What a Wonderful World: zkSNARKs in the Algebraic Group Model are Universally Composable

Gaspard Anthoine[1,2], Dario Fiore[1], and Mahak Pancholi[1]

[1] IMDEA Software Institute, Madrid, Spain
[2] Universidad Politécnica de Madrid, Madrid, Spain

Abstract. Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs) are important cryptographic primitives critical in many real-world applications. zkSNARKs are not used in isolation but are deployed within a broader context in which other cryptographic protocols may be concurrently executed. Universal-Composability (UC) allows rigorous analysis of cryptographic primitives being used in such arbitrary contexts. A UC analysis is even more desirable for popular, well-audited, and heavily deployed zkSNARKs already being used in practice.

Prior works that study the UC security of existing zkSNARKs (without modifications) are either not modular, hence requiring case-by-case analysis for new proof systems, or have largely focused on zkSNARKs in the Random Oracle Model (ROM). The latter includes zkSNARKs with logarithmic proof sizes compiled from Interactive Oracle Proofs. This state of the art leaves out a large family of very efficient, often constant-size, zkSNARKs that rely on the Algebraic Group Model (AGM) and optionally on the ROM. This includes zkSNARKs compiled from Polynomial Interactive Oracle Proofs, such as Plonk and Marlin, among others.

In this work, we address the UC security for unmodified zkSNARKs that are proven secure in AGM (+ROM). Our approach is modular: we identify simple, and mostly standard properties on the underlying zkSNARK that imply UC security. We observe that checking these properties for existing zkSNARKs is a surprisingly simple task using the rigorous formulation of AGM from Jaeger and Mohan (CRYPTO'24). The simplicity and modularity of our framework makes it easy-to-use for concluding UC security of several zkSNARKs in the same setting. Concretely, using our framework we establish that Plonk and Marlin are UC secure without any overhead.

1 Introduction

Zero-knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs) are one of the most useful cryptographic ideas that are being deployed in several real-world applications. Their prevalence necessitates best possible security guarantees. Furthermore, zkSNARKs are not used in isolation but are often composed with other cryptographic primitives within a larger application. In such settings, it is of utmost importance to ensure that the interaction of zkSNARKs with concurrently-executed arbitrary protocols or their composition within the context of a bigger protocol do not compromise their security.

The Universal Composability (UC) framework [18] addresses exactly this problem. It enables the modular design and analysis of cryptographic protocols and guarantees security even against adversaries that run multiple concurrent sessions and mount arbitrary attacks.

Prior attempts at achieving UC-secure zkSNARKs usually lift or transform an existing zkSNARK to make it secure in the UC framework [5, 32]. These approaches substantially alter the underlying scheme, incurring significant overhead, and do not address many of the existing (unaltered) zkSNARKs commonly used in practice.

This paper focuses on UC security of existing zkSNARKs without modifications. Necessary ingredients for UC security of zkSNARKs are *straight-line extraction* (SLE), *black-box* extraction (BBE), and *simulation-extractability* (SIMEXT). Informally, a proof system is SLE if a valid witness can be extracted without interacting with the adversary (hence no rewinding); it is BBE if the extractor works without knowing the code of the adversary; and it is SIMEXT if it is *non-malleable*, i.e., an adversary cannot maul an existing proof to forge a new proof on some statement without knowing the corresponding witness.

Groth [36] showed in a modular manner that if a (non-succinct) NIZK in the CRS model satisfies the aforementioned properties then it is UC-secure in the CRS-hybrid model. Succinct proof systems with straight-line black-box extraction, however, cannot exist in the plain CRS-model alone [17, 34]; instead they are extractable in some idealised model. Such models include the random oracle model (ROM), or an idealised group model such as the generic group model (GGM[3]) [53, 55] or the algebraic group model (AGM) [30, 39]. Nonetheless, in such idealised models, several modern zkSNARKs have recently been shown to satisfy SLE, SIMEXT, and sometimes BBE [28, 29, 33, 43]. This brings up the natural question:

Following [36], can we prove in a modular manner the UC security of existing zkSNARKs proven extractable in an idealised model?

To further motivate this question, let us briefly summarize the state of the art on SLE of existing zkSNARKs. We can broadly classify most of them in two categories.

Category 1: zkSNARKs with SLE in the ROM. This includes purely ROM-based zkSNARKs, such as Killian-Micali [42, 51], and the ones compiled from Interactive Oracle Proof (IOP) using a Merkle tree [9–11]. These zkSNARKs achieve logarithmic proof sizes at best.

The zkSNARKs in this category happen to naturally satisfy BBE and SLE via observability in the ROM. Leveraging this observation, two recent works [19, 22] addressed the UC security of these zkSNARKs, using a hybrid UC model where the ROM is formulated as a global programmable functionality (global-ROM). Concretely, both works show UC security for the Kilian–Micali zkSNARK [50], while [22] additionally considers zkSNARKs obtained by compiling an IOP with Merkle-tree commitments via the BCS compiler [11].

Category 2: zkSNARKs with SLE in an idealised group model (+ROM). This includes constructions such as [25, 37, 45], zkSNARKs obtained from polynomial interactive oracle proofs (PIOPs) using extractable polynomial commitment schemes [15, 16, 23, 31, 48, 48], as well as transparent zkSNARKs relying on discrete-log–type assumptions [13, 14, 54][4]. These schemes are also among the most efficient and succinct systems we have today, with many zkSNARKs in this category having constant proof size unlike those in Category 1.

For these zkSNARKs, the essential properties for UC crucially depend on an idealised group model rather than the ROM. Hence, techniques in [19, 22] are not applicable here.

The recent work of Bobolz et al. [12] is the first to address the UC security of a proof system in Category 2. They do so in a hybrid UC model where the GGM is formulated as a global group functionality (global-GGM). Now, via observing group queries, one can recover BBE and SLE as in Category 1. Concretely, they prove that Groth16 [37] is UC-secure without any modifications, under this new model.

While [12] provides a clear blueprint for establishing UC security in the GGM, it has three major limitations. First, it is a purely GGM-based model while several zkSNARKs additionally require the ROM. Second, it requires the prerequisite security properties of the zkSNARK to be defined and proven in GGM, whereas the common practice is to establish these in the AGM (+ROM). This means that to show UC security for Groth16, the analysis of SIMEXT has to be rewritten in the GGM, and cannot simply invoke the existing theorem proven in the AGM [7]. And finally, their analysis is not modular. In particular, the framework does not offer a modular way to leverage existing stand-alone security analyses, and explicitly leaves open the problem of a general *UC lifting theorem* that would allow one to derive UC security from simpler, game-based notions.

This lack of modularity is particularly unsatisfactory given that many existing zkSNARKs in Category 2 already satisfy, in the AGM, the key properties that seem necessary for UC security [28, 29, 33, 43]. From this perspective, these zkSNARKs look already "close" to being UC-secure, and one would ideally like to leverage their well-established AGM-based security analyses directly.

Hence the goal of this work is to study the following question and propose a modular framework to establish it:

If a zkSNARK satisfies SLE and SIMEXT in the AGM(+ROM), then is it UC secure in the presence of global-GGM+(global-ROM) functionality?

[3] Whenever we refer to GGM, we mean Shoup's GGM introduced in [55].

[4] We include these zkSNARKs in Category 2 since SLE very likely requires the algebraic group model (for example, see [35] for an SLE analysis of Bulletproofs [14])

1.1 Our Results

Our main result is answering the aforementioned question positively, i.e., zkSNARKs satisfying straight-line simulation-extractability in the AGM+ROM are UC-secure.

1. **UC lifting theorem:** The work of [12] introduces a UC framework attempting to target AGM-based proofs. However, it is quite non-trivial to use since it essentially requires most proofs to be redone from scratch within their framework (and in the GGM). Our main result leverages the observation that AGM-based zkSNARKs are *already* very close to UC: We identify simple properties for zkSNARKs in the stand-alone model formulated in GGM (+ROM) that imply UC security in the presence of a global-GGM +(global-ROM) functionality. Surprisingly, these properties closely mirror standard notions for zkSNARKs, or can be easily checked. This has the added benefit that potentially one does not need to prove *new* properties to obtain UC security.

   Theorem 1. *(Informal.) If a zkSNARK is complete, non-interactive zero-knowledge, and straight-line and black-box simulation-extractable in GGM (+ROM), then it is UC-secure in the presence of global-GGM (+non-programmable global-ROM) functionality.*

Given that most analyses exist in AGM (+ROM), the next question is how to lift these analyses to GGM (+ROM) without having to redo them. As a stepping stone, we can first lift guarantees to an intermediate model, called the *type-safe*[5] *(GGM+ROM)*. What remains is to show that guarantees in the type-safe(GGM+ROM) can then be lifted to Shoup's GGM + standard ROM.

2. **Type-safe(GGM+ROM) and GGM+ROM equivalence:** Extending the results of [40], the work of [57] establishes an equivalence between the type-safe GGM and GGM for single-stage, i.e., adversaries that pass full internal state between stages, and for primitives that can be expressed in the type-safe model. Building on similar observations, we show an analogous equivalence between the type-safe(GGM+ROM) and GGM+ROM.

   Theorem 2. *(Informal.) For primitives that exist both in type-safe(GGM+ROM) and GGM+ROM, security against single-stage adversaries is equivalent.*

Finally, applying our UC lifting theorem Theorem 1, one can conclude UC security. This yields a clear framework for obtaining UC security starting from stand-alone properties established in the AGM.

How to use our framework. We apply our framework to show that Plonk [31] and Marlin [23] are UC-secure in the presence of global-GGM and (non-programmable) global-ROM functionalities. The work of Faonio et al. [29] establishes that both Plonk and Marlin, without modifications, are SLE and SIMEXT in the (non-programmable) ROM and the AGM. Starting from this result, we can apply the lifting result in [39] to obtain SIMEXT of the same protocols in the type-safe(GGM+ROM). Using the equivalence result (Theorem 2), we then conclude that these zkSNARKs are SLE and SIMEXT in GGM+ROM. Completeness and NIZK follow similarly. Since these properties are the prerequisite for our UC lifting theorem (Theorem 1), we ultimately obtain UC security for Plonk and Marlin in the presence of global-GGM and (non-programmable) global-ROM functionalities.

2 Technical Overview

2.1 UC lifting theorem

Our goal is to identify properties in the stand-alone setting that are sufficient for proving UC. In this sense, we are closer to [22] in our methodology than [12, 19]. While ideally, one would like to model AGM behaviour in UC to be closer to the stand-alone setting, the UC-AGM model [1] is not suitable for non-interactive

[5] A type-safe GGM is closer to Maurer's GGM [49] where group elements are stored in a table and group operations can be performed only via their position in the table. As such no concrete group representation is available.

extractable primitives like zkSNARKs (see Appendix A). The next best option is to model GGM behaviour in UC as done in [12].

Proving UC security via game hops. We give a short recap on the common approach for showing UC security for zkSNARKs in the literature. UC security is proven by showing indistinguishability between a real-world and an ideal-world experiment. For a zkSNARK Π, the real-world experiment is defined as: the honest parties in a session *sid* execute the (prove,verify) protocols specified in Π while having access to the global and (if any) local functionalities. While in the ideal-world experiment, no protocol is executed. Instead the honest parties access an ideal NIZK functionality that, at a very high level, behaves as follows: On receiving a prove command from an honest party, run a simulator algorithm on just the statement to generate a proof string (which ensures zero-knowledge). On receiving a verify command from an honest party, run the extractor on the proof string, and check if the output is a valid witness. If it is valid, then output 1, otherwise output 0 (which ensures knowledge-soundness). In addition to this, a simulator algorithm is responsible for simulating the view of the corrupted party in *sid*. This includes simulating interaction with honest parties (as in the real world) and access to the local functionality.

A natural approach towards proving UC-security of a zkSNARK is via doing standard game-hops (some examples where this shows up are [22,32,36]) that go as follows:

- Hyb_0: This is the real-world experiment in UC.
- Hyb_1: The prove interface is replaced by the zero-knowledge simulator.
- Hyb_2: The verifier interface is swapped by running the extractor on the proof string to obtain a witness and checking whether the witness is valid or not. This is exactly like the ideal world UC experiment.

To argue indistinguishability between each hop, one would typically invoke an underlying stand-alone property (i.e. zero-knowledge for Hyb_0-Hyb_1, and simulation-extractability for Hyb_1-Hyb_2).

Consider the reduction for Hyb_0-Hyb_1 hop. For simplicity, assume that the parties only have access to a global-ROM (and not global-GGM). Assume that there exists an adversary $\mathcal{Z}$ who can successfully distinguish between $\mathsf{Hyb}_0, \mathsf{Hyb}_1$. Then, we build an adversary $\mathcal{A}$ given black-box access to $\mathcal{Z}$ against the zero-knowledge property. At a high level, $\mathcal{A}$ runs $\mathcal{Z}$ internally and is responsible for simulating its view. This includes answering prove,verify commands, and simulating accesses global-ROM, while interacting with the challenger in the zero-knowledge game in the presence of a standard ROM. $\mathcal{A}$ can simply forward the prove and verify commands to the underlying game and return the responses. To simulate responses to global-ROM given access to standard ROM, $\mathcal{A}$ proceeds as follows: forward all *sid* queries to the standard ROM in the underlying game, simulate all $sid' \neq sid$ queries locally, keep a local record of observable queries (this is possible since it sees all queries made by $\mathcal{Z}$), and responds appropriately when asked for the observable set[6]. Since global-ROM has a very easy domain separation and sid' responses are independent of *sid*, $\mathcal{A}$ perfectly simulates $\mathcal{Z}$'s view. Hence, any distinguishing advantage of $\mathcal{Z}$ implies the same advantage for $\mathcal{A}$ in distinguishing real proofs from simulated ones.

Extending to group-based zkSNARKs. For group-based zkSNARKs, we not only need to model a global-ROM but also a global-GGM functionality. Global-GGM models Shoup's GGM [55] as a global functionality. In Shoup's GGM, the group is modelled as a random mapping of $\mathbb{Z}_p$ to bit strings, and group operations can only be performed via queries to the GGM oracle.

The game hops follow similarly as above. However, building a reduction to an underlying property requires care. Since group elements are not independent for each session the domain separation is not as easy as in the case of global-ROM. As a result, in our reduction, we can no longer simulate sid' group queries locally, but all queries regardless of the session must be forwarded to the GGM oracle in the underlying game. Here, an issue arises when trying to simulate the observability interface of the global-GGM functionality. To understand the issue, first consider the mechanism of observability defined in [12].

In [12], every generator for some session *sid* is associated with a formal variable X_{sid}. A group element that is the result of a group operation now has an associated polynomial: for example, a group operation of the form $h \leftarrow g_1 + g_2$ has the polynomial $\mathsf{X}_1 + \mathsf{X}_2$ associated with it, where $\mathsf{X}_1, \mathsf{X}_2$ are the formal variables

[6] Global functionalities often have an OBSERVE interface that allows to observe "illegal" queries that $\mathcal{Z}$ might make to force the simulator in *sid* to fail.

previously defined for g_1, g_2, respectively. A group operation is labelled as illegal and hence observable if the polynomial associated with the result happens to have any foreign session variables $\mathsf{X}_{sid'}$ or a constant term. In short, operations involving other session's generators become observable.

In our reduction, the group operations performed by the algorithms in the underlying game (i.e., prove and verify interfaces) will not be visible to $\mathcal{A}$. This means that, while simulating observability for global-GGM, $\mathcal{A}$ might not be able to classify legal vs illegal queries, or might classify them differently than if it had the full representation. An example of this is: Let $g_{sid'}$ be the generator of sid'. On receiving a prove command from $\mathcal{Z}$, $\mathcal{A}$ forwards it to the prove interface of the underlying game, and receives a proof string $(\pi_1, \ldots, \pi_n)$ in return, where π_1 and π_2 are group elements. Later, $\mathcal{Z}$ makes a group query of the form $g_{sid'} + a \cdot \pi_1 + b \cdot \pi_2$. Looking at it naively, $\mathcal{A}$ classifies this group operation as illegal since it contains group elements from different sessions (and because it does not know the polynomials associated with π_1, π_2). However, with the full polynomial representation of π_1, π_2 perhaps the *sid* terms cancel out, and hence the operation will not be classified as illegal in the hybrid game.

We observe that the (symbolic) group oracle provides a natural mechanism to access the underlying polynomial representation. This means that maintaining partial observability is not a problem for $\mathcal{A}$. It can still determine observability of the future group elements by accessing the underlying polynomial representation via appropriate group queries to the GGM oracle. This allows it to verify if a query "cancels out" the foreign variables associated with other sessions. Concretely, it makes the group queries corresponding to the operation $a \cdot \pi_1 + b \cdot \pi_2$ and checks if the result equals the canonical generator (hence the *sid* terms cancel out). If yes, then classify as legal, and otherwise mark as observable.

A related issue arises when reducing to simulation-extractability in the last hop. In the case of global-ROM alone, the argument is that the extractor in the underlying game has an identical view as that of the extractor in the hybrid (only queries related to *sid*). Hence, they succeed or fail with the same probability. However, in our reduction, the extractor in the game has full view of the queries made by $\mathcal{Z}$ to the group oracle (since $\mathcal{A}$ forwards everything), while in the hybrid game it only has *sid* related views. We argue that if the generators for each session are chosen independently and if whenever the honest protocol samples a group element it samples it with high min-entropy, then the additional information available in the underlying game is not more useful. Hence, the success probability can now be related identically.

What we have shown above is that the usual game-hop strategy follows through even in the case of global-GGM with mild assumptions on the zkSNARK, and without drastically modifying the game-based properties. The game-based properties must exist in GGM+ROM for our reduction to work. However, most known simulation-extractability analyses exist in the AGM+ROM. To avoid having to redo analysis in Shoup's GGM (as is done in [12] for Groth16; simulation-extractability for Groth16 was already known in the AGM in a previous work [7]) and avoid subtle security gaps [58], below we discuss how to lift security proofs in the AGM+ROM to Shoup's GGM+ROM.

2.2 AGM+ROM to GGM+ROM lifting

The second result of this paper shows that security in *type-safe*(GGM+ROM) implies security in Shoup's GGM + standard ROM. Together with the lifting from AGM+ROM to type-safe(GGM+ROM) implicit in [39], this completes the chain from existing AGM-based analyses all the way to UC security.

Type-Safe model of computation [39]. In this framework, each variable in the algorithm is assigned a data type: $\langle \mathsf{bits} \rangle$ for a bit string, and $\langle \mathsf{elem} \rangle$ for a group element. Data types restrict how an algorithm can interact with specific variables. Algorithms may perform any computation over $\langle \mathsf{bits} \rangle$ variables locally, but computation over $\langle \mathsf{elem} \rangle$ variables is restricted. To perform group operations, algorithms must query dedicated oracle functions that accept $\langle \mathsf{elem} \rangle$ variables as input and return $\langle \mathsf{elem} \rangle$ outputs. Additionally, they may have access to an encoding function $\sigma : \langle \mathsf{elem} \rangle \to \langle \mathsf{bits} \rangle$ that outputs a concrete representation of a group element, and a decoding function $\sigma^{-1} : \langle \mathsf{bits} \rangle \to \langle \mathsf{elem} \rangle$ that outputs a group element given its representation. The type of allowed access defines various models: "type-safe" GGM algorithms may only call group operation functions and have no access to σ, σ^{-1}; "type-safe" AGM algorithms additionally have access to σ but not σ^{-1}, meaning that they have access to the concrete group encoding but cannot generate group elements locally without using functions for group operations

This framework is then used to rigorously formalize common intuitions about the AGM (we will call this the type-safe AGM). Under this formalization, they characterize conditions under which security in type-safe AGM can be lifted to type-safe GGM.

