An Analysis of the Algebraic Group Model*

Jonathan Katz
University of Maryland

Cong Zhang
University of Maryland

Hong-Sheng Zhou
Virginia Commonwealth University

February 21, 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.

In this work, we analyze the relationship between the AGM and Shoup's GGM (Eurocrypt 1997) and give evidence that:

- 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) 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

*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. 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 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 4 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 $\mathsf{A}^{\mathbf{G}}_{\mathsf{alg}}$ who can win in game $\mathbf{G}$, can be converted into another algebraic adversary $\mathsf{A}^{\mathbf{H}}_{\mathsf{alg}} = \mathsf{R}^{\mathsf{A}^{\mathbf{G}}_{\mathsf{alg}}}$ who can win in game $\mathbf{H}$.
>
> (2) game $\mathbf{H}$ is hard in the GGM; that is, there exists no generic adversary $\mathsf{A}^{\mathbf{H}}_{\mathsf{gen}}$ 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 $\mathsf{A}^{\mathbf{G}}_{\mathsf{gen}}$ 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 $\mathsf{A}^{\mathbf{G}}_{\mathsf{gen}}$ who can win in game $\mathbf{G}$.

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

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

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

Step 5: Finally, we expect we complete the proof, since the existence of generic adversary $\mathsf{A}^{\mathbf{H}}_{\mathsf{gen}}$ 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 $\mathsf{A}^{\mathbf{H}}_{\mathsf{gen}}$ 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 $\mathsf{A}^{\mathbf{H}}_{\mathsf{gen}}$ should not use essentially the concrete group description, even such concrete group description is provided.

On the other hand, the generic adversary $\mathsf{A}^{\mathbf{H}}_{\mathsf{gen}}$, in Step 4, has to use the algebraic adversary $\mathsf{A}^{\mathbf{H}}_{\mathsf{alg}}$ as the subroutine. To effectively use the algebraic adversary $\mathsf{A}^{\mathbf{H}}_{\mathsf{alg}}$ as the subroutine, the generic adversary $\mathsf{A}^{\mathbf{H}}_{\mathsf{gen}}$ must use concrete group description to call the subroutine $\mathsf{A}^{\mathbf{H}}_{\mathsf{alg}}$. 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 of this writeup).

1.1 Our results

We make a first attempt to answer the questions above and analyze the relationship between the AGM and Shoup's GGM. Based on our analysis, we present the following results:

Theorem 1.1 (Informal)**.** *The hardness of security games in Shoup's GGM cannot be transferred via a generic reduction in the AGM.*

Theorem 1.2 (Informal)**.** *The AGM and Shoup's GGM are incomparable.*

In this work, we first focus on the generic algorithms in Shoup's GGM [Sho97], and we say an algorithm is Shoup-generic if it is generic in Shoup's GGM. To show that the hardness of games in Shoup's GGM cannot be transferred via a generic reduction in the AGM, we first consider the reduction is Shoup-generic. Before the analysis, in Section 3, we present some discussions on terms in [FKL17]; we believe these terms are not explicitly defined. Then, based on the discussions, we design two games, called Oracle-associated Diffie-Hellman (ODH in Section 4.1) and Binary-Representation related Diffie-Hellman (BRDH in Section 4.2).

In Section 4, we build a Shoup-generic reduction from DLOG to ODH (and BRDH) in the AGM. Note that DLOG is widely believed to be hard in the AGM, and due to the (Shoup-generic) reduction, ODH and BRDH can be assumed to be hard in the AGM. However, we prove that both ODH and BRDH are easy in Shoup's GGM[1]. This implies that the hardness of security games in Shoup's GGM cannot be transferred via a Shoup-generic reduction in the AGM.

We then turn attention to the generic algorithms in Maurer's GGM [Mau05]. Analogously, we say an algorithm is Maurer-generic if it is generic in Maurer's GGM. Note that, an algorithm is Maurer-generic implies that it is Shoup-generic [MPZ20], while the opposite direction does not hold. In fact, there exist algorithms (see the toy example in Section 2.1) that are Shoup-generic but not Maurer-generic.

In Section 5, we strengthen the results by considering that the reduction is Maurer-generic. We design a security game, called "variant of Dent's example" (VDLOG in Section 5.2). We then define another game $\mathbf{G}$ such that the adversary wins $\mathbf{G}$ if the adversary outputs its inputs. Trivial to note that $\mathbf{G}$ is easy in both Shoup's GGM and the AGM. We then build a Maurer-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. If the hardness of games in Shoup's GGM can be transferred via this reduction, then $\mathbf{G}$ must be hard in Shoup's GGM. Therefore, the hardness of security games in Shoup's GGM cannot be transferred via a Maurer-generic reduction in the AGM.

