties of group $\mathcal{G}$. Informally, an algorithm is generic if it works regardless of what group it is run in. This is usually modeled by giving an algorithm indirect access to group elements via abstract handles. It is straight-forward to translate all of our algebraic games into games that are syntactically compatible with generic algorithms accessing group elements only via abstract handles.

We say that winning algebraic game $\mathbf{G}_{\mathcal{G}}$ is (ϵ, t)*-hard in the generic group model* if for every generic algorithm $\mathsf{A}_{\mathsf{gen}}$ it holds that

$$\mathbf{Time}_{\mathcal{G},\mathsf{A}_{\mathsf{gen}}} \leq t \Longrightarrow \mathbf{Adv}_{\mathcal{G},\mathsf{A}_{\mathsf{gen}}} \leq \epsilon.$$

We remark that usually in the generic group model one considers group operations (i.e., oracle calls) instead of the running time. In our context it is more convenient to measure the running time instead, assuming every oracle call takes one unit time.

In Section 2.3 of the full version of Fuchsbauer et al's paper [FKL17], generic reductions between algebraic security games has been defined. Let $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ be two algebraic security games. We write $\mathbf{H}_{\mathcal{G}} \xRightarrow[\mathsf{alg}]{(\Delta_\epsilon, \Delta_t)} \mathbf{G}_{\mathcal{G}}$ if there exists a generic algorithm $\mathsf{R}_{\mathsf{gen}}$ (called generic $(\Delta_\epsilon, \Delta_t)$*-reduction*) such that for every algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$, algorithm $\mathsf{B}_{\mathsf{alg}}$ defined as $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ satisfies

$$\mathbf{Adv}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \geq \frac{1}{\Delta_\epsilon} \cdot \mathbf{Adv}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}, \qquad \mathbf{Time}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \leq \Delta_t \cdot \mathbf{Time}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}.$$

Note that we deliberately require reduction $\mathsf{R}_{\mathsf{gen}}$ to be generic. Hence, if $\mathsf{A}_{\mathsf{alg}}$ is algebraic, then $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ is algebraic; if $\mathsf{A}_{\mathsf{alg}}$ is generic, then $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ is generic. If one is only interested in algebraic adversaries, then it suffices to require reduction $\mathsf{R}_{\mathsf{gen}}$ to be algebraic. But in that case one can no longer infer that $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ is generic in case $\mathsf{A}_{\mathsf{alg}}$ is generic.

2.4 Evasive relations on groups

Based on the *evasive relationship* [CGH98], Dent defines *evasive relations on groups*, which will be used for constructing a counter example. The following is taken from Section 3 of [Den02].

Definition 2.4 (Evasive Group Relation). *A relation $R \subset G \times S$ is said to be an* evasive group relation *if for any* PPT *machine* A *we have*

$$\Pr[x \leftarrow \mathsf{A}^{\rho}(1^{\lambda}) \ : \ (x, \rho(x)) \in R] \leq \text{negl}(\lambda),$$

where the probability is taken uniformly over all choices for an encoding function $\rho : G \to S$ and the coins of A.

3 Analysis of some notions in FKL18

In this section, we review several important notions in [FKL18]. Note that, multiple definition issues have been identified. First, we review the definitions of algebraic algorithms, and find that the definition in [PV05] and that in [FKL18] are incomparable; moreover we observe that the formal definition of algebraic algorithms does not capture the intuition of algebraic algorithms described in [FKL18]. Then, we illustrate that, if following the definition of generic algorithms in Shoup's GGM, the definition of "generic reduction between algebraic games" is not well-defined. After that, we point out that some technical terms in [FKL18], e.g.,"depend on", are not explicitly defined.

3.1 FKL18 vs. PV05

Algebraic algorithms have been investigated in literature (e.g., [PV05,FKL18]). As mentioned at the beginning of this section, we find the evidence that previous definition of algebraic algorithms in [PV05] and the one in [FKL18] are incomparable. Besides, in [FKL18], Fuchsbauer *et al.* explicitly stated the same intuition for defining algebraic algorithms as that in a previous result [PV05] by Paillier and Vergnaud, their actual definition deviates from the intuition. We next provide a more careful elaboration. Specifically, we present two algorithms A_1 and A_2, illustrated in Figure 2 and 3, respectively; then we prove the following:

- according to the definition in [PV05] and the intuition of algebraic algorithms described in [FKL18], A_1 is an algebraic algorithm but A_2 is not;
- according to the definition in [FKL18], A_2 is an algebraic algorithm but A_1 is not.

```
$\mathsf{A}_1(g)$;
01 $g^2 \leftarrow g \cdot g$;
02 Return $g^2$.
```

Figure 2: Algorithm A_1, with respect to cyclic group $\mathcal{G}$, where g is the group generator, and "$\cdot$" is the binary group operator.

In [PV05], Paillier and Vergnaud define the algebraic algorithms as follows: "... a reduction algorithm A is algebraic with respect to a group $\mathcal{G}$ if A is limited to perform group operations on group elements."[5] It is easy to note that, if we follow this definition, A_1 is an algebraic algorithm

```
$\mathsf{A}_2(g)$;
01 Sample $r_1 \leftarrow \mathbb{Z}_p$ and $r_2 \leftarrow \mathbb{Z}_p$;
02 Compute $s_1 \leftarrow r_1 \bmod p$ and $s_2 \leftarrow r_2 \bmod p$;
03 $r \leftarrow s_1 \times s_2$;
04 $s \leftarrow r \bmod p$;
05 Return $(s, r)$.
```

Figure 3: Algorithm A_2, with respect to additive group $\mathbb{Z}_p$, where $g = 1 \bmod p$ and "$+ \bmod p$" is the binary group operator. Note that, "$s_1 \times s_2$" in line 03 means that s_1 multiplies s_2.

while A_2 is not: algorithm A_2 in Figure 3 performs *non*-group operation on group elements in line 03.