Type-Safe Model with Random Oracle. Many cryptographic primitives, including zkSNARKs that are compiled using the Fiat-Shamir transform (which covers most known zkSNARK constructions), require access to a random oracle. Typically, for standard random oracles, this requires algorithms to convert group elements to their encoding in bitstring format. This is unavailable in the type-safe model[7]. Hence, naively zkSNARKs cannot be expressed as a type-safe computation.

This can be resolved by using a *typed* random oracle. A type-safe random oracle is defined as $\{\langle \mathsf{elem} \rangle, \langle \mathsf{bits} \rangle\}^* \to \langle \mathsf{bits} \rangle$, i.e., it accepts group elements directly as inputs rather than requiring their bit-string encodings.

Even in this case, the original AGM to type-safe GGM lifting result can be recovered. The key insight is that while a typed random oracle could in principle be used to derive a concrete encoding for a group element (by treating RO outputs as encodings), this does not break type-safety: as long as the algorithm can't compute the inverse $\hat{\sigma}^{-1}$, group elements can only come from group queries.

Using this augmented model of type-safety, we can lift security guarantees of most zkSNARKs—including Plonk and Marlin—from AGM+type-safe ROM to type-safe(GGM+ROM). Our main goal, however, is security in Shoup's GGM + standard ROM.

Type-Safe(GGM+ROM) to Shoup's GGM + standard ROM. Zhandry [57] showed an equivalence between the type-safe GGM and Shoup's GGM for single-stage adversaries and for primitives that can be expressed in the type-safe model. But that work did not consider access to random oracle. Our result builds on Zhandry's proof strategy, with careful modifications to handle the augmented model with random oracle access. As in [57], we show equivalence in both directions, though for the purposes of this paper only one direction is strictly necessary.

To show equivalence, we must show how to translate an algorithm written in the type-safe(GGM+ROM) model to one in Shoup's GGM + standard ROM. The main subtlety lies in appropriately translating RO queries: a typed RO query takes $(\langle \mathsf{elem} \rangle, \langle \mathsf{bits} \rangle)$ pairs as input, while a standard RO query is an arbitrary bit string. An obvious attempt is to replace each $\langle \mathsf{elem} \rangle$ input with its bit-string encoding, yielding a standard RO query. However, this can cause distinct queries in the type-safe ROM to collide in the standard ROM. Dealing with this part requires some care when proving the two directions of the equivalence, as we describe next.

Direction 1: Shoup's to type-safety: We need to show that an algorithm in one model can be translated to an equivalent one in the other with identical winning probability; collisions among RO queries are therefore a potential issue. However, since the encoding in Shoup's GGM is a random injection, the probability of such a collision is negligible, and the argument goes through except with a small error.

Direction 2: type-safety to Shoup's: This requires constructing a type-safe adversary given a Shoup's GGM adversary. A Shoup's GGM adversary makes standard RO queries as arbitrary bit strings, but the type-safe adversary must forward these to a typed RO that expects $(\langle \mathsf{elem} \rangle, \langle \mathsf{bits} \rangle)$ pairs. Parsing an arbitrary bit string as a sequence of group elements and bit strings without context is not possible.

The key observation is that the underlying security game only requires RO queries of a certain structure to be faithfully forwarded: namely, for zkSNARKs, those that an honest verifier could make. These are determined by the protocol description: in particular, their expected length and the information on how to parse them. Only these queries are relevant to the winning condition of the underlying game. All other RO queries can be simulated locally (without requiring any parsing). Now, given an arbitrary bit-string query to the standard RO, the reduction will check whether it has the expected length; if so, it optimistically parses it as prescribed by the protocol and forwards it to the typed RO.

We remark that because of this second result, the type-safe random oracle turns out to be a stepping stone towards proving security in the standard random oracle model, and is never used by the final algorithms.

[7] The full version of [39] analyses algorithms that require access to a random oracle. However, their treatment differs slightly compared to this work. We describe in more detail in Section 2.5.

2.3 Plonk and Marlin are UC-secure

As an application of our framework we show that Plonk and Marlin are UC-secure. The main "non-blackbox" task here is to check that the reductions (in [29]) to a computational problem are all type-safe. Intuitively since such reductions are written in the algebraic group model, they very likely satisfy the formal requirement as well. Nonetheless, in Appendix H, we reason that this is indeed the case. After this, we can in a black-box manner lift these results all the way up to UC.

2.4 Limitations

An important criticism of the generic group model is that it allows to prove certain (contrived) schemes secure that cannot be securely instantiated under any group in the standard model [26]. Moreover, security in GGM only holds against generic adversaries. What would be ideal is security against standard adversaries since a known class of sub-exponential time attacks on the discrete-log assumption are via index calculus algorithms which make use of the specific group structure. However, composable guarantees in the GGM at the very least confirm that any attack on the system must inherently rely on the group structure.

2.5 Related Works

We present a more complete survey of related-works in Appendix A.

UC-secure zkSNARKs in global-ROM. Ganesh et al. [32] were the first to present a UC-secure zkSNARK with constant-sized proofs in the CRS-model. Badertscher et al. [5] build upon [32] to achieve a UC zkSNARK in the transparent model. Both works are in the non-programmable global-ROM, and present a compiler that can lift an underlying zkSNARK with appropriate properties to a UC-secure one. The works of [19, 22] study UC security of existing zkSNARKs without modifications. In the programmable global-ROM (programmability seems necessary for such unmodified zkSNARKs), both show that Killian-Micali zkSNARK is UC-secure, while [22] additionally shows UC-security for certain IOP-based zkSNARKs obtained via the BCS transform [11]. Their constructions are secure against adaptive adversaries.

The work of [22] is closer to our work in terms of methodology as they too aim to identify stand-alone properties that imply UC security. However, they are only able to realise a weaker NIZK functionality. Informally the weaker functionality is not required to extract if the statement has already been proven in this session. On the other hand, [19] models all subprotocols directly in UC, for example, by introducing functionalities for non-interactive (vector) commitment schemes, and building a UC-secure zkSNARK from these.

For simplicity, we only consider static adversaries in this work, leaving the formalisation for an adaptive corruption as an open problem.

UC-secure zkSNARKs in global-GGM. To tackle AGM-based zkSNARKs, Bobolz et al. [12] present a new UC framework modelling the group as a global generic functionality with restricted observability. Under this new model, they prove Groth16 [37] to be UC-secure. Since Groth16 is malleable, it can only realise a weak form of NIZK functionality, while in this paper we aim for the standard NIZK functionality.

Type-safe random oracle. The full version of [39], and a concurrent work of Wang et al. [56], both require a type-safe ROM in their analyses. The former considers a specific instantiation of random oracle that can be type-safe, namely, the oracle that works by lazy sampling responses. While the latter (similar to our work) treats it more generically. The ultimate goal of [56] differs from our work: the work of [56] introduces a type-safe random oracle to complete the framework originally defined in (the conference version of) [39] and uses it analyse security of signature schemes (while staying in the type-safe framework throughout). Alternatively, we use it as a stepping stone towards UC security of proof systems under the standard Shoup and RO formulations.

AGM to GGM lifting results. The original AGM formulation in [30] already showed such a lifting result claiming that under mild conditions AGM security carries over to the GGM. Subsequently Katz et al. [58] presented a counter-example to this claim by exploiting certain ambiguities in the original formulation. This

motivated the work of [39] that established a formal framework to derive a hierarchy of idealised models and recover the lifting result. The full version of [39], and a concurrent work of Wang et al. [56] extends this to include access to random oracles. Although, both works provide a way to interpret Shoup's generic algorithms within the hierarchy, there are two issues with it: (1) The class of Shoup adversaries is a superset of type-safe adversaries; the hierarchy does not say anything about implying security from a weaker model to a stronger model. And, (2) the security guarantee obtained in the type-safe model would inevitably rely on a *typed*-RO and would syntactically differ from the standard Shoup's and RO formulations. An important goal in this paper is: obtain UC security under the *existing* models of generic group and random oracle functionalities. This is so as to avoid a *yet* another variant of UC functionalities (e.g., translating it in the pseudocode framework of [39]).

Zhandry [57] showed that any single-stage adversary in Shoup's GGM can be reduced to type-safe adversaries but doesn't consider random oracles. We build upon this result to include access to *standard* random oracles.

3 Preliminaries: Stand-Alone Properties

General notational conventions. We denote all stand-alone algorithms with font-type A, stand-alone oracles with O. The UC algorithms are boldfaced, while the functionalities are calligraphic. For a function $\tau : A \to B$, we write $\mathsf{img}(\tau)$ to denote its image, i.e., $\{y \text{ s.t. } \exists x \in A, \tau(x) = y \in B\}$. We say **assert** x to mean if $\neg x$ then output $\perp$. Scalars are denoted by a, vectors by $\boldsymbol{a}$, and formal variables by X, polynomials in $\mathbb{Z}_p[\mathsf{X}]$ by f. $A : B$ denotes list concatenation. We use $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ to denote inner-product and $\boldsymbol{a} \circ \boldsymbol{b}$ to denote hadamard product.

3.1 Idealized models

Generic group models. We work in Shoup's Generic Group Model (GGM) with pairings [53, 55, 57]. The model is parametrized by (p, S_1, S_2, S_t), where S_1, S_2 are source groups, S_t is the target group, p is a prime. We assume sets S_1, S_2, S_t to be disjoint. Three random injections $\tau_i \xleftarrow{\$} \textsc{Inj}(\mathbb{Z}_p, S_i)$ for $i \in \{1, 2, t\}$ are fixed at initialization, and all group operations are performed exclusively via oracle queries to the group oracle (denoted by G). We remark that the group functionalities only consider a random encoding of size p, therefore there is no need to consider cases where the adversary may query an invalid encoding. The standard definition can be found in (G, Figure 1).

Figure 1: Shoup's GGM oracle: G

$\textsc{Init}()$

1: For $i \in \{1, 2, t\} : \tau_i \xleftarrow{\$} \textsc{Inj}(\mathbb{Z}_p, S_i)$

$\textsc{Op}(i, (g_1, a_1), (g_2, a_2))$

1: **assert** $(g_1, g_2, a_1, a_2) \in S_i^2 \times \mathbb{Z}_p^2$
2: $h \leftarrow \tau_i(a_1 \tau_i^{-1}(g_1) + a_2 \tau_i^{-1}(g_2))$
3: **return** h

$\textsc{CanonicalGen}(i)$

1: **return** $\tau_i(1)$

$\textsc{Pair}(g_1, g_2)$

1: **assert** $(g_1, g_2) \in S_1 \times S_2$
2: $h \leftarrow \tau_t(\tau_1^{-1}(g_1) \cdot \tau_2^{-1}(g_2))$
3: **return** h

Standard random oracle. Let $\mathsf{RO} : \langle \mathsf{bits} \rangle \{0,1\}^* \to \langle \mathsf{bits} \rangle \{0,1\}^n$ be a random oracle taking arbitrary length bit strings as input and outputting length-n bit strings.

3.2 Succinct non-interactive zero-knowledge proofs

(Succinct) non-interactive zero knowledge argument of knowledge. A zkSNARK for an indexed relation $\mathcal{R}$ is a tuple of PPT algorithms $\Pi = (Setup, P, V)$ that work as follows. All algorithms have oracle access to RO, G,

– $\mathsf{pp} \leftarrow Gen(1^\lambda)$: On input security parameter, it outputs a set of public parameters consisting of group generators.

– $(\mathsf{crs}, \mathsf{td}) \leftarrow Setup(\mathsf{p})$: On input p, it outputs the setup string crs.

– $\pi \leftarrow P(\mathsf{crs}, x, w)$: On input statement x and the witness w, it outputs a proof string π.

– $b \leftarrow V(\mathsf{crs}, x, \pi)$ On input x, π it outputs a decision bit.

We require a zkSNARK to satisfy completeness, non-interactive zero-knowledge (NIZK), and simulation-extractability (SIMEXT) (definitions in Appendix B). We use a (mostly) standard NIZK definition with minor syntactic differences needed for our work.

Figure 2: Security game for SIMEXT

$\underline{\mathsf{SIMEXT}_{\mathcal{A}}(1^\lambda)}$

1: $\mathsf{pp} \leftarrow Gen(1^\lambda)$
2: $(\mathsf{crs}, \mathsf{td}) \leftarrow Setup(\mathsf{pp})$
3: $\mathcal{Q}_S := \{\}, \mathsf{adWin} = 0$
4: $\mathsf{st} := (\mathsf{td}, \mathcal{Q}_S, \mathsf{adWin})$
5: $\mathcal{A}^{\mathsf{RO},\mathsf{G},\tilde{P},\tilde{V}}(\mathsf{crs})$
6: **return** adWin

$\underline{\tilde{P}(x, w)}$

1: **assert**$(x, w) \in \mathcal{R}$
2: $(\pi, \mathsf{st}) \leftarrow S(\mathsf{st}, \mathsf{crs}, x)$
3: $\mathcal{Q}_S \leftarrow \mathcal{Q}_S : (x, \pi)$
4: **return** π

$\underline{\tilde{V}(x, \pi)}$

1: $b \leftarrow V^{\mathsf{RO},\mathsf{G}}(\mathsf{crs}, x, \pi)$
2: If $(x, \pi) \in \mathcal{Q}_S$ **return** 1
3: **for** $i \in \{1, 2, t\}$ **do**,
4: $\quad \{(i, h_j, B_j)\}_{j \in k_i} \leftarrow E_0(\mathsf{crs}, x, \pi)$
5: $\quad$ For $j \in k_i$, $\boldsymbol{c}^{(i)}[j] \leftarrow \textsc{FindRep}(i, h_j, B_j, \mathsf{tr}_\mathsf{G})$
6: **end for**
7: Run $w \leftarrow E_1(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \boldsymbol{c}^{(t)})$
8: If $b = 1 \wedge (x, w) \notin \mathcal{R}$ **set** adWin $= 1$
9: **return** b

Definition 1 (Simulation Extractability). *Let $\Pi = (Setup, P, V)$ be a non-interactive argument of knowledge for relation $\mathcal{R}$ satisfying NIZK w.r.t simulator S (Definition 7). Π satisfies simulation-extractability in the non-programmable random oracle and generic group model w.r.t S, if there exists an efficient extractor $E = (E_0, E_1)$ sharing states, such that for all PPT distinguishers $\mathcal{A}$ the following advantage is negligible in λ,*

$$\mathsf{Adv}^{\mathsf{SIMEXT}}_{\mathcal{A}}(\lambda) := \Pr\left[\mathsf{SIMEXT}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where game SIMEXT *is defined in Figure 2.* tr_G *denotes the oracle transcript w.r.t* G.

Differences from standard notions of simulation-extractability. Our definition fixes a specific extraction strategy which, as we argue later, is natural for group-based zkSNARKs. The extractor splits into two sub-algorithms: (1) E_0 takes (crs, x, π) and identifies group elements h_j along with basis $\{B_j\}$; (2) h_j's representations w.r.t. basis elements are then recovered via FindRep from the group oracle transcript; (3) finally E_1 decodes these to output the witness.

The function FindRep (taken from [12], Algorithm 1) linearly scans tr_G and records representation of all group elements w.r.t. some basis B.

This "canonical extractor" mirrors AGM extraction closely: there, the adversary supplies group representations directly; here, FindRep recovers them from the transcript. The decoding step done by E_1 is typically trivial. Moreover, in several constructions it is often clear which group elements in the proof string are necessary for extraction (as done by E_0 here). For example, in PIOP based zkSNARKs the notion of witness-carrying polynomials [4], i.e. the polynomials that encode the entire witness, explicitly marks the commitment value necessary for extraction.

3.3 Additional properties from NIZK

We require mild additional properties from the underlying NIZK for the UC reduction to go through. Together they capture the intuition that any *internal* group elements in an execution of algorithms $(Setup, \mathcal{P}, \mathcal{V}, S)$ remains hidden from the adversary unless explicitly part of the output.

Property P_1: We require that Gen will sample group generators by sampling at random from source group sets S_1, S_2.

Property P_2: Every group element computed by $Setup$ is derived via group operations on $\mathsf{p} \leftarrow Gen$. Every group element computed by $(\mathcal{P}, \mathcal{V}, S)$ is either sampled with high min-entropy or derived via group operations on crs elements. In particular, group operations on canonical generators or the case where group elements appear in the statement x are excluded.

Property P_3: Internal element hiding. For an algorithm A^{G}, let Elems be the set of group labels returned by G to A, and Pub be the set of group labels appearing in the public input or output of A. Define internal labels as $\mathsf{Hid} := \mathsf{Elems} \setminus \mathsf{Pub}$. For an adversary $\mathcal{B}$, let Query be the set of group labels $\mathcal{B}$ submits as inputs to G, and View be the group labels $\mathcal{B}$ has learned, i.e., the union of results of Query and Pub.

We say that A satisfies internal element hiding if no PPT $\mathcal{B}$ who invokes A once[8] satisfies the following except with negligible probability: there exists an element g in the inputs to Query such that it is not in View, i.e., $\mathcal{B}$ submits a hidden element to G before having learned it.

We require algorithms $Setup, \mathcal{P}, \mathcal{V}, S$ to satisfy internal element hiding.

Discussion. Properties 1 and 2 are natural and stated explicitly for completeness. Property 3 is more subtle. In Shoup's GGM, adversaries have access to group encodings. So a NIZK prover may leak internal group elements without violating SIMEXT: for example by including in the proof a bit-flipping of the encoding of an intermediate value g^a not required to be hidden for zero-knowledge purposes. Such leakage would create a distinguishable event in the UC hybrids. Note however that any such leakage is harmless since the underlying protocol satisfies zero-knowledge. More concretely, for any leaky $\mathcal{P}$ there exists a non-leaky $\mathcal{P}'$ that simply outputs the leaked value explicitly, making it part of the public output rather than a hidden internal element. Formalizing this generically is difficult since the leakage function can be arbitrary. While type-safety (a property that restricts computation over group elements; more in Section 6) is a sufficient generic condition to ensure Property 3 (proof in Appendix B.4), we preferred to give a more general definition, as there is a broader class of non-type-safe algorithms that can be audited to verify non-leakage. Yet, for the protocols considered in this work, type-safety suffices. Finally, we believe that most natural proof systems will satisfy this property (whether type-safe or not).

4 Preliminaries: Universal Composability

The formal description of the global non-programmable random oracle functionality ($\mathcal{G}_{\mathsf{RO}}$) can be found in Appendix C (Functionality 4).

4.1 Global generic group functionality

We work with the global generic group functionality introduced in [12], available in two variants: a concrete variant $\mathcal{G}$ (Functionality 5, Appendix C) based on Shoup's GGM, and a purely symbolic variant $\mathsf{s}\text{-}\mathcal{G}$ (Functionality 1) that admits perfect session separation.

Both $\mathcal{G}$, $\mathsf{s}\text{-}\mathcal{G}$ track group operations across sessions and flag illegal ones as observable, i.e. operations that mix elements from different sessions or introduce a non-zero constant term. Concretely, each generator for session sid is assigned a fresh formal variable X_{sid}, and every derived group element carries an associated polynomial: for example, $h \leftarrow g_1 + g_2$ gets polynomial $X_1 + X_2$ where X_1, X_2 are the variables of g_1, g_2. A group operation is then flagged as observable if its resulting polynomial contains variables from a foreign session $\mathsf{X}_{sid'}$ or a non-zero constant term. For pairing operations, observability additionally requires that any degree-1 term in the result only involves target group variables (disallowing operations with the canonical generators).