Moreover, ODH and BRDH are 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. 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, the AGM and Shoup's GGM are incomparable.

1.2 Discussions

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

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 three examples we present are very artificial, and they are not the security games/assumptions

[1]The hardness of ODH is super-poly but sub-exponential.

that we are generally interested in. 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, and generic algorithms. Before the analysis, in Section 3, we illustrate some discussion on several terms in [FKL17]. In Section 4, we present two examples to show that the hardness in Shoup's GGM cannot be transferred via a Shoup-generic reduction in the AGM. We then strengthen our results by introducing another example in Section 5, to show that the hardness in Shoup's GGM cannot be transferred via a Maurer-generic reduction in the AGM. In Appendix A, we describe the preliminary about the evasive relations on groups.

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 $s \leftarrow S$ 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 $y \leftarrow \mathsf{A}(x_1, \ldots, x_n)$. If A has access to an algorithm B (via oracle access) during its execution, we write $y \leftarrow \mathsf{A}^{\mathsf{B}}(x_1, \ldots, x_n)$.

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

Figure 1: Discrete Logarithm Game $\mathbf{dlog}$, 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

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)**.** *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*, 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 it 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. In Maurer's GGM, the generic algorithms only receive abstract handles (or null); therefore the generic algorithms in Maurer's GGM is not allowed to make use of the representation of group elements. In contrast, the generic algorithms in Shoup's GGM is allowed to make use of the representation of group elements. In the following, we present an example to illustrate this difference.

Let A be an algorithm that takes a group element $\sigma(x)$ as input and outputs b_1, where $b_1 || \cdots || b_L$ is the canonical binary representation of $\sigma(x)$. Trivial to note that, in Maurer's GGM, A is not generic as it makes use of b_1. However, in Shoup's GGM, A is generic.

In this work, we say an algorithm A is Maurer-generic, if A is generic in Maurer's GGM. Analogously, we say an algorithm A is Shoup-generic, if A is generic in Shoup's GGM.

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 2.2 (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 $\mathbf{H}_{\mathcal{G}} \xRightarrow{(\Delta_\epsilon,\Delta_t)}_{\mathsf{alg}} \mathbf{G}_{\mathcal{G}}$ 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.

3 Discussions on some terms in [FKL17]

We point out that some terms in [FKL17] are not explicitly defined. For the sake of better conveying our ideas, we here first present some discussions on these terms, before going to our technical sections (Section 4 and Section 5).

On the term "depend on". We quote a sentence from [FKL17] in the following gray box (see page 6 in [FKL17]):

> 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 stress that, the term "*depend on*" is not explicitly defined. We are not attempting to formalize this term as we believe it is hard to give a precise mathematical definition for describing the term "*depend on*". In this work, when we say an element is a non-group element, we mean that, this element is not identical to any group element.

In [FKL17], Fuchsbauer *et al.* state that, in the AGM, the algebraic algorithms must avoid the non-group elements which depend on some group elements and they call such non-group elements, *pathological*. Again, we are not attempting to formalize these notions. However, we want to clarify that, when a single bit (either "0" or "1") is treated as a non-group element, this non-group element is *not* pathological. Indeed, in [FKL17], the algebraic algorithms *are* allowed to have the oracle access, where the oracle may return a single bit for each query; please see the Strong Diffie-Hellman assumption (SDH) in [FKL17] for a concrete example.

On the term "received". Similarly, we quote another sentence from [FKL17] in the following gray box (part of Definition 2.2):

> ...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 $\mathbf{Z} = \Pi_i \mathbf{L}^{z_i}$...

Again, the term "*received*" is not explicitly defined in [FKL17]. For the received group elements, at least the following two cases should be considered:

- Case 1: the inputs[2] of algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$ only consist of group elements, e.g., $\mathbf{L}_0, \ldots, \mathbf{L}_m \in \mathbb{G}$;
- Case 2: the inputs of algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$ consist of group elements (e.g., $\mathbf{L}_0, \ldots, \mathbf{L}_m$), along with some bits, e.g., $s_1, \ldots, s_n \in \{0,1\}^n$. We note that, these bits are non-group elements; as discussed above, these bits are not pathological.