In [FKL18], the same intuition for defining algebraic algorithms has been explicitly stated: "the only way for an algebraic algorithm to output a new group element $\mathbf{Z}$ is to derive it via group multiplications from known group elements".[6] With the same reason, if we follow the intuition of defining the algebraic algorithms in [FKL18], we again see that, algorithm A_1 is an algebraic **but algorithm A_2 is not**.

However, if following the formal definition of algebraic algorithms in [FKL18], then surprisingly, we see the opposite! Now algorithm A_1 is not algebraic, as A_1 does not output the corresponding linear representation. In this case, we point out that the formal definition of algebraic algorithms in [FKL18] does not reflect the definition in literature (e.g., [PV05]) or the intuition the authors intend to capture. More concretely, an unjustified requirement for defining algebraic algorithms is introduced in the formal definition in [FKL18]: the algebraic algorithm must additionally output the linear representation. This requirement rules out any algorithm that does not output the linear representation, even it performs properly. Moreover, to the best of our knowledge, it is unknown how to compute the linear representation by accessing to the algorithm in a black box manner (see more elaboration in Remark 3.1).

Besides, we point out that, requiring outputting the linear representation is *insufficient* to guarantee that the algorithm performs properly. In our example illustrated in Figure 3, with respect to the additive group $\mathbb{Z}_p$, algorithm A_2 outputs (s, r), where s can be viewed as group element g^s and r can be viewed as the corresponding linear representation. Note that, following the definition in [FKL18], **algorithm A_2 is algebraic**.

Based on the context of [FKL18], we know that the intuition for defining algebraic algorithm as already quoted above: "the only way for an algebraic algorithm to output a new group element $\mathbf{Z}$ is to derive it via group multiplications from known group elements". However, the analysis above tells that, the formal definition of algebraic algorithms in [FKL18] is neither sufficient nor necessary to capture this intuition.

Remark 3.1. *One might argue that, the linear representation can be easily computed by having the code of A_1. This idea indeed has been proposed in [PV05], by having an extractor which takes*

[5]See page 6 in [PV05].
[6]See page 6 in [FKL18].

the code of A_1 *as inputs, and outputs the linear representation. However, we emphasize that, the extractor has* **non-black-box** *access to the algorithm, which might cause inefficiency. To the best of our knowledge, there is no efficient way to compute the corresponding linear representation in a* **black-box** *manner.*

3.2 Generic Reduction between Algebraic Games

In this section, we illustrate that the definition of generic reduction between algebraic games in [FKL18] is not well-defined. Specifically, following the definition of generic algorithms in Shoup's GGM [Sho97], we have that, when a generic algorithm $\mathsf{A}_{\mathsf{gen}}$, with respect to $(\mathbb{Z}_p, S, \sigma)$, executes, it may make two types of queries to a oracle $\mathcal{O}$, namely, the labeling query and addition query. Precisely, when making a labeling query, $\mathsf{A}_{\mathsf{gen}}$ specifies a value $x \in \mathbb{Z}_p$, and then the oracle $\mathcal{O}$ responds to this query with $\sigma(x)$. When $\mathsf{A}_{\mathsf{gen}}$ makes an addition query, it specifies two values $s_1, s_2 \in S$, the oracle $\mathcal{O}$ responds as follows: if $s_1 = \sigma(x_1)$ and $s_2 = \sigma(x_2)$ for some $x_1, x_2 \in \mathbb{Z}_p$, then $\mathcal{O}$ responds with $\sigma(x_1 + x_2)$; otherwise it responds with $\perp$.

Being a generic algorithm, $\mathsf{A}_{\mathsf{gen}}$'s execution should work for all possible encoding functions, and here we consider the encoding function σ to be a random function that maps from $\mathbb{Z}_p$ to S. Next, we study the definition of generic reduction between algebraic games, which is defined as follows.

We observe that, if following the definition of generic algorithms in Shoup's GGM, then the definition of generic reduction $\mathsf{R}_{\mathsf{gen}}$ is not well-defined. More concretely, let $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ to be $\mathbf{cdh}_{\mathcal{G}}$ and $\mathbf{dlog}_{\mathcal{G}}$, for instance. On one hand, being a reduction, $\mathsf{R}_{\mathsf{gen}}$ should be capable to "transfer" any algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$ for $\mathbf{cdh}_{\mathcal{G}}$ into an algorithm $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ for $\mathbf{dlog}_{\mathcal{G}}$. On the other hand, being a generic algorithm, the challenging terms for $\mathbf{dlog}_{\mathcal{G}}$ that $\mathsf{R}_{\mathsf{gen}}$ receives would be $(\sigma(1), \sigma(x))$, where $\sigma(1)$ and $\sigma(x)$ are just random strings in S. When $\mathsf{R}_{\mathsf{gen}}$ executes $\mathsf{A}_{\mathsf{alg}}$ as a subroutine, it needs to send $(\mathbf{X}, \mathbf{Y})$ to $\mathsf{A}_{\mathsf{alg}}$ as inputs, where $\mathbf{X} := g^x$ and $\mathbf{Y} := g^y$. The main issue is that, $\mathsf{R}_{\mathsf{gen}}$ here only receives $\sigma(x)$ which is just a random string, and for an efficient reduction algorithm, it is impossible to output a valid challenging term $\mathbf{X}$.

Let $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ be two algebraic security games. We write $\mathbf{H}_{\mathcal{G}} \xrightarrow[\mathsf{alg}]{(\Delta_\epsilon, \Delta_t)} \mathbf{G}_{\mathcal{G}}$ if there exists a generic algorithm $\mathsf{R}_{\mathsf{gen}}$ (called generic $(\Delta_\epsilon, \Delta_t)$-*reduction*) such that for every algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$, algorithm $\mathsf{B}_{\mathsf{alg}}$ defined as $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ satisfies

$$\mathbf{Adv}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \geq \frac{1}{\Delta_\epsilon} \cdot \mathbf{Adv}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}, \qquad \mathbf{Time}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \leq \Delta_t \cdot \mathbf{Time}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}.$$

Note that we deliberately require reduction $\mathsf{R}_{\mathsf{gen}}$ to be generic. Hence, if $\mathsf{A}_{\mathsf{alg}}$ is algebraic, then $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ is algebraic; if $\mathsf{A}_{\mathsf{alg}}$ is generic, then $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ is generic. If one is only interested in algebraic adversaries, then it suffices to require reduction $\mathsf{R}_{\mathsf{gen}}$ to be algebraic. But in that case one can no longer infer that $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ is generic in case $\mathsf{A}_{\mathsf{alg}}$ is generic.