The key difference between the $\mathcal{G}$ and $\mathsf{s}\text{-}\mathcal{G}$ lies in how the internal map τ_i is defined and how session separation is enforced. In $\mathcal{G}$, τ_i maps scalars in $\mathbb{Z}_p$ to group elements via a random injection. As explained in [12], this may result in an imperfect domain separation, i.e., some group elements may belong to two

[8] In general, we require security against multiple invocations of A. The full definition is in Appendix B.4.

sessions. To resolve this, they propose s-$\mathcal{G}$. Here, τ_i is defined lazily over formal polynomials in $\mathbb{Z}_p[\mathsf{X}_{\mathsf{sid}}, \ldots]$, assigning fresh formal variables to generators and computing polynomials symbolically for all derived elements. Since distinct polynomials map to distinct group elements, s-$\mathcal{G}$ enforces a perfect equality check:

$$\tau_i(\mathsf{f}(\mathsf{X}\ldots)) = \tau_i(\mathsf{f}'(\mathsf{X}\ldots)) \iff \mathsf{f}(\mathsf{X}\ldots) = \mathsf{f}'(\mathsf{X}\ldots)$$

which guarantees perfect session separation by construction.

It was shown in [12] that $\mathcal{G}$ and s-$\mathcal{G}$ are indistinguishable to all polynomial-time algorithms (Lemma 3 in [12]), so we work primarily with s-$\mathcal{G}$ in our proofs.

We use a simplified version of s-$\mathcal{G}$ that omits interfaces not needed for our black-box reduction (e.g., ComputeSymbolic, Laurent polynomials).

4.2 NIZK and setup functionality

We use a syntactic variant of the standard $\mathcal{F}_{\mathsf{nizk}}$ functionality [12, 38], and a setup functionality $\mathcal{F}_{\mathsf{setup}}$ to model trusted generation of the setup string (crs) for the non-interactive proof system. Informally, it has two interfaces: (1) $\textsc{Init}_{sid}$ is invoked at the time of initialization. It first generates the group generators for each source group, by sampling at random from sets S_i and registering them with $\mathcal{G}$ via Touch. It then generates crs by running $Setup$ using these generators. (2) $\textsc{GetCrs}_{sid}$ can be called by parties in session sid to obtain crs.

Functionality 2: $\mathcal{F}_{\mathsf{setup}}$

(Parametrized by the algorithm $Setup$ of a NIZK Π.)

$\textsc{Init}_{sid}()$

1: **for** $i \in \{1,2\}$ **do**:
2: $\quad g_{i,sid} \xleftarrow{\$} S_i$
3: $\quad \textsc{Touch}_{sid}(g_{i,sid})$
4: **end for**
5: $(\mathsf{crs}, \mathsf{td}) \leftarrow Setup(g_{1,sid}, g_{2,sid})$

$\textsc{GetCrs}_{sid}()$

1: **return** crs

Functionality 3: $\mathcal{F}_{\mathsf{nizk}}$

$\textsc{Init}_{sid}()$

1: $T \leftarrow [\,]$

$\textsc{Prove}_{sid}(x, w)$

1: **assert** $(x, w) \in \mathcal{R}$
2: $\pi \leftarrow \mathcal{S}.\textsc{Simulate}_{sid}(x)$
3: $T \leftarrow T \cup (x, \pi)$
4: **return** π

$\textsc{Verify}_{sid}(x, \pi)$

1: **if** $(x, \pi) \in T$ **then return** 1
2: **end if**
3: $w \leftarrow \mathcal{S}.\textsc{Extract}_{sid}(x, \pi)$
4: **if** $(x, w) \in \mathcal{R}$ **then return** 1
5: **end if**
6: **return** 0

4.3 UC security in the $\mathcal{G}$-hybrid

We recall the standard syntax for ideal functionalities and protocols in the UC framework in Appendix C.

The UC security of protocol Π against static corruptions is formally defined in Definition 2 and consists of showing that no efficient environment who can corrupt parties statically can distinguish between the real and the ideal world.

Functionality 1: s-$\mathcal{G}$

Parameters of the functionality:

- A prime number p
- Sets S_1, S_2, S_t with $|S_i| = p$ for all $i \in \{1, 2, t\}$.

Internal state maintained by the functionality:

- τ_i: Maps polynomials in $\mathbb{Z}_p[\mathsf{X}_{sid}, \ldots]$ (instead of $\mathbb{Z}_p$) to S_i, for $i \in \{1, 2, t\}$.
- $\mathsf{X}_{i,sid}$: Initially empty list. This will record formal variables defined for session *sid*. X_{sid} denotes the concatenation $\mathsf{X}_{1,sid} : \mathsf{X}_{2,sid} : \mathsf{X}_{t,sid}$.
- Ob: Initially empty list. This will record the set of observable group operation queries.

Furthermore:

- A polynomial f is in $\mathsf{Legal}_{sid}^{\text{OP}}$ if it only contains variables in X_{sid} and a zero constant term.
- A polynomial f is in $\mathsf{Legal}_{sid}^{\text{PAIR}}$ if in addition to constraints in $\mathsf{Legal}_{sid}^{\text{OP}}$, it has no degree 1 monomials.

$\underline{\text{INIT}_{sid}()}$

for $i \in \{1, 2, t\}$: **do**
  $\tau_i \leftarrow \{\}$
  TAU$(i, 1)$
end for

$\underline{\text{CANONICALGEN}_{sid}(i)}$

return TAU$(i, 1)$

$\underline{\text{OBSERVE}_{sid}}$

return Ob

$\underline{\text{OP}_{sid}(i, (g_1, a_1), (g_2, a_2))}$

assert$(g_1, g_2, a_1, a_2) \in S_i^2 \times \mathbb{Z}_p^2$
for $j \in \{1, 2\}$ **do**
  TOUCH$_{sid}(i, g_j)$
end for
$\mathsf{f} \leftarrow a_1 \tau_i^{-1}(g_1) + a_2 \tau_i^{-1}(g_2)$
$h \leftarrow$ TAU(i, f)
if $\mathsf{f} \notin \mathsf{Legal}_{sid}$ **then**
  $Ob \leftarrow Ob \cup \{(\text{OP}, i, g_1, g_2, a_1, a_2, h)\}$
end if
return h

$\underline{\text{TOUCH}_{sid}(i, g)}$

if $g \notin Im(\tau_i)$ **then**
  Initialize fresh variable X
  $\mathsf{X}_{i,sid} \leftarrow \mathsf{X}_{i,sid} : [\mathsf{X}]$
  $\tau_i(\mathsf{X}) \leftarrow g$
end if

$\underline{\text{TAU}(i, \mathsf{f})}$ //Internal function

if $\tau_i(\mathsf{f}) = \perp$ **then**
  $\tau_i(\mathsf{f}) \xleftarrow{\$} S_i \setminus Im(\tau_i)$
end if
return $\tau_i(\mathsf{f})$

$\underline{\text{PAIR}_{sid}(g_1, g_2)}$

assert$(g_1, g_2) \in S_1 \times S_2$
for $i \in \{1, 2\}$ **do**
  TOUCH$_{sid}(i, g_i)$
end for
$\mathsf{f} \leftarrow \tau_t(\tau_1^{-1}(g_1) \cdot \tau_2^{-1}(g_2))$
$h \leftarrow$ TAU(t, f)
if $\mathsf{f} \notin \mathsf{Legal}_{sid}$ **then**
  $Ob \leftarrow Ob : [(\text{PAIR}, g_1, g_2, h)]$
end if
return h

Definition 2. *We say that protocol Π UC-realizes the functionality $\mathcal{F}_{nizk}$ in the $(\mathcal{G}, \mathcal{F}_{setup})$-hybrid model if for all PPT adversaries $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for all PPT environments $\mathcal{Z}$, the following ensembles are indistinguishable:*

$$\left\{\mathsf{REAL}^{\mathcal{G},\mathcal{F}_{setup}}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx \left\{\mathsf{IDEAL}^{\mathcal{G}}_{\mathcal{F}_{nizk},\mathcal{S},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

5 UC security from Game-Based Definitions

In this section, we present a general lifting theorem that bridges security definitions in the stand-alone setting with UC framework. We show that any zkSNARK satisfying completeness, zero-knowledge, simulation-extractability, and mild conditions specified in Section 3.3 in Shoup's GGM+non-programmable ROM is UC-secure in the global-GGM+non-programmable GROM hybrid model.

5.1 Stand-alone protocol to UC-secure protocol

Let *Gen* be the algortihm producing public group parameters pp given the description of the group, and let $\Pi = (Setup, P, V)$ be a NIZK in the presence of G+non-programmable ROM. A natural transformation of Π, denoted by $\boldsymbol{\Pi = (\mathbf{Setup}, \mathbf{P}, \mathbf{V})}$ (Protocol 1) is the corresponding protocol in the UC-framework in the $\mathcal{F}_{\mathsf{setup}}$, $\mathcal{G}, \mathcal{G}_{\mathsf{RO}}$-hybrid.

$\boldsymbol{\Pi}$ is essentially a wrapper around the stand-alone algorithms along with the following changes: calls to G oracle are forwarded to the $\mathcal{G}$ functionality, and calls to RO are forwarded to $\mathcal{G}_{\mathsf{RO}}$ (with the *sid* added).

Protocol 1: $\boldsymbol{\Pi}$

$\underline{\mathbf{Setup}_{sid}}$
1: Call $\mathcal{F}_{\mathsf{setup}}.\textsc{Init}()$

$\underline{\mathbf{V}_{sid}}(x, \pi)$
1: $\mathsf{crs} \leftarrow \mathcal{F}_{\mathsf{setup}}.\textsc{GetCrs}_{sid}()$
2: $b \leftarrow V^{\mathcal{G},\mathcal{G}_{\mathsf{RO}}(sid,\cdot)}(\mathsf{crs}, x, \pi)$
3: **return** b

$\underline{\mathbf{P}_{sid}}(x, w)$
1: **if** $(x, w) \notin \mathcal{R}$ **then return** $\perp$
2: **end if**
3: $\mathsf{crs} \leftarrow \mathcal{F}_{\mathsf{setup}}.\textsc{GetCrs}_{sid}()$
4: $\pi \leftarrow P^{\mathcal{G},\mathcal{G}_{\mathsf{RO}}(sid,\cdot)}(\mathsf{crs}, x, w)$
5: **return** π

Theorem 3. *Let Gen be the algorithm producing public group parameters and let $\Pi = (Setup, P, V)$ be a SNARK satisfying completeness (Definition 6), non-interactive zero-knowledge (Definition 7) simulation-extractability (Definition 1), and properties $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$ (Section 3.3) in Shoup's GGM (Figure 1) and the non-programmable random oracle model. Then the protocol $\boldsymbol{\Pi = (Setup, P, V)}$ in Protocol 1 UC-realises $\mathcal{F}_{nizk}$ in the $\mathcal{F}_{setup}$, $\mathcal{G}$, $\mathcal{G}_{\mathsf{RO}}$-hybrid model.*

Next, we describe hybrids from Real to Ideal experiments in this section. We refer to Section 5.2 for the description of the reduction adversary and Section 5.3 for indistinguishability claims.

Proof. **Simulator definition.** The simulator $\mathcal{S}$ (Algorithm 2). Let (E, S) be the extractor and simulator algorithms for Π. $\mathcal{S}$ wraps all group and random oracle queries made by the adversary, maintaining a local observation list $Ob_{\mathcal{S}}$. It simulates $\mathcal{F}_{\mathsf{setup}}$ and hence knows the crs trapdoor. On $\textsc{Prove}$ command it runs S given the trapdoor. On $\textsc{Verify}$ command it runs E after completing the observation list via a call to $\textsc{Observe}_{sid}$.

Hybrids. W.l.o.g., we consider two sessions: the challenge session *sid* and one other session sid'. $\mathcal{Z}$ issues $\textsc{Init}, \textsc{Prove}, \textsc{Verify}$ only for *sid* and may query $\mathcal{G}, \mathcal{G}_{\mathsf{RO}}$ via sid'. We drop the group subscript i in X and τ whenever it is clear from the context. Pairings are expressed as Hadamard products: $\mathsf{X}_{sid} \circ \mathsf{X}_{sid'} = \mathsf{X}_{1,sid} \circ \mathsf{X}_{2,sid'} + \mathsf{X}_{2,sid} \circ \mathsf{X}_{1,sid'}$. $\mathcal{Z}$ makes s_1 $\textsc{Prove}$ commands and s_2 $\textsc{Verify}$ commands.

Note that hybrids Hyb_4-Hyb_8 mainly have syntactic differences between them. We prefer to write them explicitly in the hybrids to bring it closer to the reduction adversary's description.

- Hyb_0: Real world security game (as defined in Definition 2).
- Hyb_1: Replace $\mathcal{G}$ with its symbolic counter-part ($\mathsf{s}\text{-}\mathcal{G}$).
- Hyb_2: Replace all the honest parties with a single party $\mathcal{B}$. It simulates the real-world view of the adversary and the environment for the rest of the protocol. It runs on behalf of honest parties. Additionally, it intercepts queries to $\mathcal{G}, \mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{setup}}$ and relays back the responses.
- Hyb_3: In this hybrid we introduce an additional abort condition corresponding to the case where $\mathcal{Z}$ makes a group query using an element g such that g appears as an intermediate group element in an honest algorithm's execution. This is rule out by Property P_3 (internal element hiding, Appendix B.4).
- Hyb_4: Change behaviour of $\mathsf{s}\text{-}\mathcal{G}$ for observability: On deriving the resultant polynomial in $\textsc{Op}, \textsc{Pair}$ queries, if the constant term is non zero then add it to Ob (i.e., even without checking the full polynomial).
- Hyb_5: Change observability for $h \leftarrow \textsc{Op}_{sid}(i, g_1, g_2, a_1, a_2)$ queries: Compute $\mathsf{f} := a_1\tau_i^{-1}(g_1) + a_2\tau_i^{-1}(g_2)$. Partition according to sid, sid' parts: $\mathsf{f} := \mathsf{f}^{(1)}(\mathsf{X}_{i,sid}) + \mathsf{f}^{(2)}(\mathsf{X}_{i,sid'})$, where both polynomials are linear with zero constant term (because of the previous hybrid). Add this tuple to $\textsc{Observe}$ if $\mathsf{f}^{(2)}(\mathsf{X}_{i,sid'}) \neq 0$.
- Hyb_6: Change observability for $\textsc{Op}_{sid'}$ queries: Partition f as in Hyb_5 and add tuple to Ob if $\mathsf{f}^{(1)}(\mathsf{X}_{i,sid}) \neq 0$.
- Hyb_7: Change observability for $h \leftarrow \textsc{Pair}_{sid}(g_1, g_2)$ queries: Compute $\mathsf{f} := \tau_1^{-1}(g_1) \cdot \tau_2^{-1}(g_2)$ and partition as $\mathsf{f} := \mathsf{f}^{(1)}(\mathsf{X}_{sid}) + \mathsf{f}^{(2)}(\mathsf{X}_{sid}, \mathsf{X}_{sid'})$, where $\mathsf{f}^{(2)} := \langle \boldsymbol{v}_1, \mathsf{X}_{sid} \circ \mathsf{X}_{sid'} \rangle + \langle \boldsymbol{v}_2, \mathsf{X}_{sid'} \circ \mathsf{X}_{sid'} \rangle$, for some vector of scalars $\boldsymbol{v}_1, \boldsymbol{v_2}$. Add to Ob if either: (1) writing $\langle \boldsymbol{v}_1, \mathsf{X}_{sid} \circ \mathsf{X}_{sid'} \rangle$ as $\mathsf{f}^{(3)}(\mathsf{X}_{sid'}) \cdot \mathsf{f}^{(4)}(\mathsf{X}_{sid})$, the product is non-zero; or (2) $\langle \boldsymbol{v}_2, \mathsf{X}_{sid'} \circ \mathsf{X}_{sid'} \rangle$ is a non-zero polynomial.
- Hyb_8: Change observability for $\textsc{Pair}_{sid'}$ queries: Partition f as in Hyb_7. Add tuple to Ob if either: (1) $\mathsf{f}^{(1)} \neq 0$; or (2) $\langle \boldsymbol{v}_1, \mathsf{X}_{sid} \circ \mathsf{X}_{sid'} \rangle \neq 0$ (as checked in the previous hybrid).
- Hyb_9: Instead of forwarding calls to $\mathcal{F}_{\mathsf{setup}}$, run $\textsc{Init}_{sid}()$ and $\textsc{GetCrs}_{sid}$ code in the simulator $\mathcal{S}.\textsc{Init}_{sid}$ and $\mathcal{S}.\textsc{GetCrs}_{sid}$ (Algorithm 2) to respond to the queries.
- For $j \in [\mathsf{s}_1]$, Hyb_{9+j}: For the j-th $\textsc{Prove}(x, w)$ command issued by $\mathcal{Z}$ for an honest party, replace the honest prover's algorithm with $\mathcal{S}.\textsc{Simulate}_{sid}(x)$ (as described in Algorithm 2).
- For $j \in [\mathsf{s}_2]$, $\mathsf{Hyb}_{9+\mathsf{s}_1+j}$: For the j-th $\textsc{Verify}(x, \pi)$ command issued by $\mathcal{Z}$ for an honest verifier, replace the honest verifier's algorithm with $\mathcal{S}.\textsc{Extract}_{sid}(x, \pi)$ (as described in Algorithm 2).
- $\mathsf{Hyb}_{9+\mathsf{s}_1+\mathsf{s}_2}$: Change $\mathsf{s}\text{-}\mathcal{G}$ behaviour for deciding observability to as it is decided in Hyb_1.
- $\mathsf{Hyb}_{10+\mathsf{s}_1+\mathsf{s}_2}$: Replace $\mathsf{s}\text{-}\mathcal{G}$ with its non-symbolic counterpart.
- $\mathsf{Hyb}_{11+\mathsf{s}_1+\mathsf{s}_2}$: Replace $\mathcal{B}$ by $\mathcal{S}$. This is the ideal world.

5.2 Adversary's definition

Before arguing indistinguishability between each hybrid, we informally describe the adversary $\mathcal{A}^*$ against some property based definition with interfaces $(Setup, P, V)$. Since $\mathcal{A}^*$'s algorithm is the same for each hop, we define it only once. The full formal description can be found in Appendix D.

The overall reduction strategy is as follows: Let $\mathcal{Z}$ be a UC adversary that can successfully distinguish between $\mathsf{Hyb}_i, \mathsf{Hyb}_{i+1}$. $\mathcal{A}^*$ is given black-box access to $\mathcal{Z}$ and is responsible for simulating its view as in the hybrid games, while interacting with the challenger of the underlying security notion and with access to stand-alone $\mathsf{s}\text{-}\mathsf{G}, \mathsf{RO}$[9]. $\mathcal{A}^*$ wraps around $\mathcal{Z}$: it forwards $\mathcal{G}_{\mathsf{RO}}$ queries with sid prefix to RO, locally simulates sid' responses via lazy sampling. $\mathcal{A}^*$ can correctly answer $\mathcal{G}_{\mathsf{RO}}.\textsc{Observe}$ since it sees all of $\mathcal{Z}$'s queries. For group queries, all $\mathsf{s}\text{-}\mathcal{G}$ queries, regardless of the session will be forwarded to $\mathsf{s}\text{-}\mathsf{G}$. $\mathcal{A}^*$ additionally maintains its own version of tables τ (which will be identical to what is maintained by $\mathsf{s}\text{-}\mathsf{G}$ apart from variables having different representations)

[9] We will actually reduce to the NIZK property in presence of a *symbolic* variant of $\mathsf{s}\text{-}\mathsf{G}$ (Figure 4). Similar to the indistinguishability between UC functionalities $(\mathcal{G}, \mathsf{s}\text{-}\mathcal{G})$, one can also show indistinguishability between stand-alone model $(\mathsf{G}, \mathsf{s}\text{-}\mathsf{G})$. A change that would be needed to make the proof go through is to add an empty $\textsc{Touch}$ interface for G.