For the former case (Case 1), it is natural to treat $\mathbf{L}_0, \ldots, \mathbf{L}_m$ as the received group elements. For the latter case (Case 2), we argue that the term "received" is not explicitly defined. Specifically, there are two potential treatments for the latter case:

- Treatment (I): those non-group elements are ignored and $\mathbf{L}_0, \ldots, \mathbf{L}_m$ are treated as the received group elements;
- Treatment (II): additional received group elements can be obtained from those non-group elements $s_1, \ldots, s_n$.

While the Treatment (I) is clear, we argue that, it is hard to formalize the Treatment (II) with a proper justification. Below we provide our intuitive elaboration.

Let $\mathbf{L} \in \mathbb{G}$ be a new group element and $s_1 || \ldots || s_L$ be its binary representation, where L is the length of the group element $\mathbf{L}$. Multiple sub-cases can be considered when non-group elements can be used for deriving new group elements. Next, we focus on two of them (with the goal of illustrating our intuition):

- Subcase (i): the inputs of $\mathsf{A}_{\mathsf{alg}}$ consist of group elements, and a single non-group element $s_1 \in \{0,1\}$;
- Subcase (ii): the inputs of $\mathsf{A}_{\mathsf{alg}}$ consist of group elements, and ℓ number of non-group elements $s_1, \ldots, s_\ell$, where $\ell = L - \log^2(\lambda)$. (Note that, L is the length of the group element $\mathbf{L}$, as described above.)

For the first subcase (subcase (i)), it's natural to claim that, the algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$ cannot extract additional group elements from s_1; otherwise "s_1" is pathological. However, for the second subcase (subcase (ii)), we note that, the algebraic algorithm $\mathsf{A}_{\mathsf{alg}}$ receives all bits of $\mathbf{L}$ except the last $\log^2(\lambda)$ bits. Note that $2^{\log^2(\lambda)} = \lambda^{\log \lambda}$ is super-poly. As a result, it is natural to believe that, in subcase (ii), no efficient algebraic algorithm can extract additional group elements from those non-group elements, because it is hard for any efficient algebraic algorithm to guess the last $\log^2(\lambda)$ bits correctly with a noticeable probability.

Based on the discussions above, we can see for Treatment (II), it is non-trivial to deal with the non-group elements in the inputs of an algebraic algorithm. In the remaining of the paper, when the inputs of an algebraic algorithm consist of group elements, along with a single non-group element (as in subcase (i) above), or along with multiple non-group elements as in subcase (ii) above), we decide not to derive group elements from the non-group element(s) in the inputs.

[2]The inputs of an algebraic algorithm include both the initial inputs and elements received during execution, e.g., the response of the oracles.

4 Hardness of security games in Shoup's GGM cannot be transferred, via a Shoup-generic reduction in the AGM

In this section, we give evidence that hardness of security games in Shoup's GGM cannot be transferred via a Shoup-generic reduction in the AGM. We first recall the main lemma, i.e., their Lemma 2.2, in [FKL17] and its proof in the grey box below.

Lemma 2.2. *Let* $\mathbf{G}_{\mathcal{G}}$ *and* $\mathbf{H}_{\mathcal{G}}$ *be algebraic security games such that* $\mathbf{H}_{\mathcal{G}} \xrightarrow{(\Delta_\epsilon,\Delta_t)}_{\mathsf{alg}} \mathbf{G}_{\mathcal{G}}$ *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.

According to the proof, we note that the reduction algorithm $\mathsf{R}_{\mathsf{gen}}$ is generic. In this section, we consider the case that $\mathsf{R}_{\mathsf{gen}}$ is *Shoup-generic*. More concretely, we design two algebraic security games say $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$: there exists a Shoup-generic reduction $\mathsf{R}_{\mathsf{gen}}$ such that winning $\mathbf{H}_{\mathcal{G}}$ in the AGM can be reduced to winning $\mathbf{G}_{\mathcal{G}}$ in the AGM, and $\mathbf{H}_{\mathcal{G}}$ is hard in Shoup's GGM, but $\mathbf{G}_{\mathcal{G}}$ is not as hard as $\mathbf{H}_{\mathcal{G}}$ in Shoup's GGM. More formally,

Theorem 4.1. *There exist algebraic security games* $\mathbf{G}_{\mathcal{G}}$ *and* $\mathbf{H}_{\mathcal{G}}$ *such that*

- $\mathbf{H}_{\mathcal{G}} \xrightarrow{(\Delta_\epsilon,\Delta_t)}_{\mathsf{alg}} \mathbf{G}_{\mathcal{G}}$ *with respect to a Shoup-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.*

4.1 Example: Oracle-associated Diffie-Hellman (ODH)

In this section, we present our first example. We first introduce an algebraic security game, called *Oracle-associated Diffie-Hellman Assumption*, as described via game $\mathbf{o\text{-}dh}_{\mathcal{G}}$ in Figure 2. Next, in subsection 4.1.1 we show that $\mathbf{dlog}_{\mathcal{G}}$ implies $\mathbf{o\text{-}dh}_{\mathcal{G}}$ in the AGM, with respect to a Shoup-generic reduction.