One might argue that, if setting the encoding function σ to be a function that is induced by concrete group description $\mathcal{G} = (\mathbb{G}, g, p)$, i.e., $\sigma(i) = g^i$, then the issue above probably does not apply, because $\mathsf{R}_{\mathsf{gen}}$ can just forward $\sigma(x)$ as the challenge term $\mathbf{X}$. Unfortunately, even in this specific setting, $\mathsf{R}_{\mathsf{gen}}$ is still not well-defined. Concretely, we note that, when $\mathsf{R}_{\mathsf{gen}}$ executes $\mathsf{A}_{\mathsf{alg}}$ as a subroutine, it has to also provides $\mathcal{G}$ to $\mathsf{A}_{\mathsf{alg}}$ as input. However, $\mathsf{R}_{\mathsf{gen}}$, as a generic algorithm, *only* has access to σ; it is impossible $\mathsf{R}_{\mathsf{gen}}$ to obtain $\mathcal{G}$, which refers to that $\mathsf{R}_{\mathsf{gen}}$ cannot execute $\mathsf{A}_{\mathsf{alg}}$ as a subroutine.

To avoid to trivially invalidate the AGM, in Section 4, we propose two strengthened versions of generic algorithms, called Type-I generic algorithms and Type-II generic algorithms, to make sure that the notion of generic reduction between algebraic games is well-defined.

3.3 Discussions on some terms in FKL18

On the term "depend on". We quote a sentence from [FKL18] in the following gray box (see page 6 in [FKL18]):

In algebraic security games, we syntactically distinguish between elements of group $\mathbb{G}$ (written in bold, uppercase letters, e.g., $\mathbf{A}$) and all other elements, which must not depend on any group elements.

We point out that, the term "*depend on*" is not explicitly defined. In contrast to Zhandry's result [Zha22], we are not attempting to formalize this term in this work.

On the term "received". Similarly, we quote another sentence from [FKL18] in the following gray box (part of Definition 2.3):

...That is, if $\vec{\mathbf{L}}$ is the list of group elements $\mathbf{L}_0, \dots, \mathbf{L}_m \in \mathbb{G}$ that $\mathsf{A}_{\mathsf{alg}}$ has received so far (w.l.o.g. $\mathbf{L}_0 = g$), then $\mathsf{A}_{\mathsf{alg}}$ must also provide a vector $\vec{z}$ such that $\mathbf{Z} = \Pi_i \mathbf{L}_i^{z_i}$...

We point out that, the term "*received*" is also not explicitly defined in [FKL18], especially when the algebraic algorithms have oracle access (please see the Strong Diffie-Hellman assumption (SDH) in [FKL17] for a concrete example). Again, in this work, we are not going to formalize this term, but we stress the following statement. If the inputs of an algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$ only consist of group elements, e.g., $\mathbf{L}_0, \dots, \mathbf{L}_m \in \mathbb{G}$, then $\mathsf{A}_{\mathsf{alg}}$ treats $\mathbf{L}_0, \dots, \mathbf{L}_m$ as the received group elements.

4 Strengthened versions of the Generic Algorithms

In section 3.2, we illustrate that, the definition of the *generic reduction between algebraic games* in [FKL18] is not well-defined. To avoid to trivially invalidate the AGM, in this section, we define two strengthened versions of generic algorithms to make sure that a (generic) algorithm is capable to execute an algebraic algorithm as a subroutine. Intuitively, we set $\mathcal{G}$ to be public parameter and now the strengthened version of generic algorithm is allowed to forward the parameter $\mathcal{G}$ to its subroutine[7]. Moreover, we set the encoding function σ to be a function that is induced by $\mathcal{G}$, i.e., $\sigma(i) := g^i$.

4.1 Type-I generic algorithms

We here present the formal definition of Type-I generic algorithms. Intuitively, we say an algorithm A is a Type-I generic algorithm, with respect to $\mathcal{G}$, if A is a generic algorithm in Shoup's GGM such that 1) the encoding function σ is induced by $\mathcal{G}$; 2) A is allowed to forward $\mathcal{G}$ to its subroutine. Formally,

[7]Note that, the algorithm is allowed to forward the parameter $\mathcal{G}$ to its subroutine *only*, and the algorithm is not allowed to use $\mathcal{G}$ for any other purposes.

Definition 4.1 (**Type-I generic algorithms**). *Let $\mathbb{Z}_p$ be the additive group of integers $\pmod{p}$, let $\mathcal{G} = (\mathbf{G}, g, p)$ be a group description, where $\mathbf{G}$ is a cyclic group of prime order p generated by g. Let σ be an encoding function that is induce by $\mathcal{G}$, i.e., $\sigma(i) = g^i$. We say a probabilistic algorithm $\mathsf{A}_{\mathsf{gen}}$ is a Type-I generic algorithm with respect to $\mathcal{G}$ if it behaves as follows: algorithm $\mathsf{A}_{\mathsf{gen}}$ may make three types of queries, the labeling queries, the addition queries and the forward queries, to an oracle $\mathcal{O}$. When making a labeling query, $\mathsf{A}_{\mathsf{gen}}$ specifies a value $x \in \mathbb{Z}_p$, and then the oracle $\mathcal{O}$ responds to this query with $\sigma(x)$. When $\mathsf{A}_{\mathsf{gen}}$ makes an addition query, it specifies two values s_1, s_2, the oracle $\mathcal{O}$ responds as follows: if $s_1 = \sigma(x_1)$ and $s_2 = \sigma(x_2)$ for some $x_1, x_2 \in \mathbb{Z}_p$, then $\mathcal{O}$ responds with $\sigma(x_1 + x_2)$; otherwise it responds with $\perp$. When making a forward query, the oracle $\mathcal{O}$ responds with $\mathcal{G}$; algorithm $\mathsf{A}_{\mathsf{gen}}$ can only perform forward operation with $\mathcal{G}$.*