On INIT_{sid} command, $\mathcal{A}^*$ interacts with the challenger (Ch) of underlying game to receive the crs. All PROVE_{sid}/VERIFY_{sid} commands are passed on to the prove/verify interfaces of the underlying security game. This creates a difficulty for maintaining the group observability sets: unlike in the real UC experiment, $\mathcal{A}^*$ only has partial access to group representations. Specifically, it cannot see the representations of group elements computed internally by Ch.

The reduction relies on the observation that $\mathcal{A}^*$ does not need the full polynomial representation of honest elements to decide observability. It only needs to verify if a query cancels out the "foreign" variables of sid'. Concretely, $\mathcal{A}^*$ decomposes operations into components depending on known adversarial variables and unknown honest proof variables. It then performs local checks via group oracle queries to determine which components vanish. For example, let π be a group element in the output of a PROVE query. Since $\mathcal{A}^*$ does not know how it was computed, it just assigns a fresh variable X_π for it. Later if $\mathcal{Z}$ makes $\text{OP}_{sid'}(i, g_1, g_2, a_1, a_2)$ query where $\mathcal{A}^*$ can decompose g_1 as $a \cdot \mathsf{X}_{sid} + b \cdot \mathsf{X}_\pi$, and g_2 as $\mathsf{Y}_{sid'}$. To decide observability for sid', $\mathcal{A}^*$ can check whether all sid terms vanish by checking if $\mathsf{s\text{-}G}.\text{OP}(i, g, \pi, a, b)$ returns an encoding of 0.

Completing the informal description: to decide observability, $\mathcal{A}^*$ decomposes the resultant polynomials exactly as specified in hybrids Hyb_4-Hyb_8 but in terms of variables it stores locally. Instead of checking if some specified term of polynomial is non-zero (as is done in each hybrid), it perform group operations corresponding to this term and checks if the result is an encoding of 0.

5.3 Indistinguishability between hybrids

Notations. X_π denotes the variables associated with group elements output by either the PROVE command or INIT_{sid} execution. $\mathsf{X}_{sid} \cup \mathsf{X}_\pi$ contain all variables belonging to session sid. The shorthand $\mathsf{f}^{(3)}(\tau(\mathsf{X}_{sid'}))$ means: replace each X in f by its corresponding group element and perform group operations as per f's definition.

Indistinguishability for $\mathsf{Hyb}_0 - \mathsf{Hyb}_1$ is given by Lemma 3 in [12]. Hyb_1-Hyb_9 differ mainly syntactically (claims and proofs in Appendix E). Below we focus on the reduction to NIZK and SIMEXT for showing indistinguishability for hybrids Hyb_{9+j} to $\mathsf{Hyb}_{10+\mathsf{s}_1+\mathsf{s}_2}$.

Claim 1. *For* $j \in \{0, \ldots, \mathsf{s}_1 - 1\}$, $\mathsf{Hyb}_{9+j} \approx \mathsf{Hyb}_{9+j+1}$.

Proof. The difference in Hyb_{9+j+1} is that it runs SIMULATE instead of PROVE. Given $\mathcal{Z}$ who distinguishes between the two hybrids, we can construct $\mathcal{A}_{\mathsf{NIZK}}$ against NIZK property of Π (Definition 7). The code of $\mathcal{A}_{\mathsf{NIZK}}$ is the same that of $\mathcal{A}^*$ described in Section 5 and Appendix D. It suffices to show that when $b = 0$ (resp. $b = 1$) in the NIZK game, view simulated by $\mathcal{A}_{\mathsf{NIZK}}$ is indistinguishable from Hyb_{9+j} (resp. Hyb_{9+j+1}).

When an INIT_{sid} or a PROVE_{sid} command is made the group operations performed internally Ch will not be visible to $\mathcal{A}^*$. Hence, $\mathcal{A}^*$ assigns fresh variables for group elements present in the output and adds them to the set X_π. Subsequently, $\mathcal{A}^*$ decides observability using X_π instead of the full representation available in Hyb_{9+j}. This may possibly lead to misclassification of group operation tuples in two cases:

Case 1: Collision with Internal Variables. If $\mathcal{Z}$ queries a group element g that was part of an intermediate computation by $(Setup, P)$ but never revealed, $\mathcal{A}^*$ would assign g a fresh variable in whichever session it first appears – possibly in sid'. Whereas Hyb_{9+j} would have defined it with respect to sid. This mismatch changes the contents of Ob, which $\mathcal{Z}$ can detect via an OBSERVE query. However, this case is ruled out by Hyb_3.

Case 2: Algebraic Dependencies. We consider queries where $\mathcal{Z}$ combines a group element present in the output of $\text{INIT}_{sid}, \text{PROVE}_{sid}$ query with other group elements and $\mathcal{A}^*$ must decide whether the operation if observable or not. While we concretely refer to PROVE_{sid} below, as before the argument for INIT_{sid} and PROVE_{sid} is the same. We consider each type of group operation in turn.

- **Linear Operations in sid:** $h \leftarrow \text{OP}_{sid}(i, (g_1, a_1), (g_2, a_2))$. The polynomial corresponding the result h splits as $\mathsf{f} := \mathsf{f}^{(1)}(\mathsf{X}_{sid}, \mathsf{X}_\pi) + \mathsf{f}^{(2)}(\mathsf{X}_{sid'})$. Since observability depends only on $\mathsf{f}^{(2)}$, and $\mathcal{A}_{\mathsf{NIZK}}$ maintains full symbolic representation for sid' variables, it can compute $\mathsf{f}^{(2)}$ explicitly. This yields identical observability decisions as in Hyb_{9+j}.

– **Linear Operations in sid':** $h \leftarrow \text{OP}_{sid'}(i, (g_1, a_1), (g_2, a_2))$.

Here, $\mathcal{A}_{\mathsf{NIZK}}$ must check if $\mathsf{f}^{(1)}(\mathsf{X}_{sid}, \mathsf{X}_\pi)$ cancels out. Let $\hat{\mathsf{X}}$ be the variables defined by $\mathsf{s\text{-}G}$ on queries made by $(Setup, P)$ or sid queries forwarded by $\mathcal{A}_{\mathsf{NIZK}}$. Clearly, each $\mathsf{X} \in \mathsf{X}_\pi$ can be rewritten as a linear polynomial $\mathsf{p}(\hat{\mathsf{X}})$ with zero constant term. Hence, let $\mathsf{f}^{(1)}$ be rewritten as $\mathsf{p}(\hat{\mathsf{X}})$ (for some linear polynomial p with zero constant term).

$\mathsf{f}^{(1)}(\tau(\mathsf{X}_{sid}), \tau(\mathsf{X}_\pi)) = \tau(0)$ iff $\mathsf{p}(\hat{\mathsf{X}})$ is a zero polynomial. The LHS is the check performed by $\mathcal{A}_{\mathsf{NIZK}}$, while the RHS is the check performed in Hyb_{9+j}. Hence, deciding observability is again identical for $\mathcal{A}_{\mathsf{NIZK}}$ and Hyb_{9+j}.

– **Pairing Operations in sid:** $h \leftarrow \text{PAIR}_{sid}(g_1, g_2)$.

The polynomial representation of h as defined by $\mathcal{A}_{\mathsf{NIZK}}$ can be written as $\mathsf{f} := \mathsf{f}^{(1)}(\mathsf{X}_{sid}, \mathsf{X}_\pi) + \mathsf{f}^{(2)}(\mathsf{X}_{sid}, \mathsf{X}_\pi, \mathsf{X}_{sid'})$. The term $\mathsf{f}^{(2)}$ can be further split as $\mathsf{f}^{(3)}(\mathsf{X}_{sid'}) \cdot (\mathsf{f}^{(4)}(\mathsf{X}_{sid}) + \mathsf{f}^{(5)}(\mathsf{X}_\pi)) + \mathsf{f}^{(6)}(\mathsf{X}_{sid'})$ (coefficients for f polynomials here are fully determined by $\mathcal{A}_{\mathsf{NIZK}}$).

To decide observability two checks are performed:

1. $\mathcal{A}_{\mathsf{NIZK}}$ locally checks if the polynomial $\mathsf{f}^{(6)}(\mathsf{X}_{sid'})$ is a zero polynomial. This is identical to Hyb_{9+j}'s check.
2. $\mathcal{A}_{\mathsf{NIZK}}$ checks $\mathsf{f}^{(3)}$ is non zero polynomial locally or $(\mathsf{f}^{(4)}(\tau(\mathsf{X}_{sid})) + \mathsf{f}^{(5)}(\tau(\mathsf{X}_\pi)) = \tau(0)$ (by querying $\mathsf{s\text{-}G}$ for computing group operations). Again, we can equivalently rewrite this in terms of variables $\hat{\mathsf{X}}$. The latter will be checked by Hyb_{9+j}.

– **Pairing Operations in sid':** $h \leftarrow \text{PAIR}_{sid'}(g_1, g_2)$ As in the previous case, here too the behaviour of $\mathcal{A}_{\mathsf{NIZK}}$ will be identical to that of Hyb_{9+j}.

Claim 2. *For $j \in \{0, \ldots, \mathsf{s}_2 - 1\}$ $\mathsf{Hyb}_{9+\mathsf{s}_1+j}$ and $\mathsf{Hyb}_{9+\mathsf{s}_1+j+1}$ are indistinguishable.*

Proof. The difference between the hybrids is that the extractor may fail to output a valid witness in $\mathsf{Hyb}_{9+\mathsf{s}_1+j+1}$ while the verifier accepts. This we reduce to simulation-extractability of Π. Given $\mathcal{Z}$ who distinguishes between the two hybrids, we build an adversary $\mathcal{A}_{\mathsf{SE}}$ whose code is the same as $\mathcal{A}^*$. The argument that $\mathcal{A}$ generates an indistinguishable view is the same as already made in the NIZK case.

If a proof verifies in the UC experiment, it will also verify in the SIMEXT experiment (since all oracle queries are forwarded). All we need to argue is that the view of the extractors (E_0, E_1) in the SIMEXT experiment is identical to that of the extractors in the UC experiment in hybrid $\mathsf{Hyb}_{9+\mathsf{s}_1+j+1}$. Hence, if it fails to extract a valid witness in the latter, then this fails even in the SIMEXT, breaking the assumption. The difference in the view of the extractors in the two experiments is that in SIMEXT game, the extractors learn the entire group oracle transcript (including sid' queries) denoted by tr_{G}, while in the hybrid they only learn Ob which is a strict subset. Hence, we must rule out the possibility of valid extraction given this extra information.

The inputs to E_0 is identical in both experiments and so the output (i, h_j, B_j) outputs are identical. The outputs of $\text{FINDREP}(\cdot, Ob_{sid})$ and $\text{FINDREP}(\cdot, \mathsf{tr}_{\mathsf{G}})$ may differ if $Rep[h]$ is set differently given access to a larger tr.

We can partition tr_{G} as $\mathsf{tr}_{ob} || \mathsf{tr}_{se} || \mathsf{tr}_{\bar{ob}}$, where tr_{ob} denotes queries in Ob (as maintained by $\mathcal{A}_{\mathsf{SE}}$), tr_{se} are queries by SIMEXT interfaces, and $\mathsf{tr}_{\bar{ob}}$ the rest (this only includes legal queries for sid' session maintained by $\mathcal{A}_{\mathsf{SE}}$). This partitioning is perfect because of the symbolic representation.

If for some $(\text{OP}, i, g_1, g_2, a_1, a_2, h) \in \mathsf{tr}_{\bar{ob}}, Rep[h] \neq 0$, then h is expressible in terms of sid basis elements, making the operation observable and contradicting membership in $\mathsf{tr}_{\bar{ob}}$.

For pairing operations, we will have $Rep[h] := Rep_t(0^{n_t}, Rep_1[g_1] \otimes Rep_2[g_2])$. Then we partition the entire transcript as before to obtain: $\mathsf{tr}_{ob} || \mathsf{tr}_{se} || \mathsf{tr}_{\bar{ob}}$. As $\mathbb{Z}_p$ has no zero divisors, the only way for the tensor product $Rep[g_1] \otimes Rep[g_2]$ to be non-zero is to have $Rep[g_1] \neq 0$ and $Rep[g_2] \neq 0$, therefore for all pairing operation $(\text{PAIR}, g_1, g_2, h) \in \mathsf{tr}_{\bar{ob}}, Rep[h] = \mathbf{0}$ otherwise they would be observable.

6 Inferring GGM security from AGM security

Most security analysis of zkSNARKs are in FKL-AGM, i.e. in the AGM model defined by Fuchsbauer, Kiltz, and Loss in [30]. Security in the AGM does not automatically transfer to Shoup's GGM [57,58]. However, [39] showed that this link holds between AGM and *Maurer's* GGM, extending the result to cover type-safe random oracle access in the full-version by instantiating RO as a type-safe algorithm. We provide an alternative argument on the consistency of adding a type-safe random oracle to the type-safe model, removing any assumption on the structure of the underlying oracle[10].

Furthermore, security in Maurer's GGM is not enough to invoke our UC lifting theorem, which requires security to hold in Shoup's GGM with a *standard* ROM. Bridging this remaining gap is the main contribution of this section.

In Section 6.1 we recall the lifting result of [39]; then, in Section 6.2, we prove that security in Maurer's GGM with a typed random oracle implies security in Shoup's GGM with a standard random oracle.

6.1 Preliminaries: Type-Safe Oracle Framework

Data-types. Variables in algorithms are explicitly typed as $\langle \mathsf{bits} \rangle$ or $\langle \mathsf{elem} \rangle$. The latter type is associated with group elements and can be seen as containing the discrete logarithm.

Typed Random Oracle. A typed random oracle is a natural generalization of the classical ROM. Here, the oracle can be queried with group elements directly without requiring their bit-string encoding.

Definition 3 (Typed Random Oracle). *A typed random oracle is* $\mathsf{RO}_{\mathsf{ts}} : ((\langle \mathsf{elem} \rangle e, \langle \mathsf{bits} \rangle s)^*) \to \langle \mathsf{bits} \rangle r$ *that takes a list of tuples* $e \in \mathbb{G} \cup \{\emptyset\}$ *and a bit string* $s \in \{0,1\}^* \cup \{\emptyset\}$ *and outputs a uniformly sampled bit string* r *of size* n*.*

Given a typed-random oracle, one can generalize the pseudocode framework to algorithms that require access to a random oracle. Intuitively, this does not break the "type-safeness" because the random oracle itself is typed, and can be directly queried on group elements rather than requiring the algorithm to interpret group elements as bit strings (which would be the case in classical random oracles). Notice that a query on group element g and on a string s such that s has the same value as the concrete representation of g yields different outputs. This is the expected behaviour as they should be seen as two different inputs (or alternatively, one can interpret these two queries as directed to two distinct random oracles).

Importantly, in most settings of interest, typed-random oracles can easily replace the classical random oracle without any modifications to the algorithms or their security argument (as we will see later in the case studies for Plonk and Marlin). A real-world example where this is not the case is ECDSA [52], whose signer algorithm necessarily requires concrete representation of the group elements.

Universe of Computation [39]. To model bilinear groups, we will consider $S = S_1 \sqcup S_2 \sqcup S_t$, each set is disjoint. The encoding function is a tuple $(\sigma_1, \sigma_2, \sigma_t)$, such that $\sigma(A) = \sigma_i(A)$ for $A \in S_i$, and σ^{-1} is defined similarly as a tuple of three functions. A universe of computation U_σ specifies the tuple (S, Π, Σ, σ), where $\Pi = (\textsc{Op}, \textsc{Pair}, \textsc{Gen})$ and $\Sigma = \textsc{Eq}$ (described in Figure 3).

Framework for computation. In this universe, algorithms may only compute over $\langle \mathsf{elem} \rangle$ variables via the oracles in Figure 3[11]. It is implicit here that $\langle \mathsf{elem} \rangle$ variables containing a value x should be thought of as the value g_i^x corresponding to an implicit group generator g_i where $i \in \{1, 2, t\}$ indicates which group S_i the variable belongs to. Group operations within the same group S_i are performed via $\textsc{Op}$ while $\textsc{Pair}$ computes the pairing operation. Fresh generators can be sampled via $\textsc{Gen}$. Computation over $\langle \mathsf{bit} \rangle$ can be local. Additionally, they have access to a typed random oracle $\mathsf{RO}_{\mathsf{ts}}$.

[10] In [39], the type-safe random oracle is required to be modelled as a lazy sampling algorithm.

[11] The only difference from the pseudocode framework in [39] is that instead of viewing computation as internal functions, we view them as oracles. This simplifies the incorporation of random oracles and allows observing queries in the UC setting. Note that this is different from the oracle framework defined in [39], which is meant to be a faithful formulation of Maurer's GGM. As a matter of fact, the latter appears to be *stateful* which can be problematic as observed by Zhandry et al. in [57].

Figure 3: Types of Oracles Accesses

Oracle $\textsc{Gen}(i)$
1: **assert** $i \in \{1, 2\}$
2: Sample $l \xleftarrow{\$} S_i$
3: **return** $\langle \mathsf{elem} \rangle l$

Oracle $\textsc{Op}(i, \langle \mathsf{elem} \rangle l_1, \langle \mathsf{elem} \rangle l_2, \langle \mathsf{bits} \rangle a, \langle \mathsf{bits} \rangle b)$
1: **assert** $a, b \in \mathbb{Z}_p^2$
2: **assert** $l_1, l_2 \in S_i^2$
3: Compute $l \leftarrow a \cdot l_1 + b \cdot l_2$
4: **return** $\langle \mathsf{elem} \rangle l$

Oracle $\textsc{Pair}(\langle \mathsf{elem} \rangle l_1, \langle \mathsf{elem} \rangle l_2)$
1: **assert** $l_1 \in S_1, l_2 \in S_2$
2: Compute $l \leftarrow l_1 \times l_2$
3: **return** $\langle \mathsf{elem} \rangle l$

Oracle $\textsc{Eq}(i, \langle \mathsf{elem} \rangle l_1, \langle \mathsf{elem} \rangle l_2)$
1: **assert** $l_1, l_2 \in S_i^2$
2: If l_1, l_2 contain the same $x \in \mathbb{Z}_p$, **return** 1.
3: Otherwise, **return** 0.

Oracle $\mathcal{O}_\sigma(\langle \mathsf{elem} \rangle l)$
1: **return** $\langle \mathsf{bits} \rangle a \leftarrow \sigma(l)$

Oracle $\mathcal{O}_{\sigma^{-1}}(\langle \mathsf{bits} \rangle b)$
1: **return** $\langle \mathsf{elem} \rangle g = \sigma^{-1}(b)$.

Oracle $\mathsf{RO}_{\mathsf{ts}}(x)$
1: T is initially empty table
2: Parse x as series of $\langle \mathsf{elem} \rangle$, $\langle \mathsf{bits} \rangle$ variables
3: **if** $x \in T$: **then**
4: **return** $T[x]$
5: **end if**
6: Sample $y \xleftarrow{\$} \{0,1\}^\lambda$
7: Store $T[x] \leftarrow y$
8: **return** y

Hierarchy of algorithms. The hierarchy $\mathsf{TSA}[U] \subset \mathsf{AA}[U_\sigma] \subset \mathsf{SA}[U_\sigma]$ for an *efficient* encoding σ is defined by oracle access: type-safe algorithms $\mathsf{TSA}[U]$ use only $(\textsc{Op}, \textsc{Pair}, \textsc{Eq})$, algebraic algorithms $(\mathsf{AA}[U_\sigma])$ additionally access $\mathcal{O}_\sigma$ to obtain group element representations but are still restricted to perform group operations via oracles, standard algorithms $(\mathsf{SA}[U_\sigma])$ can also access $\mathcal{O}_{\sigma^{-1}}$, enabling local group operations. This hierarchy is preserved when algorithms additionally have access to $\mathsf{RO}_{\mathsf{ts}}$.