4.1.1 Shoup-generic reduction from DLOG to ODH

Claim 4.2. $\mathbf{dlog} \xrightarrow{(2,1)}_{\mathsf{alg}} \mathbf{o\text{-}dh}$.

Proof. Let $\mathsf{A}_{\mathsf{alg}}$ be an algebraic adversary executed in game $\mathbf{o\text{-}dh}_{\mathcal{G}}$; cf. Figure 3. Note that, the inputs of $\mathsf{A}_{\mathsf{alg}}$ is $(g, \mathbf{X}, \mathbf{Y})$ and $(s_1, \ldots, s_\ell)$, where $s_1 || \ldots || s_L$ is the canonical binary representation of g^{xy}, and $\ell = L - \log^2(\lambda)$. According to the discussion in Section 3, the received group elements of $\mathsf{A}_{\mathsf{alg}}$ is $(g, \mathbf{X}, \mathbf{Y})$. Therefore, when $\mathsf{A}_{\mathsf{alg}}$ returns a solution $\mathbf{Z}$, it must return a representation $\vec{b} = (b_1, b_2, b_3) \in \mathbb{Z}_p^3$ such that

$$\mathbf{Z} = g^{b_1} \mathbf{X}^{b_2} \mathbf{Y}^{b_3} \tag{1}$$

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

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

$\underline{\mathsf{O}_{\mathcal{G},x,y}(\mathbf{I})}$;
04 If $\mathbf{I} = g^i$ and $i \leq \ell$;
Return the i-th bit of g^{xy};
05 Else return $\perp$.

Figure 2: Oracle-associated Diffie-Hellman Game **o-dh** relative to $\mathcal{G}$ and adversary A. Here $\ell = L - \log^2(\lambda)$, where $L = \mathrm{poly}(\lambda)$ is the (maximal) length of each group element; and oracle $\mathsf{O}_{\mathcal{G},x,y}(\cdot)$ is parameterized with the group description $\mathcal{G}$ and values x, y.

In the following, we construct a Shoup-generic reduction $\mathsf{R}_{\mathsf{gen}}$ that calls $\mathsf{A}_{\mathsf{alg}}$ exactly once such that for $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}^{\mathsf{A}_{\mathsf{alg}}}_{\mathsf{gen}}$ we have

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

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

00 $x, y \twoheadleftarrow \mathbb{Z}_p$;
01 $(\mathbf{X}, \mathbf{Y}) := (g^x, g^y)$;
02 $[\mathbf{Z}]_{\vec{b}} \twoheadleftarrow \mathsf{A}^{\mathsf{O}_{\mathcal{G},x,y}(\cdot)}_{\mathsf{alg}}(\mathbf{X}, \mathbf{Y})$;
03 Return $(\mathbf{Z} = g^{xy})$.

$\underline{\mathsf{O}_{\mathcal{G},x,y}([\mathbf{I}]_{\vec{c}})}$;
04 If $(\mathbf{I} = g^{c_1}\mathbf{X}^{c_2}\mathbf{Y}^{c_3})$ and $(\mathbf{I} = g^i \wedge i \leq \ell)$;
Return the i-th bit of g^{xy};
05 Else return $\perp$.

Figure 3: Algebraic adversary $\mathsf{A}_{\mathsf{alg}}$. Here $\ell = L - \log^2(\lambda)$, where $L = \mathrm{poly}(\lambda)$ is the (maximal) length of each group element; and oracle $\mathsf{O}_{\mathcal{G},x,y}(\cdot)$ is parameterized with the group description $\mathcal{G}$ and values x, y; and $\vec{c} = (c_1, c_2, c_3)$.

Next, we present the description of the Shoup-generic reduction algorithm $\mathsf{R}_{\mathsf{gen}}$. On input a challenging term $\mathbf{X} := g^x$, it first randomly samples $y \twoheadleftarrow \mathbb{Z}_p$ and a bit $b \twoheadleftarrow \{0, 1\}$ [3]. After that, $\mathsf{R}_{\mathsf{gen}}$ computes $\mathbf{Y} := g^y$ and $\mathbf{X}^y$. If $b = 0$, then $\mathsf{R}_{\mathsf{gen}}$ calls $\mathsf{A}_{\mathsf{alg}}$ with inputs $(\mathbf{X}, \mathbf{Y})$, and if $b = 1$, then $\mathsf{R}_{\mathsf{gen}}$ calls $\mathsf{A}_{\mathsf{alg}}$ with inputs $(\mathbf{Y}, \mathbf{X})$.