4.2 Type-II generic algorithms

Now, we present the formal definition of Type-II generic algorithms. According to the definition above, an algorithm is Type-I generic if it is a generic algorithm in Shoup's GGM along with additional capability of forwarding group description parameter $\mathcal{G}$ to its subroutines. Now, we consider the generic algorithms in Maurer's GGM with the same capability. In a high level, we say an algorithm A is a Type-II generic algorithm, with respect to $\mathcal{G}$, if A is a generic algorithm in Maurer's GGM such that 1) the abstract handle is induced by $\mathcal{G}$; 2) A is allowed to forward $\mathcal{G}$ to its subroutine. Formally,

Definition 4.2 (**Type-II generic algorithms**). *Let $\mathbb{Z}_p$ be the additive group of integers mod p, let $\mathcal{G} = (\mathbf{G}, g, p)$ be a group description, where $\mathbf{G}$ is a cyclic group of prime order p generated by g. We say a probabilistic algorithm $\mathsf{A}_{\mathsf{gen}}$ is a Type-II generic algorithm with respect to $\mathcal{G}$ if it behaves as follows: algorithm $\mathsf{A}_{\mathsf{gen}}$ may make four types of queries, the labeling queries, the addition queries, equality test query and the forward queries, to an oracle $\mathcal{O}$. The oracle keeps a table T which is initially empty. When making a labeling query, $\mathsf{A}_{\mathsf{gen}}$ specifies a value $x \in \mathbb{Z}_p$, and then the oracle $\mathcal{O}$ responds to this query with a handle $h := g^x$ and stores the tuple (x, h) into T. Algorithm $\mathsf{A}_{\mathsf{gen}}$ is only allowed to perform addition queries and equality test queries with the handle. When $\mathsf{A}_{\mathsf{gen}}$ makes an addition query, it specifies two handles h_1, h_2, the oracle $\mathcal{O}$ responds as follows: if $\exists (x_1, h_1), (x_2, h_2) \in T$, then responds to this query with a handle $h' := g^{x_1 + x_2}$ and stores $(x_1 + x_2, h')$ into T, otherwise it responds with $\perp$. When making an equality test query, algorithm $\mathsf{A}_{\mathsf{gen}}$ specifies two handles h_1, h_2, the oracle $\mathcal{O}$ responds as follows: if $\exists (x_1, h_1), (x_2, h_2) \in T$ such that $x_1 = x_2$, then responds to the query with 1, otherwise responds with 0. When making a forward query, the oracle $\mathcal{O}$ responds with $\mathcal{G}$; algorithm $\mathsf{A}_{\mathsf{gen}}$ can only perform forward operation with $\mathcal{G}$.*

Note that, both Type-I and Type-II generic algorithms, with respect to $\mathcal{G}$, are able to run an algebraic algorithm as a subroutine. Therefore by applying these two versions of generic algorithms, the notion of "generic reduction between algebraic algorithms" can be well-defined. However, in the following sections we prove that, by applying either Type-I or Type-II, the main technical lemma in [FKL18] is not sound, with respect to Shoup's GGM.

5 Example against the main technical lemma with Type-I generic reduction

In this section, we give evidence that if considering the generic reduction to be Type-I generic, then the main technical lemma in [FKL18] does not hold, with respect to Shoup's GGM. We first recall the lemma, i.e., Lemma 2.2 in [FKL18], and its proof in the grey box below.

In this section, we treat the reduction $\mathsf{R}_{\mathsf{gen}}$ to be Type-I generic and the GGM to be Shoup's GGM. We design two algebraic security games say $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ such that: 1) there is a Type-I generic reduction $\mathsf{R}_{\mathsf{gen}}$ from $\mathbf{H}_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$; 2) $\mathbf{H}_{\mathcal{G}}$ is hard in Shoup's GGM, but $\mathbf{G}_{\mathcal{G}}$ is easy in Shoup's GGM. More formally,

Lemma 2.2. *Let $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ be algebraic security games such that $\mathbf{H}_{\mathcal{G}} \xrightarrow[\mathsf{alg}]{(\Delta_\epsilon,\Delta_t)} \mathbf{G}_{\mathcal{G}}$ and winning $\mathbf{H}_{\mathcal{G}}$ is (ϵ, t)-hard in the GGM. Then, $\mathbf{G}_{\mathcal{G}}$ is $(\epsilon \cdot \Delta_\epsilon, t/\Delta_t)$-hard in the GGM.*

Proof. Let $\mathsf{A}_{\mathsf{gen}}$ be a generic algorithm playing in game $\mathbf{G}_{\mathcal{G}}$. Then by our premise there exists a generic algorithm $\mathsf{B}_{\mathsf{alg}} = \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ such that

$$\mathbf{Adv}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \geq 1/\Delta_\epsilon \cdot \mathbf{Adv}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}, \qquad \mathbf{Time}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \leq \Delta_t \cdot \mathbf{Time}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}.$$

Assume $\mathbf{Time}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}} \leq t/\Delta_t$; then $\mathbf{Time}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \leq \Delta_t \cdot \mathbf{Time}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}} \leq t$. Since winning $\mathbf{H}_{\mathcal{G}}$ is (ϵ, t)-hard in the GGM, it follows that

$$\epsilon \geq \mathbf{Adv}^{\mathbf{H}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \geq 1/\Delta_\epsilon \cdot \mathbf{Adv}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}$$

and thus $\epsilon \cdot \Delta_\epsilon \geq \mathbf{Adv}^{\mathbf{G}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}$, which proves that $\mathbf{G}_{\mathcal{G}}$ is $(\epsilon \cdot \Delta_\epsilon, t/\Delta_t)$-hard in the GGM.