While adding a random oracle to a type-safe model might seem counter-intuitive at first, Jaeger and Mohan showed that it was coherent as long as the type-safe random oracle can be represented as a lazy sampling algorithm. We now argue that even without assuming any structure about the oracle, the hierarchy is preserved. We observe that $\mathsf{TSA}[U]^{\mathsf{RO}_{\mathsf{ts}}} \cong \mathsf{AA}[U_{\bar{\sigma}}]$, i.e. giving to a type-safe algorithm access to a typed random oracle is equivalent to giving to an algebraic algorithm access to a randomized encoding $\bar{\sigma}$. Furthermore, for an algorithm in $\mathsf{AA}[U_{\bar{\sigma}}]$, any group operation must still be performed via queries to (Π, Σ) (since it does not have access to $\bar{\sigma}^{-1}$), i.e., $\mathsf{TSA}[U] \subset \mathsf{AA}[U_{\bar{\sigma}}]$. We also have that $\mathsf{TSA}[U] \subset \mathsf{TSA}[U]^{\mathsf{RO}_{\mathsf{ts}}}$ and $\mathsf{AA}[U_{\bar{\sigma}}] \subset \mathsf{AA}^{\mathsf{RO}_{\mathsf{ts}}}[U_\sigma]$ (former has access to a strict subset of oracles compared to the latter). Hence, we get $\mathsf{TSA}[U] \subset \mathsf{TSA}[U]^{\mathsf{RO}_{\mathsf{ts}}} \subset \mathsf{AA}^{\mathsf{RO}_{\mathsf{ts}}}[U_\sigma]$. From this observation we can derive the augmented hierarchy of Definition 4.

Definition 4. *(Hierarchy of Algorithms) Fix universe U, and let $\mathsf{RO}_{\mathsf{ts}}$ be a typed-random oracle. Let Figure 3 specify functions w.r.t U. Then,*

- $\mathsf{TSA}^{\mathsf{RO}_{\mathsf{ts}}}[U]$ *or Type-safe algorithms (w.r.t $\mathsf{RO}_{\mathsf{ts}}$) is the set of all oracle algorithms that have access to oracles Π, Σ, and $\mathsf{RO}_{\mathsf{ts}}$ only.*
- $\mathsf{AA}^{\mathsf{RO}_{\mathsf{ts}}}[U_\sigma]$ *or Algebraic algorithms (w.r.t $\mathsf{RO}_{\mathsf{ts}}$) is the set of all oracle algorithms that have access to oracle $\Pi, \Sigma, \mathsf{RO}_{\mathsf{ts}}$, and $\mathcal{O}_\sigma$.*
- $\mathsf{SA}^{\mathsf{RO}_{\mathsf{ts}}}[U_\sigma]$ *or Standard algorithms is the set of all oracle algorithms that have access to all the oracles in Figure 3.*

Lifting theorem. Assume that a primitive Π is shown to be secure against *algebraic* adversaries via a reduction R to an underlying problem. If the reduction is a type-safe algorithm whenever the adversary is type-safe, and if the underlying problem is hard against type-safe adversaries, then Π is secure against *type-safe* adversaries (relevant preliminaries can be found in Appendix F).

Theorem 4. *Fix universe U and advantage metrics $\mathsf{Adv}_U^{met_1}, \mathsf{Adv}_U^{met_2}$, where $\mathsf{Adv}_U^{met_2}$ is generically hard. Let R be Δ-AGM reduction for this metric that is model preserving and efficiency preserving. Then $\mathsf{Adv}_U^{met_1}$ is generically hard.*

6.2 Lifting to security in Shoup's GGM

The final step is to lift security from the type-safe model to the Shoup-GGM model, as required by the UC lifting theorem. The full security reduction is defined in Appendix G, here we only give the theorem statement and proof intuition.

Canonical translation. We first define the canonical translation of type-safe algorithms to algorithms with access to G and RO. For simplicity, we explicitly describe this step for the case of one group; the pairing setting follows by replicating the process component-wise. The principal technicality is dealing with the random oracle: a typed query must be encoded as a specific bit string to be passed to the standard random oracle.

Definition 5 (Canonical translation from $\mathsf{TSA}^{\mathsf{RO_{ts}}}$ to Shoup's GGM). *Let $U = (S, \Pi, \Sigma)$ be a type-safe universe of computation, let (τ_1, τ_2, τ_t) be the random injections sampled by G, and (S_1', S_2', S_t') be the group sets corresponding to G. Let $\mathsf{RO_{ts}}, \mathsf{RO}$ be typed and standard random oracles respectively. Finally, let $\mathcal{A} \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$ be an algorithm described in the typed oracle framework. Its canonical translation is the algorithm $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ obtained as follows:*

- *Each $\langle \mathsf{elem} \rangle$ variable e containing a discrete log a is replaced by the label $\ell = \tau(a)$ corresponding to G.*
- *Each call to GEN is replaced by the following code: sample $\langle \mathsf{bits} \rangle g \leftarrow S_i'$.*
- *Oracle calls to OP, PAIR are replaced by the corresponding $\mathsf{G}.$OP, and $\mathsf{G}.$PAIR queries.*
- *Equality calls on $\langle \mathsf{elem} \rangle$ variables are replaced by equality tests on bitstring labels.*
- *Each typed random-oracle query $q = ((e_j, s_j))_{j \in [n]}$, where e_j contains the discrete log a_j, is replaced by a standard random-oracle query using an encoding function $\mathsf{Enc}_\tau(q) \in \{0,1\}^*$.*

We can use two encodings for typed random-oracle queries:

Weak encoding. *Define:*

$$\mathsf{Enc}_\tau^{\mathsf{wk}}((a_j, s_j)_{j \in [n]}) := \tau(a_1) \| s_1 \| \cdots \| \tau(a_n) \| s_n.$$

Strong encoding. *Fix any injective, prefix-free integer encoding $\langle \cdot \rangle$[12]. Define:*

$$\mathsf{Block}_\tau(a, s) := 0 \| \langle \mathsf{len}(\tau(a)) \rangle \| \tau(a) \| 1 \| \langle \mathsf{len}(s) \rangle \| s.$$

Then

$$\mathsf{Enc}_\tau^{\mathsf{str}}((e_j, s_j)_{j \in [n]}) := \langle n \rangle \| \mathsf{Block}_\tau(e_1, s_1) \| \cdots \| \mathsf{Block}_\tau(e_n, s_n).$$

This encoding is injective and uniquely decodable.

Strong encodings. We suggest two forms of encodings for translating typed RO queries to standard RO. The "strong" encoding is injective and uniquely decodable: given a standard RO query, one can always recover the unique $\mathsf{RO_{ts}}$ query it came from, making the equivalence between the two models straightforward.

A similar behaviour can also be enforced by using two independent random oracles, one that is implicitly meant for group elements, and the other for bit strings. Protocols can then be modified to query the two ROs and link their responses appropriately. However, this type of encoding imposes a restriction on the behaviour of the algorithm (either by forcing it to use certain—potentially sub-optimal—encodings, or by forcing a specific querying behaviour).

Weak encoding and protocol context. The "weak" encoding avoids this by simply concatenating Shoup's labels and bit strings directly, staying closer to how real-world protocols behave. As pointed out in Section 2.2

[12] The concrete choice of a specific prefix-free integer encoding is not important for our discussion here; self-delimiting codes [27] are one option.

this poses two issues: several distinct $\mathsf{RO}_{\mathsf{ts}}$ queries may map to the same RO query (*Direction 1: Shoup's to type-safety*); and, given a RO query, without further context, it is no longer possible to parse it into elements and bitstrings (*Direction 2: type-safety to Shoup*).

The additional "context" needed to handle *Direction 2* is a description of RO queries that are relevant for the challenger Ch to decide the outcome. Concretely, we assume that Ch and the primitive Π together determine a context ctx (after fixing protocol parameters). It includes the fixed set of expected query lengths $\{l_1, \ldots, l_n\}$ along with a known parsing rule for each length (e.g., for length l_i, the first x bits encode a group element, the next y bits a scalar, and so on). Any RO query of a different length can be ignored for the purpose of the security game. In practice this is a very mild assumption: for zkSNARKs compiled via Fiat-Shamir, each round's transcript has a fixed, publicly known format, so the parsing is unambiguous by construction. Moreover, a security game only requires access to these relevant queries (e.g., in the SIMEXT game, the extractor/verifier only needs these queries to complete their operation).

We now sketch how both issues are resolved in the proofs of our equivalence theorems.

Theorem 5. *Let Ch be a single-stage game for a primitive Π in the type-safe oracle framework, and let (Ch', Π') be its canonical Shoup-GGM translation (Definition 5) using strong encoding. If Π satisfies Ch against all PPT adversaries $\mathcal{A} \in \mathsf{TSA}^{\mathsf{RO}_{\mathsf{ts}}}[U]$, then Π' satisfies Ch' against all PPT adversaries with oracle access to G and RO. When using weak encoding, the same implication holds if (Ch, Π) determine a context ctx as described above.*

The proof is by contrapositive and follows the blueprint of [57]. Given a Shoup-GGM adversary $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ against Ch', we construct a type-safe adversary $\mathcal{A}^{\mathsf{RO}_{\mathsf{ts}}}$ against Ch. $\mathcal{A}$ runs $\mathcal{B}$ internally, simulating its access to $(\mathsf{G}, \mathsf{RO})$ and interaction with Ch' as follows: $\mathcal{A}$ maintains a local table mapping $\langle\mathsf{elem}\rangle$ variables to labels (corresponding to G). On receiving an $\langle\mathsf{elem}\rangle$ variable from Ch, it samples a fresh label, stores it in its table, and forwards the label to $\mathcal{B}$. On receiving a label message from $\mathcal{B}$ for the challenger, it looks up the corresponding $\langle\mathsf{elem}\rangle$ variable to forward to Ch. For any group query to G is also treated in the same way before forwarding to oracle defined in the universe U. Whenever $\mathcal{B}$ submits a label, for which there is no corresponding $\langle\mathsf{elem}\rangle$ variable stored in the local table, $\mathcal{A}$ samples a fresh $\langle\mathsf{elem}\rangle$ variable (this is assuming that the label encoding is exactly equal to the size of the group; a more general case can be handled by sampling $\langle\mathsf{elem}\rangle$ with a certain failure probability). Since a fresh label is unrelated to all previous labels, the discrete log associated with it is also unrelated; hence, sampling an $\langle\mathsf{elem}\rangle$ variable at random suffices for an indistinguishable view.

Random oracle queries are handled as follows: if $\mathcal{B}$ makes a query of length specified in ctx, then $\mathcal{A}$ parses it according to the parsing information in ctx, looks up $\langle\mathsf{elem}\rangle$ variables accordingly to forward the query to $\mathsf{RO}_{\mathsf{ts}}$. Queries of all other lengths are handled locally via lazy sampling.

Since all challenger messages and relevant RO queries have been forwarded by $\mathcal{A}$, if Ch' accepts/rejects then so does Ch. Finally, since $\mathcal{A}$ generates an indistinguishable view for $\mathcal{B}$, $\mathcal{A}$ has the same winning advantage.

Theorem 6. *Let Ch' be the canonical Shoup-GGM translation of a type-safe game Ch (Definition 5) using strong encoding. If Ch' is secure against adversaries with oracle access to G and RO, then Ch is secure in the $\mathsf{TSA}^{\mathsf{RO}_{\mathsf{ts}}}[U]$ model. When using weak encoding, the same implication holds but with a negligible loss.*

For completeness, we also show the implication in the other direction. Using strong encoding the translation leads to an algorithm with identical behaviour and hence preserves security guarantees. With weak encoding, we must rule out possible collisions in the translated algorithm.

7 Implications and Open Questions

Implications of our framework. We sketch in Appendix H how we conclude UC security for two important proof systems – Plonk [31] and Marlin [23].

The result follows from typing the SIMEXT result established by [29] in the AGM + non-programmable ROM where they show how to infer the aforementioned property for zkSNARKs obtained from a PIOP

compiled using KZG [41]. By analyzing three core theorems from this paper, we identify a sequence of type-safe reductions to the underlying generically hard problem. Then applying type-safe AGM to type-safe GGM lifting theorem (Theorem 4), type-safe GGM to Shoup's GGM implication (Theorem 5), and finally the UC lifting theorem(Theorem 3), we conclude UC security for Plonk and Marlin.

Future Work. We find a few interesting directions for extensions. First, a natural question is to extend the formalization in this paper to capture security against adaptive adversaries and, since we are in the GGM, achieve unconditional security (as is done in [19,22]). Second, we believe our framework can be applied to the class of transparent, discrete-log based proof systems like [14,54] in order to show their UC security. Finally, a more interesting, and practically relevant question is to cover the scenario where group-based zkSNARKs can safely reuse parts of the CRS. This question is motivated by practical scenarios where a KZG-based zkSNARK might use a powers-of-tau CRS that is also used by other zkSNARKs or cryptographic primitives (e.g., vector commitments). This scenario escapes our current modelling since it would need to work with a global CRS functionality where, however, the elements cannot belong to any particular session. This extension might require redefining observability for the global-GGM functionality as well as identifying conditions in the zkSNARK design under which such a reuse of the CRS is safe.

References

1. Abdalla, M., Barbosa, M., Katz, J., Loss, J., Xu, J.: Algebraic adversaries in the universal composability framework. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part III. Lecture Notes in Computer Science, vol. 13092, pp. 311–341. Springer, Cham, Switzerland, Singapore (Dec 6–10, 2021). https://doi.org/10.1007/978-3-030-92078-4_11
2. Abdolmaleki, B., Glaeser, N., Ramacher, S., Slamanig, D.: Circuit-succinct universally-composable NIZKs with updatable CRS. In: CSF 2024: IEEE 37th Computer Security Foundations Symposium. pp. 527–542. IEEE Computer Society Press, Enschede, The Netherlands (Jul 8–12, 2024). https://doi.org/10.1109/CSF61375.2024.00006
3. Abdolmaleki, B., Ramacher, S., Slamanig, D.: Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020: 27th Conference on Computer and Communications Security. pp. 1987–2005. ACM Press, Virtual Event, USA (Nov 9–13, 2020). https://doi.org/10.1145/3372297.3417228
4. Aranha, D.F., Bennedsen, E.M., Campanelli, M., Ganesh, C., Orlandi, C., Takahashi, A.: ECLIPSE: Enhanced compiling method for Pedersen-committed zkSNARK engines. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 13177, pp. 584–614. Springer, Cham, Switzerland, Virtual Event (Mar 8–11, 2022). https://doi.org/10.1007/978-3-030-97121-2_21
5. Badertscher, C., Campanelli, M., Ciampi, M., Russo, L., Siniscalchi, L.: Universally composable SNARKs with transparent setup without programmable random oracle. In: Kalai, Y.T., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025, Part VII. Lecture Notes in Computer Science, vol. 16006, pp. 225–258. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2025). https://doi.org/10.1007/978-3-032-01907-3_8
6. Badertscher, C., Canetti, R., Hesse, J., Tackmann, B., Zikas, V.: Universal composition with global subroutines: Capturing global setup within plain UC. In: Pass, R., Pietrzak, K. (eds.) TCC 2020: 18th Theory of Cryptography Conference, Part III. Lecture Notes in Computer Science, vol. 12552, pp. 1–30. Springer, Cham, Switzerland, Durham, NC, USA (Nov 16–19, 2020). https://doi.org/10.1007/978-3-030-64381-2_1
7. Baghery, K., Kohlweiss, M., Siim, J., Volkhov, M.: Another look at extraction and randomization of groth's zk-SNARK. In: Borisov, N., Díaz, C. (eds.) FC 2021: 25th International Conference on Financial Cryptography and Data Security, Part I. Lecture Notes in Computer Science, vol. 12674, pp. 457–475. Springer Berlin Heidelberg, Germany, Virtual Event (Mar 1–5, 2021). https://doi.org/10.1007/978-3-662-64322-8_22
8. Baghery, K., Sedaghat, M.: Tiramisu: Black-box simulation extractable NIZKs in the updatable CRS model. In: Conti, M., Stevens, M., Krenn, S. (eds.) CANS 21: 20th International Conference on Cryptology and Network Security. Lecture Notes in Computer Science, vol. 13099, pp. 531–551. Springer, Cham, Switzerland, Vienna, Austria (Dec 13–15, 2021). https://doi.org/10.1007/978-3-030-92548-2_28
9. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 701–732. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2019). https://doi.org/10.1007/978-3-030-26954-8_23

10. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part I. Lecture Notes in Computer Science, vol. 11476, pp. 103–128. Springer, Cham, Switzerland, Darmstadt, Germany (May 19–23, 2019). https://doi.org/10.1007/978-3-030-17653-2_4
11. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B: 14th Theory of Cryptography Conference, Part II. Lecture Notes in Computer Science, vol. 9986, pp. 31–60. Springer Berlin Heidelberg, Germany, Beijing, China (Oct 31 – Nov 3, 2016). https://doi.org/10.1007/978-3-662-53644-5_2
12. Bobolz, J., Farshim, P., Kohlweiss, M., Takahashi, A.: The brave new world of global generic groups and UC-secure zero-overhead SNARKs. In: Boyle, E., Mahmoody, M. (eds.) TCC 2024: 22nd Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 15364, pp. 90–124. Springer, Cham, Switzerland, Milan, Italy (Dec 2–6, 2024). https://doi.org/10.1007/978-3-031-78011-0_4
13. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 327–357. Springer Berlin Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-5_12
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press, San Francisco, CA, USA (May 21–23, 2018). https://doi.org/10.1109/SP.2018.00020
15. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part III. Lecture Notes in Computer Science, vol. 13092, pp. 3–33. Springer, Cham, Switzerland, Singapore (Dec 6–10, 2021). https://doi.org/10.1007/978-3-030-92078-4_1
16. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security. pp. 2075–2092. ACM Press, London, UK (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3339820
17. Campanelli, M., Ganesh, C., Khoshakhlagh, H., Siim, J.: Impossibilities in succinct arguments: Black-box extraction and more. In: El Mrabet, N., De Feo, L., Duquesne, S. (eds.) AFRICACRYPT 23: 14th International Conference on Cryptology in Africa. Lecture Notes in Computer Science, vol. 14064, pp. 465–489. Springer, Cham, Switzerland, Sousse, Tunisia (Jul 19–21, 2023). https://doi.org/10.1007/978-3-031-37679-5_20
18. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science. pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001). https://doi.org/10.1109/SFCS.2001.959888
19. Canetti, R., Chen, M.: Universally composable succinct vector commitments and applications. In: Applebaum, B., Lin, H.R. (eds.) TCC 2025: 23rd Theory of Cryptography Conference, Part IV. Lecture Notes in Computer Science, vol. 16271, pp. 642–674. Springer, Cham, Switzerland, Aarhus, Denmark (Dec 1–5, 2025). https://doi.org/10.1007/978-3-032-12290-2_21
20. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th Annual ACM Symposium on Theory of Computing. pp. 494–503. ACM Press, Montréal, Québec, Canada (May 19–21, 2002). https://doi.org/10.1145/509907.509980
21. Canetti, R., Sarkar, P., Wang, X.: Triply adaptive UC NIZK. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part II. Lecture Notes in Computer Science, vol. 13792, pp. 466–495. Springer, Cham, Switzerland, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22966-4_16
22. Chiesa, A., Fenzi, G.: zkSNARKs in the ROM with unconditional UC-security. In: Boyle, E., Mahmoody, M. (eds.) TCC 2024: 22nd Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 15364, pp. 67–89. Springer, Cham, Switzerland, Milan, Italy (Dec 2–6, 2024). https://doi.org/10.1007/978-3-031-78011-0_3
23. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 738–768. Springer, Cham, Switzerland, Zagreb, Croatia (May 10–14, 2020). https://doi.org/10.1007/978-3-030-45721-1_26
24. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) Advances in Cryptology – CRYPTO 2001. Lecture Notes in Computer Science, vol. 2139, pp. 566–598. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2001). https://doi.org/10.1007/3-540-44647-8_33
25. Dellepere, M., Mishra, P., Shirzad, A.: Garuda and pari: Faster and smaller SNARKs via equifficient polynomial commitments. Cryptology ePrint Archive, Report 2024/1245 (2024), https://eprint.iacr.org/2024/1245