Next, $\mathsf{R}_{\mathsf{gen}}$ simulates the oracle $\mathsf{O}_{\mathcal{G},x,y}(\cdot)$ for $\mathsf{A}_{\mathsf{alg}}$ as follows: when $\mathsf{A}_{\mathsf{alg}}$ sends a query $[\mathbf{I}]_{\vec{c}}$, responds with the i-th bit of $\mathbf{X}^y$[4] if $(\mathbf{I} = g^{c_1}\mathbf{X}^{c_2}\mathbf{Y}^{c_3})$ and $(\mathbf{I} = g^i$ where $i \leq \ell)$, otherwise it returns $\perp$.

At the end of the procedure, $\mathsf{A}_{\mathsf{alg}}$ outputs a solution $\mathbf{Z} = g^{xy}$ associated with a representation $\vec{b} = (b_1, b_2, b_3) \in \mathbb{Z}_p^3$. The reduction $\mathsf{R}_{\mathsf{gen}}$ computes $\mathbf{W} = g^{b_2}$, and solves x as follows:

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

[3] To ensure $\mathsf{R}_{\mathsf{gen}}$ is a generic algorithm, we stress that y and b are independent of the description of g^x.
[4] We remark that these operations are allowed in Shoup's GGM, but are forbidden in Maurer's GGM.

In case 1, Equation (1) is equivalent to the equation $xy \equiv_p b_1 + b_2 x + b_3 y$, which means $x \equiv_p \frac{b_1+b_3 y}{y-b_2}$. 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

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

Moreover, we note that the bit b is independent of the $\mathsf{A}_{\mathsf{alg}}$'s view, which means that $\mathsf{A}_{\mathsf{alg}}$'s output is independent of b. Besides, when $b = 0$, $\mathsf{A}_{\mathsf{alg}}$ takes inputs $(\mathbf{X}, \mathbf{Y})$, and when $b = 1$, $\mathsf{A}_{\mathsf{alg}}$ takes inputs $(\mathbf{Y}, \mathbf{X})$, where we note that from $\mathsf{A}_{\mathsf{alg}}$'s view, $(\mathbf{X}, \mathbf{Y})$ and $(\mathbf{Y}, \mathbf{X})$ are identical distributed. Thus, it is apparent that,

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

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

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

□

4.1.2 Hardness of the ODH in Shoup's GGM

According to the existing hardness results for Discrete Logarithm problem in Shoup's GGM [Sho97], we have that $\mathbf{dlog}_{\mathcal{G}}$ is $(O(\frac{t^2}{p}), t)$-hard. By the reduction in Section 4.1.1, it suffices to prove that $\mathbf{o\text{-}dh}_{\mathcal{G}}$ is not $(O(\frac{t^2}{p}), t)$-hard in Shoup's GGM.

Claim 4.3. **o-dh** *is not* $(O(\frac{t^2}{p}), t)$*-hard in Shoup's GGM, with respect to random encoding.*

Proof. We here design a Shoup-generic attacker $\mathsf{A}_{\mathsf{gen}}$ that plays game $\mathbf{o\text{-}dh}_{\mathcal{G}}$ with advantage $\frac{1}{\lambda^{\log(\lambda)}}$.

On input challenging terms $(\sigma(1), \sigma(x), \sigma(y))$, $\mathsf{A}_{\mathsf{gen}}$ first makes ℓ labeling queries with $\{1, \ldots, \ell\}$ and obtains $\{\sigma(1), \ldots, \sigma(\ell)\}$. Then, for all $i \in \{1, \ldots, \ell\}$, $\mathsf{A}_{\mathsf{gen}}$ calls its oracle $\mathsf{O}_{\sigma,x,y}(\cdot)$ with $\sigma(i)$ and obtains a bit s_i. After that, $\mathsf{A}_{\mathsf{gen}}$ uniformly samples additional $\log^2(\lambda)$ bits: $s_{\ell+1}, \ldots, s_{\ell+\log^2(\lambda)} \twoheadleftarrow \{0,1\}^{\log^2(\lambda)}$. At the end, $\mathsf{A}_{\mathsf{gen}}$ outputs $s_1 || \cdots || s_{\ell+\log^2(\lambda)}$.