Theorem 5.1. *There exist algebraic security games $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ such that*

- $\mathbf{H}_{\mathcal{G}} \xrightarrow[\mathsf{alg}]{(\Delta_\epsilon,\Delta_t)} \mathbf{G}_{\mathcal{G}}$ *with respect to a Type-I generic reduction $\mathsf{R}_{\mathsf{gen}}$;*
- $\mathbf{H}_{\mathcal{G}}$ *is (ϵ, t)-hard in Shoup's GGM;*
- $\mathbf{G}_{\mathcal{G}}$ *is not $(\epsilon \cdot \Delta_\epsilon, t/\Delta_t)$-hard in Shoup's GGM.*

5.1 Binary-Representation related Diffie-Hellman (BRDH)

In this section, we present the description of the two algebraic games and the corresponding Type-I generic reduction. We introduce an algebraic security game, called *Binary-Representation related Diffie-Hellman Assumption*, as described via game $\mathbf{br\text{-}dh}_{\mathcal{G}}$ in Figure 4. In the following, we prove that $\mathbf{dlog}_{\mathcal{G}}$ implies $\mathbf{br\text{-}dh}_{\mathcal{G}}$ in the AGM, with respect to a Type-I generic reduction.

$\mathbf{br\text{-}dh}^{\mathsf{A}}_{\mathcal{G}}$

```
00 $x, y \twoheadleftarrow \mathbb{Z}_p$;
01 $z := xy$;
02 $\mathbf{Z} := g^z$;
03 parse group element $\mathbf{Z}$ into its canonical binary representation $z_1||\ldots||z_L$;
04 $(\mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L) := (g^x, g^y, g^{2+z_1}, \ldots, g^{2i+z_i}, \ldots, g^{2L+z_L})$;
05 $\mathbf{Z} \twoheadleftarrow \mathsf{A}(\mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$;
06 Return $(\mathbf{Z} = g^{xy})$.
```

Figure 4: Binary-representation related Diffie-Hellman Game **br-dh** relative to $\mathcal{G}$ and adversary A. Here $L = \text{poly}(\lambda)$ is the (maximal) length of each group element.

5.2 Type-I generic reduction from DLOG to BRDH

Claim 5.2. **dlog** $\overset{(2,1)}{\underset{\mathsf{alg}}{\Longrightarrow}}$ **br-dh**.

Proof. Let $\mathsf{A}_{\mathsf{alg}}$ be an algebraic adversary executed in game $\mathbf{br\text{-}dh}_{\mathcal{G}}$; cf. Figure 5. Note that, the inputs of $\mathsf{A}_{\mathsf{alg}}$ is $(g, \mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$. According to the discussion in Section 3.3, the received group elements of $\mathsf{A}_{\mathsf{alg}}$ are $(g, \mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$. Therefore, when $\mathsf{A}_{\mathsf{alg}}$ returns a solution $\mathbf{Z}$, it must return a representation $\vec{v} = (b_1, b_2, b_3, a_1, \ldots, a_L) \in \mathbb{Z}_p^{3+L}$ such that

$$\mathbf{Z} = g^{b_1}\mathbf{X}^{b_2}\mathbf{Y}^{b_3}\mathbf{Z}_1^{a_1}\ldots\mathbf{Z}_L^{a_L}. \tag{1}$$

In the following, we illustrate the Type-I reduction $\mathsf{R}_{\mathsf{gen}}$, with respect to $\mathcal{G}$, from DLOG to BRDH such that

$$\mathbf{Adv}^{\mathbf{dlog}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \geq \frac{1}{2}\mathbf{Adv}^{\mathbf{br\text{-}dh}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}.$$

$\mathbf{br\text{-}dh}_{\mathcal{G}}^{\mathsf{A}_{\mathsf{alg}}}$

```
00 $x, y \twoheadleftarrow \mathbb{Z}_p$;
01 $z := xy$;
02 $\mathbf{Z} := g^z$;
03 parse group element $\mathbf{Z}$ into its canonical binary representation $z_1||\ldots||z_L$;
04 $(\mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L) := (g^x, g^y, g^{2+z_1}, \ldots, g^{2i+z_i}, \ldots, g^{2L+z_L})$;
05 $[\mathbf{Z}]_{\vec{v}} \twoheadleftarrow \mathsf{A}(\mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$;
06 Return $(\mathbf{Z} = g^{xy})$.
```

Figure 5: binary-representation related Diffie-Hellman Game **br-dh** relative to $\mathcal{G}$ and adversary A. Here $L = \text{poly}(\lambda)$ is the (maximal) length of each group element.

On input the challenge term $(g, \mathbf{X} := g^x)$, $\mathsf{R}_{\mathsf{gen}}$ works as follows:

- Step 1: sample $y \twoheadleftarrow \mathbb{Z}_p$; $b \twoheadleftarrow \{0, 1\}$;
- Step 2: make a labeling query with input$y$ and obtain output $\sigma(y) := g^y$;
- Step 3: make addition queries and computes $\mathbf{Z} := X^y$;
- Step 4: parse $\mathbf{Z}$ into its canonical binary representation $z_1||\cdots||z_L$;[8]
- Step 5: make L number of labeling queries with input $2i+z_i$ and obtain output $\mathbf{U}_i := \sigma(2i+z_i)$;
- Step 6: make a forward query and obtain $\mathcal{G}$;
- Step 7: if $b = 0$, then call $\mathsf{A}_{\mathsf{alg}}$ with inputs $(\mathcal{G}, \mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$, else call $\mathsf{A}_{\mathsf{alg}}$ with inputs $(\mathcal{G}, \mathbf{Y}, \mathbf{X}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$.

[8] $\mathsf{R}_{\mathsf{gen}}$ here is Type-I generic but not Type-II generic, because Type-II generic algorithms can only perform addition and equality test queries.