26. Dent, A.W.: Adapting the weaknesses of the random oracle model to the generic group model. In: Zheng, Y. (ed.) Advances in Cryptology – ASIACRYPT 2002. Lecture Notes in Computer Science, vol. 2501, pp. 100–109. Springer Berlin Heidelberg, Germany, Queenstown, New Zealand (Dec 1–5, 2002). https://doi.org/10.1007/3-540-36178-2_6
27. Elias, P.: Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory **21**(2), 194–203 (1975). https://doi.org/10.1109/TIT.1975.1055349
28. Faonio, A., Fiore, D., Kohlweiss, M., Russo, L., Zajac, M.: From polynomial IOP and commitments to non-malleable zkSNARKs. In: Rothblum, G.N., Wee, H. (eds.) TCC 2023: 21st Theory of Cryptography Conference, Part III. Lecture Notes in Computer Science, vol. 14371, pp. 455–485. Springer, Cham, Switzerland, Taipei, Taiwan (Nov 29 – Dec 2, 2023). https://doi.org/10.1007/978-3-031-48621-0_16
29. Faonio, A., Fiore, D., Russo, L.: Real-world universal zkSNARKs are non-malleable. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024: 31st Conference on Computer and Communications Security. pp. 3138–3151. ACM Press, Salt Lake City, UT, USA (Oct 14–18, 2024). https://doi.org/10.1145/3658644.3690351
30. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96881-0_2
31. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
32. Ganesh, C., Kondi, Y., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Witness-succinct universally-composable SNARKs. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part II. Lecture Notes in Computer Science, vol. 14005, pp. 315–346. Springer, Cham, Switzerland, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30617-4_11
33. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part II. Lecture Notes in Computer Science, vol. 13276, pp. 397–426. Springer, Cham, Switzerland, Trondheim, Norway (May 30 – Jun 3, 2022). https://doi.org/10.1007/978-3-031-07085-3_14
34. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd Annual ACM Symposium on Theory of Computing. pp. 99–108. ACM Press, San Jose, CA, USA (Jun 6–8, 2011). https://doi.org/10.1145/1993636.1993651
35. Ghoshal, A., Tessaro, S.: Tight state-restoration soundness in the algebraic group model. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 64–93. Springer, Cham, Switzerland, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84252-9_3
36. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) Advances in Cryptology – ASIACRYPT 2006. Lecture Notes in Computer Science, vol. 4284, pp. 444–459. Springer Berlin Heidelberg, Germany, Shanghai, China (Dec 3–7, 2006). https://doi.org/10.1007/11935230_29
37. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 305–326. Springer Berlin Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-5_11
38. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: Vaudenay, S. (ed.) Advances in Cryptology – EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 339–358. Springer Berlin Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_21
39. Jaeger, J., Mohan, D.I.: Generic and algebraic computation models: When AGM proofs transfer to the GGM. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part V. Lecture Notes in Computer Science, vol. 14924, pp. 14–45. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68388-6_2
40. Jager, T., Schwenk, J.: On the equivalence of generic group models. In: International Conference on Provable Security. pp. 200–209. Springer (2008)
41. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) Advances in Cryptology – ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 177–194. Springer Berlin Heidelberg, Germany, Singapore (Dec 5–9, 2010). https://doi.org/10.1007/978-3-642-17373-8_11

42. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th Annual ACM Symposium on Theory of Computing. pp. 723–732. ACM Press, Victoria, BC, Canada (May 4–6, 1992). https://doi.org/10.1145/129712.129782
43. Kohlweiss, M., Pancholi, M., Takahashi, A.: How to compile polynomial IOP into simulation-extractable SNARKs: A modular approach. In: Rothblum, G.N., Wee, H. (eds.) TCC 2023: 21st Theory of Cryptography Conference, Part III. Lecture Notes in Computer Science, vol. 14371, pp. 486–512. Springer, Cham, Switzerland, Taipei, Taiwan (Nov 29 – Dec 2, 2023). https://doi.org/10.1007/978-3-031-48621-0_17
44. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., shelat, a., Shi, E.: C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015), https://eprint.iacr.org/2015/1093
45. Lipmaa, H.: Polymath: Groth16 is not the limit. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part X. Lecture Notes in Computer Science, vol. 14929, pp. 170–206. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68403-6_6
46. Lysyanskaya, A., Rosenbloom, L.N.: Efficient and universally composable non-interactive zero-knowledge proofs of knowledge with security against adaptive corruptions. Cryptology ePrint Archive, Report 2022/1484 (2022), https://eprint.iacr.org/2022/1484
47. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable Σ-protocols in the global random-oracle model. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022: 20th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 13747, pp. 203–233. Springer, Cham, Switzerland, Chicago, IL, USA (Nov 7–10, 2022). https://doi.org/10.1007/978-3-031-22318-1_8
48. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security. pp. 2111–2128. ACM Press, London, UK (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3339817
49. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. Lecture Notes in Computer Science, vol. 3796, pp. 1–12. Springer Berlin Heidelberg, Germany, Cirencester, UK (Dec 19–21, 2005). https://doi.org/10.1007/11586821_1
50. Micali, S.: CS proofs (extended abstracts). In: 35th Annual Symposium on Foundations of Computer Science. pp. 436–453. IEEE Computer Society Press, Santa Fe, NM, USA (Nov 20–22, 1994). https://doi.org/10.1109/SFCS.1994.365746
51. Micali, S.: Computationally sound proofs. SIAM Journal on Computing **30**(4), 1253–1298 (2000). https://doi.org/10.1137/S0097539795284959, https://doi.org/10.1137/S0097539795284959
52. National Institute of Standards and Technology: Digital Signature Standard (DSS). Tech. Rep. FIPS 186-5, National Institute of Standards and Technology (2023). https://doi.org/10.6028/NIST.FIPS.186-5
53. Nechaev, V.I.: Complexity of a determinate algorithm for the discrete logarithm. Mathematical Notes **55**(2), 165–172 (Feb 1994). https://doi.org/10.1007/BF02113297
54. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 704–737. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56877-1_25
55. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) Advances in Cryptology – EUROCRYPT'97. Lecture Notes in Computer Science, vol. 1233, pp. 256–266. Springer Berlin Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997). https://doi.org/10.1007/3-540-69053-0_18
56. Wang, T., Zhang, C., Zhou, H.S., Wang, X., Ji, K., Jia, Z., Lin, L., Wei, C., Yan, Y., Ren, K., Chen, C.: Hashing in generic groups: Completing the AGM-to-GGM transfer. Cryptology ePrint Archive, Paper 2026/466 (2026), https://eprint.iacr.org/2026/466
57. Zhandry, M.: To label, or not to label (in generic groups). In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part III. Lecture Notes in Computer Science, vol. 13509, pp. 66–96. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 15–18, 2022). https://doi.org/10.1007/978-3-031-15982-4_3
58. Zhang, C., Zhou, H.S., Katz, J.: An analysis of the algebraic group model. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part IV. Lecture Notes in Computer Science, vol. 13794, pp. 310–322. Springer, Cham, Switzerland, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22972-5_11

A Additional Related Works

UC-AGM. Abdalla et al. [1] present a UC-AGM framework modelling the AGM behaviour in UC. However, this model fails to capture extractable non-interactive primitives such as zkSNARKs. The fundamental issue is that the UC-AGM does not require the adversary to output an explanation alongside the group elements when providing inputs to honest parties (the verifier in this case). Whereas this explanation is needed by the simulator to extract on a verify query. This asymmetry makes UC-AGM unsuitable for zkSNARKs. The full version of [39] informally describes how security in the AGM could imply security in the UC-AGM framework as well. Apart from being an informal discussion, it also only considers UC-AGM which is not suited for zkSNARKs.

UC secure NIZKs. Several non-succinct NIZKs have been proven to be UC secure. One common approach is via compiling a proof system in the stand-alone setting, to a UC one by essentially encrypting the witness as part of the proof. SLE can be obtained by simply decrypting this ciphertext. A series of works follow variants of this recipe [2, 3, 8, 24, 44]. Others, compile Σ protocols using Fischlin's transform to achieve SLE [46, 47]. The NIZK construction from [24] was shown to be UC secure in [20]. Finally, the works of [21, 36, 38] give specific constructions of UC secure NIZKs against various notions of adaptive adversaries.

B Preliminaries: Stand-Alone Properties

B.1 Symbolic Shoup's generic group model

Figure 4: Symbolic GGM oracle: s-G

INIT()
1: **for** $j \in \{1, 2, t\}$: **do**
2: $\tau_i \leftarrow \{\}$
3: TAU$(i, 1)$
4: **end for**

CANONICALGEN(i)
1: **return** TAU$(i, 1)$

OP$(i, (g_1, a_1), (g_2, a_2))$
1: **assert**$(g_1, g_2, a_1, a_2) \in S_i^2 \times \mathbb{Z}_p^2$
2: **for** $j \in \{1, 2\}$: **do**
3: TOUCH(i, g_j)
4: **end for**
5: $\mathsf{f} := a_1 \tau_i^{-1}(g_1) + a_2 \tau_i^{-1}(g_2)$
6: $h \leftarrow$ TAU(i, f)
7: **return** h

TOUCH(i, g)
1: **if** $g \notin \mathsf{img}(\tau_i)$ **then**
2: Initialize fresh variable X
3: $\tau_i(\mathsf{X}) := g$
4: **end if**

TAU(i, f) //Internal function
1: **if** $\tau_i(\mathsf{f}) = \perp$ **then**
2: $\tau_i(\mathsf{f}) \xleftarrow{\$} S_i \setminus \mathsf{img}(\tau_i)$
3: **end if**
4: **return** $\tau_i(\mathsf{f})$

PAIR(g_1, g_2)
1: **assert**$(g_1, g_2) \in S_1 \times S_2$
2: **for** $i \in \{1, 2\}$ **do**
3: TOUCH(i, g_i)
4: **end for**
5: $\mathsf{f} := \tau_1^{-1}(g_1) \cdot \tau_2^{-1}(g_2)$
6: $h \leftarrow$ TAU(t, f)
7: **return** h

B.2 Succinct non-interactive zero knowledge argument of knowledge

Definition 6 (Completeness). *For all PPT adversaries $\mathcal{A}$, for all $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{c|l} (\mathsf{i}, x, w) \in \mathcal{R} & \mathsf{pp} \leftarrow \mathit{Gen}(1^\lambda);\ \ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathit{Setup}(\mathsf{pp}); \\ \implies V(\mathsf{crs}, x, \pi) = 1 & (\mathsf{i}, x, w) \leftarrow \mathcal{A}(\mathsf{crs});\ \ \pi \leftarrow P(\mathsf{crs}, x, w) \end{array}\right] = 1.$$

All algorithms $(\mathcal{A}, \mathit{Setup}, P, V)$ have oracle access to RO, G.

Definition 7 (Non-Interactive Zero Knowledge). *Let $\Pi = (\mathit{Setup}, P, V)$ be a non-interactive argument of knowledge for relation $\mathcal{R}$. Π satisfies* NIZK *in the non-programmable random oracle and generic group model, if there exists an efficient simulator S such that for all PPT distinguishers $\mathcal{A}$ the following advantage is negligible in λ,*

$$\mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{NIZK\text{-}0}_{\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{NIZK\text{-}1}_{\mathcal{A}}(1^\lambda) = 1\right]\right|$$

where NIZK-0 *and* NIZK-1 *are defined in Fig. 5.*

Figure 5: Security game for NIZK

$\underline{\mathsf{NIZK\text{-}0}_{\mathcal{A}}(1^\lambda)}$

1: $\mathsf{pp} \leftarrow Gen(1^\lambda)$
2: $(\mathsf{crs}, \mathsf{td}) \leftarrow Setup(\mathsf{pp})$
3: $b \leftarrow \mathcal{A}^{\mathsf{RO},\mathsf{G},P,V}(\mathsf{crs})$
4: **return** b

$\underline{\mathsf{NIZK\text{-}1}_{\mathcal{A}}(1^\lambda)}$

1: $\mathsf{pp} \leftarrow Gen(1^\lambda)$
2: $(\mathsf{crs}, \mathsf{td}) \leftarrow Setup(\mathsf{pp})$
3: $\mathcal{Q}_S := \{\}$
4: $\mathsf{st} := (\mathsf{td}, \mathcal{Q}_S)$
5: $b \leftarrow \mathcal{A}^{\mathsf{RO},\mathsf{G},S_1,V}(\mathsf{crs})$
6: **return** b

$\underline{S_1(x, w)}$

1: **assert**$(x, w) \in \mathcal{R}$
2: $(\pi, \mathsf{st}) \leftarrow S^{\mathsf{RO},\mathsf{G}}(\mathsf{st}, \mathsf{crs}, x)$
3: $\mathcal{Q}_S \leftarrow \mathcal{Q}_S : (x, \pi)$
4: **return** π

B.3 Computing group representation

Algorithm 1: FindRep

$\underline{\textsc{FindRep}(i, h, B, \mathsf{tr})}$

1: **assert** $i \in \{1, 2\}$
2: Parse $B := (B_1, \ldots, B_n)$
3: Initially $Rep[h] = 0^n, \forall h \in S_i$
4: Initialize $Rep[B_j][j] := 1$, for $j \in [n]$
5: **for** $op = (\textsc{Op}, i, g_1, g_2, a_1, a_2, h) \in \mathsf{tr}_{\mathcal{G}}$ **do**:
6: $\quad Rep[h] := a_1 \cdot Rep[g_1] + a_2 \cdot Rep[g_2]$
7: **end for**
8: **return** $Rep[h]$

$\underline{\textsc{FindRep}(t, h, B, \mathsf{tr})}$

1: Parse $B = (B^{(1)}, B^{(2)}, B^{(t)})$
2: **for** $i \in \{1, 2\}$ **do**,
3: $\quad$ Parse $B^{(i)} = (B_1^{(i)}, \ldots, B_{n_i}^{(i)})$
4: $\quad$ Initially $Rep_i[h] \leftarrow 0^{n_i}$
5: $\quad$ Initialize $Rep[B_j^{(i)}][j] := 1$, for $j \in [n_i]$
6: **end for**
7: Parse $B^{(t)} = (B_1^{(t)}, \ldots, B_{n_t}^{(t)})$
8: Initially $Rep_t[h] \leftarrow 0^{n_t + n_1 \cdot n_2}$
9: Initialize $Rep[B_j^{(i)}][j] := 1$, for $j \in [n_t + n_1 \cdot n_2]$
10: **for** $op \in \mathsf{tr}$: **do**
11: $\quad$ **if** $op = (\textsc{Op}, i, g_1, g_2, a_1, a_2, h) \in \mathsf{tr}$ **then**
12: $\quad\quad Rep[h] := a_1 \cdot Rep[g_1] + a_2 \cdot Rep[g_2]$
13: $\quad$ **end if**
14: $\quad$ **if** $op = (\textsc{Pair}, g_1, g_2, h) \in \mathsf{tr}$ **then**
15: $\quad\quad Rep_t[h] \leftarrow (0^{n_t}, Rep_1[g_1] \otimes Rep_2[g_2])$
16: $\quad //c \otimes d := (c_k \cdot d_\ell)_{k \in [n_1], \ell \in [n_2]}$
17: $\quad$ **end if**
18: **end for**
19: **return** $Rep_t[h]$

B.4 Internal Element Hiding

Definition 8 (Internal Element Hiding). *Let $\mathcal{A}$ be a PPT algorithm with oracle access to the Shoup generic group oracle G. Let $\mathsf{Elems}_{\mathcal{A}}^{(i)}$ denote the set of group labels returned by G to $\mathcal{A}$ during its ith execution, and let $\mathsf{Pub}^{(i)}$ be the set of group labels appearing in the public input and output of the ith execution. Define the hidden internal labels of this execution as*

$$\mathsf{Hid}_{\mathcal{A}}^{(i)} := \mathsf{Elems}_{\mathcal{A}}^{(i)} \setminus \mathsf{Pub}^{(i)}.$$

The game $\mathsf{HIL}_{\mathcal{B}}^{\mathcal{A}}(1^\lambda)$ is defined as follows.

1. *The challenger Ch initializes public parameters, and any public setup information. It is responsible for answering* G *queries (via lazy sampling). Invokes* $\mathcal{B}$ *on the initial parameters, and answers any* G *query made by* $\mathcal{B}$*.* $\mathcal{B}$ *may issue* G *queries and adaptively invoke* $\mathcal{A}$ t *times with inputs of its choice.*
2. *Ch initializes an empty sets* $\mathsf{Elem}_{\mathcal{A}}, \mathsf{Pub}_{\mathcal{A}}, \mathsf{View}_{\mathcal{B}}, \mathsf{Query}_{\mathcal{B}}$*.*
3. *On a group operation query with inputs* g_1, g_2 *and output* g_3*: Ch updates* $\mathsf{View}_{\mathcal{B}} \leftarrow \mathsf{View}_{\mathcal{B}} \cup g_3$*, and* $\mathsf{Query}_{\mathcal{B}} \leftarrow \mathsf{Query}_{\mathcal{B}} \cup \{g_1, g_2\}$*.*
4. *On a* i*-th invocation of* $\mathcal{A}$ *by* $\mathcal{B}$ *with input* u *(where* $i \leq t$*): Ch intercepts the query, runs* $\mathcal{A}(u)$ *internally and updates sets* $\mathsf{Elem}_{\mathcal{A}} \leftarrow \mathsf{Elem}_{\mathcal{A}}^{(i)} \cup \mathsf{Elem}_{\mathcal{A}}$*,* $\mathsf{Pub}_{\mathcal{A}}^{(i)} \leftarrow \mathsf{Pub}_{\mathcal{A}}$*,* $\mathsf{View}_{\mathcal{B}} \leftarrow \mathsf{View}_{\mathcal{B}} \cup \mathsf{Pub}^{(i)}$*. If* $\exists g \in \mathsf{Query}_{\mathcal{B}}$*, such that* $g \in \mathsf{Elem}_{\mathcal{A}} \setminus \mathsf{View}_{\mathcal{B}}$*, Ch outputs* 1*. Else, continue and forward the result to* $\mathcal{B}$*.*
5. *After* t *invocations, output* 0*.*

We say that $\mathcal{A}$ *satisfies internal element hiding if for every PPT adversary* $\mathcal{B}$*,*

$$\Pr\left[\mathsf{HIL}_{\mathcal{B}}^{\mathcal{A}}(1^{\lambda}) = 1\right] \leq \mathsf{negl}(\lambda).$$

Theorem 7 (Type safety implies internal element hiding). *Let* $\mathcal{A}$ *be a PPT type-safe algorithm with oracle access to a Shoup generic group oracle* G *and to a type-safe random oracle* $\mathsf{RO}_{\mathsf{ts}}$*. Assume that* $\mathcal{A}$ *has no access to an encoding oracle*

$$\sigma : \langle \mathit{elem} \rangle \to \langle \mathit{bits} \rangle$$

and no access to a decoding oracle

$$\sigma^{-1} : \langle \mathit{bits} \rangle \to \langle \mathit{elem} \rangle.$$

Then $\mathcal{A}$ *satisfies internal element hiding. More precisely, for every PPT adversary* $\mathcal{B}$ *making at most* $Q_{\mathcal{B}}$ *group queries and for* $\mathcal{A}$ *making at most a total of* $Q_{\mathcal{A}}$ *group queries, we have:*

$$\Pr\left[\mathsf{HIL}_{\mathcal{B}}^{\mathcal{A}}(1^{\lambda}) = 1\right] \leq \frac{Q_{\mathcal{B}} Q_{\mathcal{A}}}{p - Q_{\mathcal{A}} - Q_{\mathcal{B}}} + \mathsf{negl}(\lambda),$$

Proof. We prove that the adversary can win the internal-element-hiding game only by guessing a label that has not appeared in its view.