By definition, we know that, if $\mathsf{A}_{\mathsf{gen}}$ guesses the last $\log^2(\lambda)$ bits correctly, then $\mathsf{A}_{\mathsf{gen}}$ wins $\mathbf{o\text{-}dh}_{\mathcal{G}}$, which refers to

$$\mathbf{Adv}^{\mathbf{o\text{-}dh}}_{\mathcal{G},\mathsf{A}_{\mathsf{gen}}} \geq \frac{1}{2^{\log^2(\lambda)}} = \frac{1}{\lambda^{\log(\lambda)}} > \frac{\mathrm{poly}(\lambda)}{p} \geq O(\frac{t^2}{p}).$$

□

Remark 4.4. *Careful readers might note that, game* **o-dh** *cannot be modeled in Maurer's GGM, because generic adversaries only receive abstract handles (or null) in Maurer's GGM. However, we argue that,* **o-dh** *can be modeled in Shoup's GGM, by extending the notion of generic algorithms so as to include an additional oracle.*

4.2 Example: Binary-Representation related Diffie-Hellman (BRDH)

In this section, we present the second example. 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. By the description of $\mathbf{br\text{-}dh}_{\mathcal{G}}$, we note that the main difference between $\mathbf{o\text{-}dh}_{\mathcal{G}}$ and $\mathbf{br\text{-}dh}_{\mathcal{G}}$ is that, there is no oracle access in $\mathbf{br\text{-}dh}_{\mathcal{G}}$ and the inputs of $\mathbf{br\text{-}dh}_{\mathcal{G}}$ are all group elements. Now we show that $\mathbf{dlog}_{\mathcal{G}}$ implies $\mathbf{br\text{-}dh}_{\mathcal{G}}$ in the AGM, with respect to a Shoup-generic reduction.

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

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

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

4.2.1 Shoup-generic reduction from DLOG to BRDH

Claim 4.5. $\mathbf{dlog} \overset{(2,1)}{\Longrightarrow}_{\mathsf{alg}} \mathbf{br\text{-}dh}$.

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, the received group elements of $\mathsf{A}_{\mathsf{alg}}$ is $(g, \mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$. 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

$$\mathbf{Z} = g^{b_1}\mathbf{X}^{b_2}\mathbf{Y}^{b_3}\mathbf{Z}_1^{a_1}\ldots\mathbf{Z}_L^{a_L}. \tag{2}$$

In the following, we construct a Shoup-generic reduction $\mathsf{R}_{\mathsf{gen}}$ that calls $\mathsf{A}_{\mathsf{alg}}$ exactly once such that for $\mathsf{B}_{\mathsf{alg}} := \mathsf{R}_{\mathsf{gen}}^{\mathsf{A}_{\mathsf{alg}}}$ we have

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

Next, we present the description of the Shoup-generic reduction algorithm $\mathsf{R}_{\mathsf{gen}}$. On input a challenging term $\mathbf{X} := g^x$, it first randomly samples $y \twoheadleftarrow \mathbb{Z}_p$ and a bit $b \twoheadleftarrow \{0,1\}$. After that, $\mathsf{R}_{\mathsf{gen}}$ computes $\mathbf{Y} := g^y$ and $\mathbf{Z} := \mathbf{X}^y$. Let $z_1||\ldots||z_L$ be the canonical binary representation of $\mathbf{Z}$ and $\mathbf{U}_i := g^{2i+z_i}$ [5]. If $b = 0$, then $\mathsf{R}_{\mathsf{gen}}$ calls $\mathsf{A}_{\mathsf{alg}}$ with inputs $(\mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$, and if $b = 1$, then $\mathsf{R}_{\mathsf{gen}}$ calls $\mathsf{A}_{\mathsf{alg}}$ with inputs $(\mathbf{Y}, \mathbf{X}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$.

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 $\mathsf{R}_{\mathsf{gen}}$ computes $\mathbf{W} = g^{b_2}$, and solves x as follows:

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

[5] In Shoup's GGM, generic algorithms are allowed to compute $\mathbf{U}_i$, while in Maurer's GGM, those operations are not allowed.

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

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 $[\mathbf{Z}]_{\vec{v}} \twoheadleftarrow \mathsf{A}(\mathbf{X}, \mathbf{Y}, \mathbf{U}_1, \ldots, \mathbf{U}_L)$;
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.