At the end of the procedure, $\mathsf{A}_{\mathsf{alg}}$ outputs a solution $\mathbf{Z} = g^{xy}$ associated with a representation $\vec{v} = (b_1, b_2, b_3, a_1, \ldots, a_L) \in \mathbb{Z}_p^{3+L}$. The reduction $\mathsf{R}_{\mathsf{gen}}$ makes a labeling query with input b_2 and obtains output $\sigma(b_2) := g_2^b$, and solves x as follows:

- Case 1: If the bit $b = 0$ and $\mathbf{W} \neq \mathbf{Y}$, then $\mathsf{R}_{\mathsf{gen}}$ returns $\frac{b_1 + b_3 y + \sum_{i=1}^{L} a_i(2i+z_i)}{y - b_2}$;
- Case 2: If the bit $b = 1$ and $\mathbf{W} = \mathbf{X}$, then $\mathsf{R}_{\mathsf{gen}}$ returns b_2;
- Case 3: Otherwise, returns $\perp$.

In case 1, Equation (1) is equivalent to the equation

$$xy \equiv_p b_1 + b_2 x + b_3 y + \sum_{i=1}^{L} a_i(2i + z_i),$$

which refers to $x \equiv_p \frac{b_1 + b_3 y + \sum_{i=1}^{L} a_i(2i+z_i)}{y - b_2}$.

In case 2, due to $\mathbf{W} = \mathbf{X}$, we immediately have that $x = b_2$. Hence the advantage that $\mathsf{B}_{\mathsf{alg}}$ wins DLOG is

$$\mathbf{Adv}^{\mathbf{dlog}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \geq \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 0) \wedge (\mathbf{W} \neq \mathbf{Y})] + \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 1) \wedge (\mathbf{W} = \mathbf{X})].$$

Note that the distributions of $(\mathcal{G}, \mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$ and $(\mathcal{G}, \mathbf{Y}, \mathbf{X}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$ are identical, which means the bit b is independent of $\mathsf{A}_{\mathsf{alg}}$'s view. Thus, it's apparent that,

$$\Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 0) \wedge (\mathbf{W} = \mathbf{Y})] = \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 1) \wedge (\mathbf{W} = \mathbf{X})].$$

Now, we can compute the advantage of $\mathsf{B}_{\mathsf{alg}}$ as:

$$\begin{aligned}
&\mathbf{Adv}^{\mathbf{dlog}}_{\mathcal{G},\mathsf{B}_{\mathsf{alg}}} \\
&\geq \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 0) \wedge (\mathbf{W} \neq \mathbf{Y})] + \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 1) \wedge (\mathbf{W} = \mathbf{X})] \\
&\geq \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 0) \wedge (\mathbf{W} \neq \mathbf{Y})] + \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 0) \wedge (\mathbf{W} = \mathbf{Y})] \\
&= \Pr[(\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}) \wedge (b = 0)] \\
&= \Pr[\mathsf{A}_{\mathsf{alg}}\ \mathit{wins}] \times \Pr[b = 0] \quad \text{//Here } b \text{ is independent of } \mathsf{A}_{\mathsf{alg}} \\
&= \frac{1}{2}\mathbf{Adv}^{\mathbf{br\text{-}dh}}_{\mathcal{G},\mathsf{A}_{\mathsf{alg}}}.
\end{aligned}$$

□

5.3 BRDH is easy in Shoup's GGM

It is trivial to note that BRDH is easy in Shoup's GGM. In fact, the i-th bit of $\sigma(xy)$ can be easily identified, i.e., the i-th bit of $\sigma(xy)$ is "0" if and only if $\mathbf{U}_i = \sigma(2i)$.

Note that BRDH is easy in Shoup's GGM but believed to be hard in the AGM (assuming DLOG is hard in the AGM), thus we have that, the AGM is not weaker than Shoup's GGM.

6 Example against the main technical lemma with Type-II generic reduction

In Section 5, we know that the main technical lemma in [FKL18] does not hold when considering the reduction as a Type-I generic algorithm. In this section, we strengthen our results by designing new algebraic games and proving that, this main technical lemma still does not hold with respect to Shoup's GGM, even considering the reduction as Type-II generic. Besides, our example indicates that Shoup's GGM is not weaker than the AGM either; combing the analysis in Section 5, we have that the AGM and Shoup's GGM are *incomparable*. For readability, we recall again the main technical lemma in the grey box below.

Lemma 2.2. *Let $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ be algebraic security games such that $\mathbf{H}_{\mathcal{G}} \xRightarrow[\mathsf{alg}]{(\Delta_\epsilon,\Delta_t)} \mathbf{G}_{\mathcal{G}}$ and winning $\mathbf{H}_{\mathcal{G}}$ is (ϵ, t)-hard in the GGM. Then, $\mathbf{G}_{\mathcal{G}}$ is $(\epsilon \cdot \Delta_\epsilon, t/\Delta_t)$-hard in the GGM.*

In this section, we treat the reduction as a Type-II generic algorithm and the GGM to be Shoup's GGM. We design two algebraic security games say $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ such that: 1) there exists a Type-II generic reduction $\mathsf{R}_{\mathsf{gen}}$ from $\mathbf{H}_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$; 2) $\mathbf{H}_{\mathcal{G}}$ is hard in Shoup's GGM, but $\mathbf{G}_{\mathcal{G}}$ is easy in Shoup's GGM. More formally,

Theorem 6.1. *There exist algebraic security games $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$ such that*

- $\mathbf{H}_{\mathcal{G}} \xRightarrow[\mathsf{alg}]{(\Delta_\epsilon,\Delta_t)} \mathbf{G}_{\mathcal{G}}$ *with respect to a Type-II generic reduction $\mathsf{R}_{\mathsf{gen}}$;*
- $\mathbf{H}_{\mathcal{G}}$ *is (ϵ, t)-hard in Shoup's GGM;*
- $\mathbf{G}_{\mathcal{G}}$ *is not $(\epsilon \cdot \Delta_\epsilon, t/\Delta_t)$-hard in Shoup's GGM.*