Let $\mathsf{View}_{\mathcal{B}}$ be the set of group labels that have been given to $\mathcal{B}$ so far. We use the corrected winning condition: when $\mathcal{B}$ makes a group-oracle query, the game outputs 1 only if the query contains some $\ell \in \mathsf{Elem}_{\mathcal{A}} \setminus \mathsf{View}_{\mathcal{B}}$. Thus, if $\mathcal{B}$ first obtains ℓ as the output of one of its own group-oracle queries, then ℓ is added to $\mathsf{View}_{\mathcal{B}}$, and its later use does not cause the game to output 1.

Fix an arbitrary adversary $\mathcal{B}$. Consider any point in the game immediately before $\mathcal{B}$ makes a group-oracle query. Let Bad be the event that this query contains some label $\ell \in \mathsf{Elem}_{\mathcal{A}} \setminus \mathsf{View}_{\mathcal{B}}$. We bound the probability of Bad.

By type safety, $\mathcal{A}$ has no access to σ and thus cannot convert a group element into its concrete bit-string encoding. Therefore, the only ways in which a group label can become known to $\mathcal{B}$ are the following:

1. the label appears in the public input.
2. the label appears in the public group output of $\mathcal{A}$.
3. the label is returned to $\mathcal{B}$ by G as the answer to one of $\mathcal{B}$'s own group-oracle queries.

Labels of types (1) and (2) are included in Pub, hence are not hidden. Labels of type (3) are included in $\mathsf{View}_{\mathcal{B}}$. Consequently, any label in $\mathsf{Elem}_{\mathcal{A}} \setminus \mathsf{View}_{\mathcal{B}}$ has not appeared as a bit string anywhere in $\mathcal{B}$'s view.

Since $\mathcal{A}$ has no access to σ, the bit-string output of $\mathcal{A}$ may only depend on non-group data or on randomness. Now condition on the entire view of $\mathcal{B}$ before its next group query, including all public parameters, all outputs of $\mathcal{A}$, and all previous group-oracle replies. Let U_i be the set of unused labels in the relevant group S_i, namely labels that have not yet appeared in $\mathsf{View}_{\mathcal{B}}$ or as public group outputs. Since G injection is sampled uniformly, every hidden internal label in $\mathsf{Elem}_{\mathcal{A}} \setminus \mathsf{View}_{\mathcal{B}}$ is uniformly distributed over U_i. Because $\mathcal{A}$ and $\mathcal{B}$ together make at most $Q_{\mathcal{A}} + Q_{\mathcal{B}}$ group-oracle queries, we have:

$$|U_i| \geq p - Q_{\mathcal{A}} - Q_{\mathcal{B}}.$$

Let m be the number of hidden internal labels currently in $\mathsf{Elem}_{\mathcal{A}} \setminus \mathsf{View}_{\mathcal{B}}$. Since $\mathcal{A}$ makes at most $Q_{\mathcal{A}}$ group-oracle queries, we have $m \leq Q_{\mathcal{A}}$. For any concrete label ℓ that $\mathcal{B}$ writes in its next group-oracle query but has not previously received, the probability that ℓ equals one of these hidden labels is therefore at most

$$\frac{m}{p - Q_{\mathcal{A}} - Q_{\mathcal{B}}} \leq \frac{Q_{\mathcal{A}}}{p - Q_{\mathcal{A}} - Q_{\mathcal{B}}}.$$

Taking a union bound over the at most $Q_{\mathcal{B}}$ group-oracle queries made by $\mathcal{B}$, we obtain

$$\Pr[\mathsf{HIL}^{\mathcal{A}}_{\mathcal{B}}(1^{\lambda}) = 1] \leq \frac{Q_{\mathcal{B}} Q_{\mathcal{A}}}{p - Q_{\mathcal{A}} - Q_{\mathcal{B}}}.$$

Which is negligible for a large enough p and thus $\mathcal{A}$ satisfies internal element hiding.

C Preliminaries: Universal Composability

Syntax for ideal functionalities and protocols. As [12] we will follow the UCGS [6] formalism. In both worlds, in a session *sid*, the environment $\mathcal{Z}$ initializes a number of honest parties and an adversary $\mathcal{A}$ (i.e. static corruption). It provides inputs to all, reads outputs of all parties, and has full access to the view of $\mathcal{A}$. $\mathcal{G}$ is a global functionality available to all parties, $\mathcal{A}$ and the environment $\mathcal{Z}$, and can be directly queried.

Real World: The execution takes place in the $(\mathcal{F}_{\mathsf{setup}}, \mathcal{G})$-hybrid model. $\mathcal{F}_{\mathsf{setup}}$ is a local functionality available only to parties within the session, while $\mathcal{G}$ is a global functionality as above. During the execution, $\mathcal{Z}$ can issue PROVE, VERIFY commands. On receiving $(\text{PROVE}_{sid}, x, w)$, an honest party executes the SNARK proving algorithm P, and on receiving $(\text{VERIFY}_{sid}, x, \pi)$, an honest verifier executes the SNARK verification algorithm V. In addition to this, $\mathcal{Z}$ can query $\mathcal{G}$ on OBSERVE to receive the set of illegal queries.

Ideal World: In this experiment, honest parties in *sid* forward their inputs to $\mathcal{F}_{\mathsf{nizk}}$. Access to $\mathcal{A}$, $\mathcal{G}$ remains the same in both worlds.

$\mathsf{EXEC}^{\mathcal{F}}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$ denotes the distribution of $\mathcal{Z}$'s binary output after an execution Π in presence of $\mathcal{A}$ and given access to (local or global) functionality $\mathcal{F}$, where $\lambda \in \mathbb{N}$ is the security parameter and $z \in \{0,1\}^*$ is an auxiliary input to $\mathcal{Z}$. $\{\mathsf{EXEC}^{\mathcal{F}}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$ denotes the ensemble of such random variables.

Functionality 4: $\mathcal{G}_{\mathsf{RO}}$

$\mathcal{G}_{\mathsf{RO}}$ is parametrized by the output length $\ell(\lambda)$.

QUERY: Upon receiving a query (QUERY, x), from some party $\mathcal{P} = (pid, sid)$ or from the adversary do:

1: Look up v if there is a pair (x, v) for some $v \in \{0,1\}^{\ell(\lambda)}$ in the (initially empty) list $\mathcal{Q}_{\mathsf{RO}}$ of past queries. Else, choose uniformly $v \in \{0,1\}^{\ell(\lambda)}$ and store the pair (x, v) in $\mathcal{Q}_{\mathsf{RO}}$.
2: Parse x as (s, x'). If $sid \neq s$ then add (s, x', v) to the (initially empty) observable set Ob_{RO}
3: Return v to $\mathcal{P}$.

OBSERVE: Upon receiving a request $(\text{OBSERVE}, sid)$, return the list Ob_{RO} restricted to tuples with *sid*.

Functionality 5: $\mathcal{G}$

Parameters of the functionality:

- A prime number p
- Sets S_1, S_2, S_t with $|S_i| = p$ for all $i \in \{1, 2, t\}$.

Internal state maintained by the functionality:

- τ_i: Random encoding $\mathbb{Z}_p \to S_i$, for $i \in \{1, 2, t\}$.
- $\mathsf{X}_{i,sid}$: Initially empty list. This will record formal variables defined for session *sid*. X_{sid} denotes the concatenation $\mathsf{X}_{1,sid} : \mathsf{X}_{2,sid} : \mathsf{X}_{t,sid}$.
- R_i: Initially empty sets of polynomial. For a group element $g \in S_i$, $\mathsf{R}_i[g]$ will record the set of polynomial representation of g.
- *Ob*: Initially empty list. This will record the set of observable group operation queries.

Furthermore:

- A polynomial f is in $\mathsf{Legal}^{\text{OP}}_{sid}$ if it only contains variables in X_{sid} and a zero constant term.
- A polynomial f is in $\mathsf{Legal}^{\text{PAIR}}_{sid}$ if in addition to constraints in $\mathsf{Legal}^{\text{OP}}_{sid}$, it has no degree 1 monomials.

$\underline{\text{INIT}_{sid}()}$

1: **for** $i \in \{1, 2, t\}$: **do**
2: $\quad \tau_i \xleftarrow{\$} \text{INJ}(\mathbb{Z}_p, S_i)$
3: $\quad \mathsf{R}_i[\tau_i(1)] \leftarrow \{1\}$
4: **end for**

$\underline{\text{CANONICALGEN}_{sid}(i)}$

1: **return** $\tau_i(1)$

$\underline{\text{OBSERVE}_{sid}}$

1: **return** *Ob*

$\underline{\text{OP}_{sid}(i, (g_1, a_1), (g_2, a_2))}$

1: **assert**$(g_1, g_2, a_1, a_2) \in S_i^2 \times \mathbb{Z}_p^2$
2: **for** $j \in \{1, 2\}$ **do**
3: $\quad \text{TOUCH}_{sid}(i, g_j)$
4: **end for**
5: $h \leftarrow \tau_i(a_1\tau_i^{-1}(g_1) + a_2\tau_i^{-1}(g_2))$
6: $\mathsf{R}_i[h] \leftarrow \mathsf{R}_i[h] \cup \left(a_1\mathsf{R}_i[g_1] + a_2\mathsf{R}_i[g_2]\right)$
7: **if** $\exists \mathsf{f} \in \mathsf{R}_i[h] : \mathsf{f} \notin \mathsf{Legal}_{sid}$ **then**
8: $\quad Ob \leftarrow Ob \cup \{(\text{OP}, i, g_1, g_2, a_1, a_2, h)\}$
9: **end if**
10: **return** h

$\underline{\text{TOUCH}_{sid}(i, g)}$

1: **if** $\mathsf{R}_i[g] = \emptyset$ **then**
2: $\quad$ Initialize fresh variable X
3: $\quad \mathsf{X}_{i,sid} \leftarrow \mathsf{X}_{i,sid} : [\mathsf{X}]$
4: $\quad \mathsf{R}_i[g] \leftarrow \{\mathsf{X}\}$
5: **end if**

$\underline{\text{PAIR}_{sid}(g_1, g_2)}$

1: **assert**$(g_1, g_2) \in S_1 \times S_2$
2: **for** $i \in \{1, 2\}$ **do**
3: $\quad \text{TOUCH}_{sid}(i, g_i)$
4: **end for**
5: $h \leftarrow \tau_t(\tau_1^{-1}(g_1) \cdot \tau_2^{-1}(g_2))$
6: $\mathsf{R}_t[h] \leftarrow \mathsf{R}_t[h] \cup (\mathsf{R}_1[g_1] \cdot \mathsf{R}_2[g_2])$
7: **if** $\exists f \in \mathsf{R}_t[h] : \mathsf{f} \notin \mathsf{Legal}_{sid}$ **then**
8: $\quad Ob \leftarrow Ob : [(\text{PAIR}, g_1, g_2, h)]$
9: **end if**
10: **return** h

D UC Simulator and Adversary's Definition

Algorithm 2: Simulator $\mathcal{S}$

$\underline{\textsc{Init}_{sid}()}$
1: Run the code of $\mathcal{F}_{\mathsf{setup}}$
2: Store $\mathsf{td}, \mathsf{crs}$
3: Initialize $\mathsf{st} := (\mathsf{td}, \mathcal{Q}_S)$
4: Initialise empty set $Ob_{\mathcal{S}}$

$\underline{\textsc{GetCrs}_{sid}()}$
1: Return crs

$\underline{\textsc{CanonicalGen}_{sid}()}$
1: Forward the query to $\mathcal{G}$.

$\underline{\textsc{Observe}_{sid}()}$
1: Forward the query to $\mathcal{G}$.

$\underline{\textsc{Op}_{sid}()}$
1: Forward the query to $\mathcal{G}$.
2: Update $Ob_{\mathcal{S}}$

$\underline{\textsc{Touch}_{sid}()}$
1: Forward the query to $\mathcal{G}$.

$\underline{\textsc{Pair}_{sid}()}$
1: Forward the query to $\mathcal{G}$.
2: Update $Ob_{\mathcal{S}}$

$\underline{\mathcal{G}_{\mathsf{RO}}.\textsc{Query}_{sid}()}$
1: Forward the query to $\mathcal{G}_{\mathsf{RO}}$.

$\underline{\mathcal{G}_{\mathsf{RO}}.\textsc{Observe}_{sid}()}$
1: Forward the query to $\mathcal{G}_{\mathsf{RO}}$.

$\underline{\mathcal{S}.\textsc{Simulate}_{sid}(x)}$
1: $\pi \leftarrow S(\mathsf{st}, \mathsf{crs}, x)$
2: $\mathcal{Q}_S \leftarrow \mathcal{Q}_S : (x, \pi)$

$\underline{\mathcal{S}.\textsc{Extract}_{sid}(x, \pi)}$
1: $b \leftarrow V^{\mathcal{G}_{\mathsf{RO}}, \mathcal{G}}(\mathsf{crs}, x, \pi)$
2: If $b = 0$, return $\perp$
3: $Ob_{sid} \leftarrow \textsc{Observe}_{sid}$
4: $Ob_{\mathcal{S}} := Ob_{sid} \cup Ob_{\mathcal{S}}$
5: **for** $i \in \{1, 2, t\}$ **do**,
6: $\quad \{(i, h_j, B_j)\}_{j \in k_i} \leftarrow E_0(\mathsf{crs}, x, \pi)$
7: $\quad$ **for** $j \in k_i$ **do**
8: $\quad\quad \boldsymbol{c}^{(i)}[j] \leftarrow \textsc{FindRep}(i, h_j, B_j, Ob_{\mathcal{S}})$
9: $\quad$ **end for**
10: **end for**
11: Run $w \leftarrow E_1(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \boldsymbol{c}^{(t)})$
12: **return** w

Notations. Recall that s-G refers to the symbolic version of Shoup's GGM. For common data-structures (for example τ) that would be maintained both by the adversary and s-G, we denote the former simply as τ, while s-G.τ for the latter. We use, X_π denote the set of variables associated with group elements that are output by either the Prove command or $\textsc{Init}_{sid}$ execution. X_{sid} and X_π together contain all variables belonging to session sid. When we write $\mathsf{f}^{(3)}(\tau(\mathsf{X}_{sid'}))$, it is a short hand to mean that each X in f should be replaced by corresponding group element, and sequence of group operation should be performed according to the definition of f.

$\mathcal{A}^*(\mathcal{Z})$: Run the environment $\mathcal{Z}$ answering the queries it requests as follows:

1. $\mathcal{G}_{\mathsf{RO}}$ queries:
   - Query: Parse the query as (sid^*, x). If $sid^* = sid$, then forward the query x to standard RO in the underlying game. Otherwise, simulate a response by lazy sampling. If the query was made in a session $\neq sid^*$, record it in the observable set Ob_{RO}.
   - Observe: Return Ob_{RO}.
2. s-$\mathcal{G}$ queries:
   - $\underline{\textsc{Init}()}$: To initialize s-$\mathcal{G}$ functionality,
     (a) Invoke s-G.Init()
     (b) For $i \in \{1, 2, t\}$, call s-G.CanonicalGen(i) to obtain s-G.$\tau_i(1)$ and initialize $\tau_i(1) :=$ s-G.$\tau_i(1)$.
     (c) For $i \in \{1, 2, n\}$, $g \xleftarrow{\$} S_i$ and query s-G.Op($i, g, g, 1, -1$) to receive h_i. Initialize $\tau_i(0) := h_i$.
     (d) Initialize empty lists for $\mathsf{X}_{i,sid}$, $\mathsf{X}_{i,\pi}, \mathsf{X}_{i,sid'}$ Ob.
   - $\underline{\textsc{CanonicalGen}_{sid}(i)}$: Forward query to s-G.CanonicalGen(i), and return the result.
   - $\underline{\textsc{Observe}_{sid}()}$: Return the set Ob.

– $\underline{\text{OP}_{sid}(i, g_1, g_2, a_1, a_2)}$:

(a) Query to $\mathsf{s\text{-}G}.\text{OP}(i, g_1, g_2, a_1, a_2)$ and receive h in return. If $h = \perp$ then return $\perp$. Else, continue.

(b) For each $j \in \{1, 2\}$, execute the code in $\text{TOUCH}_{sid}(i, g_j)$.

(c) For each $j \in \{1, 2\}$, retrieve the polynomial for g_j: $\mathsf{p}_{g_j}(\mathsf{X}_{i,sid}, \mathsf{X}_{i,\pi}, \mathsf{X}_{i,sid'}) \leftarrow \tau_i^{-1}(g_j)$ such that p_{g_j} is a linear polynomial.

(d) Compute the result polynomial $\mathsf{f}(\mathsf{X}_{i,sid}, \mathsf{X}_{i,sid'}, \mathsf{X}_{i,\pi}) \leftarrow a_1\mathsf{p}_{g_1}(\mathsf{X}_{i,sid}, \mathsf{X}_{i,\pi}, \mathsf{X}_{i,sid'}) + a_2\mathsf{p}_{g_2}(\mathsf{X}_{i,sid}, \mathsf{X}_{i,\pi}, \mathsf{X}_{i,sid'})$.

(e) Update τ_i with respect to f: if $h \notin \mathsf{im}(\tau_i)$, then define $\tau_i(\mathsf{f}) := h$.

(f) Rewrite $\mathsf{f} := \mathsf{f}^{(1)}(\mathsf{X}_{i,sid}, \mathsf{X}_{i,\pi}) + \mathsf{f}^{(2)}(\mathsf{X}_{i,sid'}) + c$, where both $\mathsf{f}^{(1)}$, $\mathsf{f}^{(2)}$ are linear polynomials with zero constant term, and c is a constant.

(g) Update Ob: Locally check if $c = 0$ and $\mathsf{f}^{(2)}$ is a zero polynomial. If not then update $Ob \leftarrow Ob : [(\text{OP}, i, g_1, g_2, a_1, a_2, h)]$.

– $\underline{\text{OP}_{sid'}(i, g_1, g_2, a_1, a_2)}$: Behave identically as OP_{sid} until the last step. To update Ob do the following,

(a) Let $\mathsf{f}^{(1)}(\mathsf{X}_{i,sid}, \mathsf{X}_{i,\pi}) := \langle \boldsymbol{a}, \mathsf{X}_{i,sid}\rangle + \langle \boldsymbol{b}, \mathsf{X}_{i,\pi}\rangle$. Compute $h \leftarrow \langle \boldsymbol{a}, \tau_i(\mathsf{X}_{i,sid})\rangle + \langle \boldsymbol{b}, \tau_i(\mathsf{X}_{i,\pi})\rangle$ by querying $\mathsf{s\text{-}G}.\text{OP}(i, \cdot)$.

(b) If $c = 0$ and $h \neq \tau_i(0)$ then update $Ob \leftarrow Ob : [(\text{OP}, i, g_1, g_2, a_1, a_2, h)]$.