- 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 (2) 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

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

Moreover, we note that the bit b is independent of the $\mathsf{A}_{\mathsf{alg}}$'s view, which means that $\mathsf{A}_{\mathsf{alg}}$'s output is independent of b. Besides, when $b = 0$, $\mathsf{A}_{\mathsf{alg}}$ takes inputs $(\mathbf{X}, \mathbf{Y})$, and when $b = 1$, $\mathsf{A}_{\mathsf{alg}}$ takes inputs $(\mathbf{Y}, \mathbf{X})$, where we note that from $\mathsf{A}_{\mathsf{alg}}$'s view, $(\mathbf{X}, \mathbf{Y})$ and $(\mathbf{Y}, \mathbf{X})$ are identically distributed. Thus, it is apparent that,

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

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

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

□

4.2.2 Hardness of the BRDH in Shoup's GGM

It is trivial to note that $\mathbf{br\text{-}dh}_{\mathcal{G}}$ is easy in Shoup's GGM, because Shoup-generic adversaries can easily identify the i-th bit of $\sigma(xy)$. Concretely, the i-th bit of $\sigma(xy)$ is "0" if and only if $\mathbf{U}_i = \sigma(2i)$.

4.3 Discussions

First, we note that both **o-dh** and **br-dh** are 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.

Second, the reduction algorithms for our examples (i.e., **o-dh** and **br-dh**) are Shoup-generic but not Maurer-generic, as both of the reductions make use of the group representation. As a result, these two examples *cannot* serve as the evidence that "hardness of security games in Shoup's GGM cannot be transferred via a Maurer-generic reduction in the AGM".

5 Hardness of security games in Shoup's GGM cannot be transferred, via a Maurer-generic reduction in the AGM

In the previous section, we see that the hardness of security games in Shoup's GGM cannot be transferred via a Shoup-generic reduction in the AGM. In this section, we strengthen our analysis by showing that the hardness in Shoup's GGM cannot be transferred, even the reduction is a *Maurer-generic* in the AGM. Besides, our example in this section indicates that Shoup's GGM is not weaker than the AGM either; this means that the AGM and Shoup's GGM are *incomparable*. For readability, we recall again the main lemma, i.e., Lemma 2.2, in [FKL17] in the grey box below.

Lemma 2.2. *Let* $\mathbf{G}_{\mathcal{G}}$ *and* $\mathbf{H}_{\mathcal{G}}$ *be algebraic security games such that* $\mathbf{H}_{\mathcal{G}} \xrightarrow{(\Delta_\epsilon,\Delta_t)}_{\mathsf{alg}} \mathbf{G}_{\mathcal{G}}$ *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 the following, we design two algebraic security games say $\mathbf{G}_{\mathcal{G}}$ and $\mathbf{H}_{\mathcal{G}}$: there exists a Maurer-generic reduction $\mathsf{R}_{\mathsf{gen}}$ such that winning $\mathbf{H}_{\mathcal{G}}$ in the AGM can be reduced to winning $\mathbf{G}_{\mathcal{G}}$ in the AGM, and $\mathbf{H}_{\mathcal{G}}$ is hard in Shoup's GGM, but $\mathbf{G}_{\mathcal{G}}$ is easy in Shoup's GGM. More formally,

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

- $\mathbf{H}_{\mathcal{G}} \xrightarrow{(\Delta_\epsilon,\Delta_t)}_{\mathsf{alg}} \mathbf{G}_{\mathcal{G}}$ *with respect to a Maurer-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.*

Intuitively, we construct a security game $\mathbf{H}_{\mathcal{G}}$; we note that, our idea here is inspired by the impossibility result in [Den02]. Concretely, Dent [Den02] introduces a modified version of the original DLOG problem, and shows that his modified DLOG problem is *easy* in the standard model but is *hard* in the Shoup's GGM. We then prove that our variant is also *easy* in the AGM and the standard model. On the other side, we construct game $\mathbf{G}_{\mathcal{G}}$ to be a security game that, the adversary wins $\mathbf{G}_{\mathcal{G}}$ if the adversary outputs its inputs. Trivial to note that, $\mathbf{G}_{\mathcal{G}}$ is easy in both AGM and Shoup's GGM.

Now, we observe that, both $\mathbf{H}_{\mathcal{G}}$ and $\mathbf{G}_{\mathcal{G}}$ are easy in the AGM, thus the adversary on $\mathbf{H}_{\mathcal{G}}$ serves as the reduction from $\mathbf{H}_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$. We then present a *Maurer-generic* adversary on $\mathbf{H}_{\mathcal{G}}$ in the

AGM. (The reason we design a variant of Dent's example is that, we can prove the reduction in Maurer-generic in the variant case, and details can be found in Section 5.4.) However, in Shoup's GGM, $\mathbf{H}_{\mathcal{G}}$ is hard and $\mathbf{G}_{\mathcal{G}}$ is easy, which immediately refers to that the results in AGM cannot be transferred to the hardness in Shoup's GGM, even the reduction is Maurer-generic.