Here we give the intuition of our example. For game $\mathbf{G}_{\mathcal{G}}$, we set it as an easy game in both the AGM and Shoup's GGM, e.g., the adversary wins $\mathbf{G}_{\mathcal{G}}$ if the adversary outputs $\perp$. For game $\mathbf{H}_{\mathcal{G}}$, we set it to be easy in the AGM but hard in Shoup's GGM. Note that, both $\mathbf{H}_{\mathcal{G}}$ and $\mathbf{G}_{\mathcal{G}}$ are easy in the AGM, which indicates a reduction from $\mathbf{H}_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$, as the adversary for $\mathbf{H}_{\mathcal{G}}$ itself serves as the reduction. Besides, in our example, $\mathbf{H}_{\mathcal{G}}$ is hard in Shoup's GGM but $\mathbf{G}_{\mathcal{G}}$ is easy in Shoup's GGM.

Note that, building $\mathbf{G}_{\mathcal{G}}$ is trivial; the subtle part in our example is to design $\mathbf{H}_{\mathcal{G}}$ and give a *Type-II generic reduction*. In our example, $\mathbf{H}_{\mathcal{G}}$ is inspired by the the impossibility result in [Den02], where Dent introduces a modified version of the original DLOG problem, and prove that his modified DLOG problem is *easy* in the standard model but is *hard* in the Shoup's GGM. In the following, we first recall Dent's example, then present the description of $\mathbf{H}_{\mathcal{G}}$ and the corresponding Type-II generic reduction.

6.1 Dent's Oracle-associated DLOG

In [Den02], a slightly modified DLOG problem has been introduced. Let's use **o-dlog** to denote the modified DLOG. Note that in [Den02], this modified DLOG problem serves as a counter example to prove that the GGM is not sound; that is, **o-dlog** is *trivial* in the standard model but is *secure* in the GGM.

The modified DLOG **o-dlog** (see Figure 6) is very similar to the original DLOG **dlog** (see Figure 1), except that now the adversary A is allowed to have access to certain oracle, as defined below. We note that, the oracle is based on an encoding ρ from $\mathbb{Z}_p$ to a set $\mathbb{G}$, and an evasive group relation R; Please see Section 2.4 for the definition of evasive group relation.

The following is taken from Section 4 of [Den02]; we define oracle $\mathsf{O}_{\rho,R}$ as:

$$\mathsf{O}_{\rho,R}(y,\rho(x)) = \begin{cases} x & \text{if } (y,\rho(y)) \in R, \\ \perp & \text{otherwise} \end{cases} \tag{2}$$

o-dlog$_{\mathcal{G}}^{\mathsf{A}}$

```
00 $x \twoheadleftarrow \mathbb{Z}_p$;
01 $\mathbf{X} := g^x$;
02 $z \twoheadleftarrow \mathsf{A}^{\mathsf{O}_{\mathcal{G},R}(\cdot,\cdot)}(\mathbf{X})$;
03 Return $(z = x)$.
```

Figure 6: Dent's Modified Discrete Logarithm Game **o-dlog**, relative to encoding ρ, evasive relation R and adversary A. Here we treat $\mathcal{G}$ as the encoding function such that $\rho(i) := g^i$.

We now restate Theorem 2 in [Den02] as follows:

Theorem 6.2. *For any evasive relation R,* **o-dlog** *is $(O(\frac{t^2}{p}), t)$-hard in Shoup's GGM, with respect to random encoding.*

6.2 A variant of Oracle-associated DLOG (VDLOG)

In this section, we give the description of $\mathbf{H}_{\mathcal{G}}$, a variant of Dent's example, denoted as **v-dlog**$_{\mathcal{G}}$. This game **v-dlog** (see Figure 7) is very similar to **o-dlog**, except the interface of the oracle. Concretely, given an encoding ρ, an evasive group relation R and $x \in \mathbb{Z}_p$,we define the oracle $\mathsf{O}_{\rho,R,x}$ as:

$$\mathsf{O}_{\rho,R,x}(y) = \begin{cases} x & \text{if } (y,\rho(y)) \in R, \\ \perp & \text{otherwise} \end{cases} \tag{3}$$

Note that, the only difference between **o-dlog** and **v-dlog** is the interface of the oracle. In **o-dlog**, the adversary A makes a query to $\mathsf{O}_{\rho,R}$ with inputs $(y,\rho(x))$, while in **v-dlog**, when the adversary makes a query to $\mathsf{O}_{\rho,R,x}$, it only sends y to the oracle.

Applying the same proof in [Den02], we have that,

Theorem 6.3. *For any evasive relation R,* **v-dlog** *is hard in Shoup's GGM, with respect to random encoding σ.*

Proof. Obviously the oracle $\mathsf{O}_{\rho,R,x}$ does not affect A unless it is queried with a value y such that $(y,\sigma(y)) \in R$. Since R is a group evasive relation, this probability is negligible, hence we may ignore the oracle $\mathsf{O}_{\rho,R,x}$. Besides, without the help of the oracle, it's trivial that the probability A outputs x is negligible.

v-dlog$_{\mathcal{G}}^{\mathsf{A}}$

```
00 $x \twoheadleftarrow \mathbb{Z}_p$;
01 $\mathbf{X} := g^x$;
02 $z \twoheadleftarrow \mathsf{A}^{\mathsf{O}_{\mathcal{G},R,x}(\cdot)}(\mathbf{X})$;
03 Return $(z = x)$.
```

Figure 7: A variant of Oracle-associated DLOG **v-dlog**, relative to evasive relation R and adversary A. Here we treat $\mathcal{G}$ as the encoding function such that $\rho(i) := g^i$.

Formally we denote E as the event that oracle $\mathsf{O}_{\rho,R,x}$ is queried with y such that $(y, \sigma(y)) \in R$ and $\bar{E}$ as the complement of E. It's apparent that

$$\begin{aligned}\Pr[x \leftarrow \mathsf{A}^{\mathsf{O}_{\rho,R,x}}(\sigma(x))] &= \Pr[x \leftarrow \mathsf{A}^{\mathsf{O}_{\rho,R,x}}(\sigma(x))|E]\Pr[E] + \Pr[x \leftarrow \mathsf{A}^{\mathsf{O}_{\rho,R,x}}(\sigma(x))|\bar{E}]\Pr[\bar{E}] \\ &\leq \Pr[E] + \Pr[x \leftarrow \mathsf{A}^{\mathsf{O}_{\rho,R,x}}(\sigma(x))|\bar{E}] \leq \text{negl}.\end{aligned}$$

□

6.3 VDLOG is easy in the AGM

In this section, we prove that **v-dlog**$_{\mathcal{G}}$ is easy in the AGM.

Claim 6.4. **v-dlog**$_{\mathcal{G}}$ *is easy in the AGM.*

Proof. In Shoup's GGM, the encoding ρ is random, while in the AGM, the encoding ρ is defined by the group description $\mathcal{G}$ such that $\rho(i) := g^i$. We now define the evasive relation R to be

$$R = \{(1, g)\}. \tag{4}$$

The oracle $\mathsf{O}_{\rho,R,x}(\cdot)$ now becomes

$$\mathsf{O}_{\mathcal{G},R,x}(y) = \begin{cases} x & \text{if } (y, g^y) \in R, \\ \perp & \text{otherwise} \end{cases} \tag{5}$$

Trivial to see that, any adversary makes a query to $\mathsf{O}_{\mathcal{G},R,x}(\cdot)$ with "1" would obtain x, which breaks **v-dlog**$_{\mathcal{G}}$ immediately.

□

6.4 Type-II generic reduction from v-dlog$_{\mathcal{G}}$ to G$_{\mathcal{G}}$

In this section, we set $\mathbf{G}_{\mathcal{G}}$ to be the game that the adversary wins $\mathbf{G}_{\mathcal{G}}$ if the adversary outputs $\perp$; it is trivial to see that $\mathbf{G}_{\mathcal{G}}$ is easy in both the AGM and Shoup's GGM. According to Section 6.3, we know that **v-dlog**$_{\mathcal{G}}$ is also easy in the AGM, which refers to a reduction from **v-dlog**$_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$, as the adversary for **v-dlog**$_{\mathcal{G}}$ itself servers as the reduction. In the following, we show that this reduction is Type-II generic with respect to $\mathcal{G}$. In fact, the adversary A for **v-dlog**$_{\mathcal{G}}$ works as follows:

- Step 1: take (g, g^x) as inputs;
- Step 2: make a query with input "1";
- Step 3: output whatever the oracle $\mathsf{O}_{\mathcal{G},R,x}(\cdot)$ returns.

Note that, the adversary A only makes a query to $\mathsf{O}_{\mathcal{G},R,x}(\cdot)$ with input "1" and returns the output of the oracle, which means it does not make any use of the handle. Therefore, the adversary A is Type-II generic, referring to that there is Type-II generic reduction from $\mathbf{v\text{-}dlog}_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$.

Remark 6.5. *The adversary for Dent's example is not Type-II generic, because when the adversary* A *makes a query to the oracle* $\mathsf{O}_{\rho,R}$*,* A *must send the challenge handle* $\mathbf{X}$ *to the oracle. According the definition in Section 4, any Type-II generic algorithm can only perform addition queries and equality test queries with handles. That is in fact the main reason why we introduce the variant of Dent's example.*

6.5 Discussions

First, we note that **v-dlog** is hard in Shoup's GGM but easy in the AGM, which refers to that the Shoup's GGM is not weaker than the AGM. Combing the analysis in Section 5, we conclude that the AGM and Shoup's GGM are incomparable.

Second, we note that, even in the standard model, our example still holds. Concretely, **v-dlog** is also easy in the standard model and the adversary against **v-dlog** in the standard model is also Type-II generic, as it does not make any use of the handle. Therefore, even replacing the AGM with the standard model, the main technical lemma does not hold, with respect to Shoup's GGM.

Third, we observe that both of our examples (i.e.,**br-dh** and **v-dlog**) cannot be characterized by Maurer's GGM. As a result, our examples *cannot* serve as evidence that "main technical lemma does not hold, with respect to Maurer's GGM".

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. Cryptology ePrint Archive, Report 2021/1218, 2021. https://ia.cr/2021/1218.

[AGK20] Benedikt Auerbach, Federico Giacon, and Eike Kiltz. Everybody's a target: Scalability in public-key encryption. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 475–506. Springer, Heidelberg, May 2020.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081, 2020. https://eprint.iacr.org/2020/081.

[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.

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. https://eprint.iacr.org/2004/331.

[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.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CH20] Geoffroy Couteau and Dominik Hartmann. Shorter non-interactive zero-knowledge arguments and ZAPs for algebraic languages. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 768–798. Springer, Heidelberg, August 2020.

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[Den02] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 100–109. Springer, Heidelberg, December 2002.

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

[FKL17] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. Cryptology ePrint Archive, Report 2017/620, 2017. https://ia.cr/2017/620.

[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.

[GRWZ20] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2007–2023. ACM Press, November 2020.

[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.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[KLX20a] Julia Kastner, Julian Loss, and Jiayu Xu. On pairing-free blind signature schemes in the algebraic group model. Cryptology ePrint Archive, Report 2020/1071, 2020. https://ia.cr/2020/1071.

[KLX20b] 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.

[Lip19] Helger Lipmaa. Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, Report 2019/612, 2019. https://ia.cr/2019/612.

[Mau05] Ueli Maurer. Abstract models of computation in cryptography. In *IMA International Conference on Cryptography and Coding*, pages 1–12. Springer, 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). Cryptology ePrint Archive, Report 2022/226, 2022. https://ia.cr/2022/226.