Moreover, we observe that, $\mathbf{H}_{\mathcal{G}}$ itself, serves as the evidence that the Shoup's GGM is not weaker than the AGM, as $\mathbf{H}_{\mathcal{G}}$ is hard in Shoup's GGM but easy in the AGM. Combing the analysis in Section 4, we conclude that the AGM and Shoup's GGM are incomparable.

5.1 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 Appendix A for the definition of evasive group relation.

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

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

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

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

5.2 A variant of Oracle-associated DLOG (VDLOG)

In this subsection, we introduce a variant of Dent's example, denoted as **v-dlog**. This variant **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:

$$\mathsf{O}_{\rho,R,x}(y) = \begin{cases} x & \text{if } (y,\rho(y)) \in R, \\ \perp & \text{otherwise} \end{cases} \tag{4}$$

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.

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

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

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

Theorem 5.3. *For any evasive relation R,* **v-dlog** *is $(O(\frac{t^2}{p}), t)$-hard in Shoup's GGM, with respect to random encoding.*

5.3 VDLOG is easy in the AGM

As in Theorem 5.3, **v-dlog**$_{\mathcal{G}}$ is hard in the GGM. However, next we can see the **v-dlog**$_{\mathcal{G}}$ is easy in the AGM.

Claim 5.4. **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

$$R = \{(1, g)\}. \tag{5}$$

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

$$\mathsf{O}_{\mathcal{G},R,x}(y) = \begin{cases} x & \text{if } (y, g^y) \in R, \\ \perp & \text{otherwise} \end{cases} \tag{6}$$

We can see that, there exists an adversary $\mathsf{A}^{\mathsf{O}_{\mathcal{G},R,x}(\cdot)}(g^x)$ that will output x with probability 1 by just making a query to the oracle $\mathsf{O}_{\mathcal{G},R,x}(\cdot)$ with inputs "1". In next subsection, we prove A is Maurer-generic.

□

5.4 On the Maurer-generic reduction

In this section, we set $\mathbf{H}_{\mathcal{G}}$ to be **v-dlog**$_{\mathcal{G}}$ and $\mathbf{G}_{\mathcal{G}}$ be the game that the adversary wins $\mathbf{G}_{\mathcal{G}}$ if the adversary outputs its inputs. We see that both $\mathbf{H}_{\mathcal{G}}$ and $\mathbf{G}_{\mathcal{G}}$ are easy in the AGM, thus the adversary against $\mathbf{H}_{\mathcal{G}}$ *serves as the reduction* from $\mathbf{H}_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$. Now, it suffices to prove that the adversary A in Section 5.3 is Maurer-generic. In fact, the adversary A works as follows:

- Step 1: takes (g, g^x) as inputs;

- Step 2: makes 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 with "1" and returns the response of oracle, which means it does not make use of any group representation[6]. Therefore, the adversary A is Maurer-generic, which means the reduction from $\mathbf{H}_{\mathcal{G}}$ to $\mathbf{G}_{\mathcal{G}}$ is Maurer-generic.

Remark 5.5. *One may argue that, the oracle $\mathsf{O}_{\mathcal{G},R,x}(\cdot)$ depends on the group representation and A cannot win* **v-dlog** *without queries to the oracle. We stress that, the oracle $\mathsf{O}_{\mathcal{G},R,x}(\cdot)$ is given by the challenger in the game* **v-dlog***, and the adversary only makes queries in the black-box way. Therefore, no matter how the oracle performs, it would not affect that A is Maurer-generic, as long as A makes no use of group representation.*

5.5 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 4, 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 Maurer-generic. Therefore, the hardness of games in Shoup's GGM cannot be transferred via a Maurer-generic reduction, even in the standard model.

Third, all our examples (i.e., **o-dh**, **br-dh**, and **v-dlog**) cannot be characterized by the Maurer's GGM. As a result, these three examples *cannot* serve as evidence that "the hardness of games in Maurer's GGM cannot be transferred via a Maurer-generic reduction in the AGM".

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.

[6] In the game **o-dlog**, when the adversary makes a query to the oracle, it has to send $(1, g^x)$, which indicates that the adversary makes use of the group representation g^x.

[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.

A 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 A.1 (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*

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

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