– $\underline{\text{PAIR}_{sid}(g_1, g_2)}$:

(a) Query to $\mathsf{s\text{-}G}.\text{PAIR}(g_1, g_2)$ and receive h in return. If $h = \perp$ then return $\perp$. Else, continue.

(b) For each $i \in \{1, 2\}$, execute the code in $\text{TOUCH}_{sid}(i, g_i)$.

(c) Retrieve the polynomial for g_i: $\mathsf{p}_{g_i}(\mathsf{X}_{sid}, \mathsf{X}_{\pi}, \mathsf{X}_{sid'}) \leftarrow \mathsf{s\text{-}}\mathcal{G}.\tau_i^{-1}(g_i)$ such that p_{g_i} is a linear polynomial with the constant term denoted by c_i.

(d) Compute the result polynomial $\mathsf{f}(\mathsf{X}_{sid}, \mathsf{X}_{sid'}, \mathsf{X}_{\pi}) \leftarrow \mathsf{p}_{g_1}(\mathsf{X}_{sid}, \mathsf{X}_{\pi}, \mathsf{X}_{sid'}) \cdot \mathsf{p}_{g_2}(\mathsf{X}_{sid}, \mathsf{X}_{\pi}, \mathsf{X}_{sid'})$.

(e) Update $\mathsf{s\text{-}}\mathcal{G}.\tau_t$ with respect to f: if $h \notin \mathsf{im}(\mathsf{s\text{-}}\mathcal{G}.\tau_t)$, define $\mathsf{s\text{-}}\mathcal{G}.\tau_t(\mathsf{f}) := h$.

(f) Check if $c_1, c_2 = 0$. If not, update $Ob \leftarrow Ob : [(\text{PAIR}, g_1, g_2, h)]$. Else, continue.

(g) Rewrite $\mathsf{f} := \mathsf{f}^{(1)}(\mathsf{X}_{sid}, \mathsf{X}_{\pi}) + \mathsf{f}^{(2)}(\mathsf{X}_{sid}, \mathsf{X}_{\pi}, \mathsf{X}_{sid'})$, where $\mathsf{f}^{(2)} := \langle \boldsymbol{v}_1, \mathsf{X}_{sid} \circ \mathsf{X}_{sid'}\rangle + \langle \boldsymbol{v}_2, \mathsf{X}_{\pi} \circ \mathsf{X}_{sid'}\rangle + \langle \boldsymbol{v}_3, \mathsf{X}_{sid'} \circ \mathsf{X}_{sid'}\rangle$ for some vector of field elements $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$.

(h) Rewrite the terms $\langle \boldsymbol{v}_1, \mathsf{X}_{sid} \circ \mathsf{X}_{sid'}\rangle + \langle \boldsymbol{v}_2, \mathsf{X}_{\pi} \circ \mathsf{X}_{sid'}\rangle$ as $\mathsf{f}^{(3)}(\mathsf{X}_{sid'}) \cdot (\mathsf{f}^{(4)}(\mathsf{X}_{sid}) + \mathsf{f}^{(5)}(\mathsf{X}_{\pi}))$.

(i) Do the following checks:

- Compute $h \leftarrow \mathsf{f}^{(3)}(\tau(\mathsf{X}_{sid'})) \cdot (\mathsf{f}^{(4)}(\tau(\mathsf{X}_{sid})) + \mathsf{f}^{(5)}(\tau(\mathsf{X}_{\pi})))$ by querying $\mathsf{s\text{-}G}.\text{OP}$ and $\mathsf{s\text{-}G}.\text{PAIR}$ appropriately.
- If $h \neq \tau_t(0)$, then update $Ob \leftarrow Ob : [(\text{PAIR}, g_1, g_2, h)]$.

– $\underline{\text{PAIR}_{sid'}(g_1, g_2)}$: Behave identically as PAIR_{sid} until (and including) rewriting f. Then do the following,

(a) Do the following checks:

- Compute $h \leftarrow \mathsf{f}^{(1)}(\tau(\mathsf{X}_{sid}), \tau(\mathsf{X}_{\pi}))$ by querying $\mathsf{s\text{-}G}.\text{OP}$ and $\mathsf{s\text{-}G}.\text{PAIR}$ appropriately. Check if $h = \tau_t(0)$.
- Check term $\langle \boldsymbol{v}_1, \mathsf{X}_{sid} \circ \mathsf{X}_{sid'}\rangle + \langle \boldsymbol{v}_2, \mathsf{X}_{\pi} \circ \mathsf{X}_{sid'}\rangle$ as in PAIR_{sid} case.

(b) Update Ob: If either check above fails then update $Ob \leftarrow Ob : [(\text{PAIR}, g_1, g_2, h)]$.

– $\underline{\text{TOUCH}_{sid}(i, g)}$:

(a) Forward query to $\mathsf{s\text{-}G}$.

(b) If $g \notin \mathsf{im}(\tau_i)$ then,

i. Initialize a fresh variable X and $\mathsf{X}_{i,sid} \leftarrow \mathsf{X}_{i,sid} : [\mathsf{X}]$

ii. $\tau_i(\mathsf{X}) \leftarrow g$

3. $\underline{\text{Init}_{sid}()}$:
   (a) Forward the query to *Setup* interface of the game and receive crs in return.
   (b) Store crs as crs_{sid}.
   (c) For all group elements $g \in \mathsf{crs}_{sid}$, execute the code in Touch_{sid} adding the fresh variable to X_π and define $\tau(\mathsf{X}) := g$.
4. $\underline{\text{GetCrs}_{sid}()}$: Output crs_{sid}.
5. $\underline{\text{Prove}_{sid}(x, w)}$:
   (a) Forward the query to P interface of the game and receive proof string $\boldsymbol{\pi}$ as the output.
   (b) For all group elements $g \in \boldsymbol{\pi}$ such that $g \notin \mathsf{im}(\tau)$, execute the code in Touch_{sid} adding the fresh variable to X_π and define $\tau(\mathsf{X}) := g$.
6. $\underline{\text{Verify}_{sid}}$: Forward the query to V interface of the game and return the decision bit b.

E Omitted Claims for Theorem 3

Claim 3. Hyb_0 *and* Hyb_1 *are indistinguishable.*

Proof. This is implied by Lemma 3 in [12].

Claim 4. Hyb_1 *and* Hyb_2 *are indistinguishable.*

Proof. In Hyb_1 $\mathcal{B}$ behaves exactly as honest parties would in the real world. In addition to this, it intercepts $\mathcal{F}_{\mathsf{setup}}$ and $\mathsf{s}\text{-}\mathcal{G}$ queries without tampering them in any way. Hence, the view of $\mathcal{Z}$ is identical in the two hybrids.

Claim 5. Hyb_2 *and* Hyb_3 *are indistinguishable.*

Proof. This is indistinguishable because of internal element hiding property (Definition 8).

Claim 6. Hyb_3 *and* Hyb_4 *are indistinguishable.*

Proof. When the constant term in the resultant polynomial is non zero, the tuple is always added to *Ob*. Also, for Pair queries if any monomial with degree 1 has a variable not corresponding to the target group, then this too is added to *Ob*. Hence the differences between the hybrid are merely syntactic and otherwise they are identical.

Claim 7. Hyb_4 *and* Hyb_5 *are indistinguishable.*

Proof. The difference between the two hybrids lies in how observability is decided for Op queries made in the challenge session *sid*. Since the polynomials are linear, the two hybrids are identical: In Hyb_4, receiving a query for $\text{Op}_{sid}(i, g_1, g_2, a_1, a_2)$ the group functionality computes the polynomial representation of the result $(a_1\tau_i(g_1) + a_2\tau(g_2))$, then checks if this is a linear polynomial of the form $\sum_{\mathsf{X} \in \mathsf{X}_{sid}} \mathsf{X} \cdot \mathbb{Z}_p[\mathsf{X}_{sid}]$ (the constant term is zero because of the previous hybrid). This is equivalent to checking if the coefficients corresponding to variables in $\mathsf{X}_{sid'}$ are zero (i.e., checking if $\mathsf{f}^{(2)}$ is zero) in Hyb_5.

Claim 8. Hyb_5 *and* Hyb_6 *are indistinguishable.*

Proof. The difference between the two hybrids lies in how observability is decided for Op queries made in session sid'. As in the previous claim, in the first hybrid the group functionality checks if the resultant polynomial is of the form $\sum_{\mathsf{X} \in \mathsf{X}_{sid'}} \mathsf{X} \cdot \mathbb{Z}_p[\mathsf{X}_{sid'}]$, which is equivalent to checking that the coefficients corresponding to variables in X_{sid} are zero (i.e., checking if $\mathsf{f}^{(1)}$ is zero).

Claim 9. Hyb_6 *and* Hyb_7 *are indistinguishable.*

Proof. The difference between the two hybrids is in how observability is decided for PAIR queries made in the challenge session *sid*. The first hybrid, determines legality by looking at the full polynomial, while in the second hybrid only term $\mathsf{f}^{(2)}$ is checked. It is easy to see that this term contains possible illegal monomials (degree 2 $\mathsf{X}_{sid'}$ terms and cross terms). Hence the checks are identical in both hybrids.

Claim 10. Hyb_7 *and* Hyb_8 *are indistinguishable.*

Proof. The difference between the two hybrids is in how observability is decided for PAIR queries made in the session sid'. Term $\mathsf{f}^{(1)} + \langle \boldsymbol{v}_1, \mathsf{X}_{sid} \cdot \mathsf{X}_{sid'} \rangle$ holds all possible illegal monomials. Hence the checks in both the hybrids are identical.

Claim 11. Hyb_8 *and* Hyb_9 *are indistinguishable.*

Proof. The difference in Hyb_9 is that instead of forwarding calls to $\mathcal{F}_{\mathsf{setup}}$, $\mathcal{B}$ runs the code internally and responds to the queries made to $\mathcal{F}_{\mathsf{setup}}$ itself. $\mathcal{B}$ behaves identically as $\mathcal{F}_{\mathsf{setup}}$ except that now it maintains the trapdoors internally. However, the view for $\mathcal{A}$ and $\mathcal{Z}$ still remains identical.

F AGM to GGM Lifting Theorem

We will now present a lifting result adapted to algorithms with access to random oracles. Most of the definitions, theorem and proofs remain largely unchanged. The big take-away of this section is that appropriate versions of random oracles can be compatible with type-safe models while preserving security guarantees. This allows to capture zkSNARKs in this framework.

Advantage Metrics. Let Ch denote a security game in $\mathsf{SA}^{\mathsf{RO_{ts}}}[U_\sigma]$. Ch is given the security parameter as the input and interacts with an adversarial algorithm $\mathsf{A} = (\mathsf{A}_1, \ldots, \mathsf{A}_k)$, where each A_i passes the entire state to the next. Both Ch and A have access to $\mathsf{RO_{ts}}$. After interaction, Ch outputs a bit b. The probability of $b = 1$ after the interaction between Ch and A is denoted by $\mathsf{Adv}_U^{\mathsf{Ch}}(\mathsf{A})$. The game Ch is said to be generically hard if $\mathsf{Adv}_U^{\mathsf{Ch}}(\mathsf{A}) \leq \mathsf{negl}(\lambda)$ for all PPT $A \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$.

Reduction. Fix U and $\mathsf{RO_{ts}}$. An algebraic reduction is a map $R : \mathsf{AA}^{\mathsf{RO_{ts}}}[U] \to \mathsf{AA}^{\mathsf{RO_{ts}}}[U]$. It is model preserving if $\mathsf{A} \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$ implies $R[A] \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$. It is efficiency preserving if $R[\mathsf{A}]$ is a PPT algorithm whenever A is.

AGM-reduction. Fix U and $\mathsf{RO_{ts}}$. Fix advantage metrics A_U^{met} and $\mathsf{Adv}_U^{met_1}, \ldots, \mathsf{Adv}_U^{met_n}$, for a constant n. We say algebraic reductions $R_1, \ldots, R_n$ are Δ_i-*AGM* reductions for these metrics if for all $\mathsf{A} \in \mathsf{AA}_U^{\mathsf{RO_{ts}}}$, we have,

$$\mathsf{Adv}_U^{met}(\mathsf{A}) \leq \Delta_1 \cdot \mathsf{Adv}_U^{met_1}(R_1[\mathsf{A}]) + \ldots \Delta_n \cdot \mathsf{Adv}_U^{met_n}(R_n[\mathsf{A}])$$

Theorem 8. *Fix universe U and advantage metrics $\mathsf{Adv}_U^{met}, \mathsf{Adv}_U^{met_1}, \ldots, \mathsf{Adv}_U^{met_n}$, where $\mathsf{Adv}_U^{met_1}, \ldots, \mathsf{Adv}_U^{met_n}$ are generically hard. Let $R_1, \ldots, R_n$ be Δ_i-AGM reductions for these metrics that are model preserving and efficiency preserving. Then Adv_U^{met} is generically hard.*

Proof. Let $\mathsf{A} \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$ be given against security game corresponding to Adv_U^{met}. Because all R_i's are Δ_i-*AGM* reductions then,

$$\mathsf{Adv}_U^{met}(\mathsf{A}) \leq \Delta_1 \cdot \mathsf{Adv}_U^{met_1}(R_1[\mathsf{A}]) + \ldots \Delta_n \cdot \mathsf{Adv}_U^{met_n}(R_n[\mathsf{A}])$$

Since each R_i is also model preserving and efficiency preserving, $R_i[\mathsf{A}]$ is efficient and in $\mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$ whenever A is. Finally, since security games corresponding to $\mathsf{Adv}_U^{met_1}, \ldots, \mathsf{Adv}_U^{met_n}$ are generically hard, each $\Delta_i \cdot \mathsf{Adv}_U^{met_i}(R_i[\mathsf{A}])$ must be negligible. Since n is a constant $\mathsf{Adv}_U^{met}(\mathsf{A})$ is upper-bounded by a negligible function.

G Equivalence of Type-Safe and Shoup-GGM Models with Random Oracles

G.1 Security in Shoup's GGM implies security in the type-Safe model

Theorem 9. *For any PPT $\mathcal{A} \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$ whose final output has type $\langle \mathsf{bits} \rangle$, let $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ be its canonical Shoup-GGM translation. With the strong encoding, the output distributions of $\mathcal{A}^{\mathsf{RO_{ts}}}$ and $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ are identical. With the weak encoding, they are statistically close.*

Proof. Observe that it is equivalent to consider an $\langle \mathsf{elem} \rangle$ variable containing discrete log a and a label $\tau(a)$ from Shoup's GGM. An algorithm $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ simply replaces variables with type $\langle \mathsf{elem} \rangle$ with a $\langle \mathsf{bits} \rangle$ corresponding to $\tau(a)$. Furthermore, for strong encoding, the random oracles outputs in both $\mathcal{A}^{\mathsf{RO_{ts}}}, \mathcal{B}^{\mathsf{G},\mathsf{RO}}$ are uniformly random. For weak encoding, since $\mathcal{A}$ is a polynomial time algorithm, all the query to $\mathsf{RO_{ts}}$ preserve the same output distribution as queries to RO except when collisions might happen: i.e., two queries that are distinct for $\mathsf{RO_{ts}}$, may result in the same RO query after translation. We can bound this probability as below.

Parse all typed random oracle queries made by $\mathcal{A}$ into (e_k, s_k), and let Q be the total number of such tuples. The worst-case test is to compare every Shoup label used in an RO tuple with every bit string used in a type safe RO tuple. Define:

$$\mathsf{BAD} := \{\exists k, \ell \in [Q] : \tau(e_k) = s_\ell\}.$$

For each fixed pair (k, ℓ), the value $\tau(e_k)$ is uniform in the Shoup label space of its sort. Hence:

$$\Pr[\mathsf{BAD}] \leq \frac{Q^2}{|S|} = \mathsf{negl}(\lambda).$$

Hence, it follows that the output distributions of $\mathcal{A}^{\mathsf{RO_{ts}}}$ and $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ are the statistically close.

We now prove the easy direction: if the canonically translated game is secure under Shoup-GGM and standard RO, then the original type-safe game is secure.

Theorem 10 (Restatement of Theorem 6). *Let Ch' be the canonical Shoup-GGM translation of a type-safe game Ch (Definition 5). If Ch' is secure against adversaries with oracle access to G and RO, then Ch is secure in the $\mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$ model.*

Proof. Suppose that $\mathcal{A} \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$ breaks Ch. Let $\mathcal{A}'$ be the canonical translation of $\mathcal{A}$ acc. Definition 5. The pair $(Ch', \mathcal{A}')$ is the canonical translation of $(Ch, \mathcal{A})$, and the two executions have the same bit output distribution up to the distance from Theorem 9 (zero for the strong encoding, negligible for the weak encoding). Hence, $\mathcal{A}'$ breaks Ch' with essentially the same probability, contradicting the assumed security of Ch' in Shoup's GGM.

G.2 Security in type-safe model implies security in Shoup's-GGM

Theorem 11 (Restatement of Theorem 5). *Let Ch be a single-stage game for a primitive Π in the type-safe oracle framework, and let (Ch', Π') be its canonical Shoup-GGM translation (Definition 5) using strong encoding. If Π satisfies Ch against all PPT adversaries $\mathcal{A} \in \mathsf{TSA}^{\mathsf{RO_{ts}}}[U]$, then Π' satisfies Ch' against all PPT adversaries with oracle access to G and RO. When using weak encoding, the same implication holds, if (Ch, Π) determine a context ctx (described in Section 6.2).*

Proof. We will proceed by proving the contrapositive. Let $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ be a single-stage PPT adversary that wins the Shoup-GGM game Ch' with probability ϵ. We construct a type-safe adversary $\mathcal{A}^{\mathsf{RO_{ts}}}$ for Ch that wins with the same advantage.

Following discussion in Section 6.2, for weak encoding we assume that (Ch, Π) determine a context ctx after fixing the protocol parameters. This context specifies the expected random-oracle query lengths and, for each such length, a public parsing rule that identifies the Shoup-label and bit-string fields. Queries of other lengths are irrelevant for deciding the security game and are answered locally by lazy sampling. For strong encoding, ctx is the unique decoder induced by the prefix-free encoding.

Adversary's definition. Below we describe explicitly for elements in group i, which should be replicated for each $i \in \{1, 2, t\}$. We use set S to denote the group set according to Shoup-GGM. Since in this paper we consider that the size of the set S is the same as the group, all labels are valid group elements.

1. **Initialization:** $\mathcal{A}$ maintains a table T mapping between $\langle \mathsf{elem} \rangle$ variables and corresponding labels from Shoup's GGM, and T_{RO}. Let $\langle \mathsf{elem} \rangle \mathsf{init}$ denote the variable corresponding to the group generator.
2. **Message from Ch to $\mathcal{A}$:** Consider a message from the challenger Ch containing a list of element variables $\{e_i\}_{i \in [k]}$ and a string $s \in \langle \mathsf{bits} \rangle$. For each e_i, $\mathcal{A}^{\mathsf{RO}_{\mathsf{ts}}}$ searches the table T. If an entry exists, it retrieves the corresponding label $\ell_i \in \langle \mathsf{bits} \rangle$. Otherwise, it samples a random label $\ell_i \xleftarrow{\$} S \setminus T$ and adds the mapping $T[e_i] \leftarrow \ell_i$. Finally, it sends the labels $\{\ell_i\}_{i \in [k]}$ and the string s to $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$.
3. **Message from $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$ to Ch':** Consider a message containing a list of labels $\{\ell_i\}_{i \in [k]}$ and a string $s \in \langle \mathsf{bits} \rangle$. For each label $\ell_i \in \langle \mathsf{bits} \rangle$, $\mathcal{A}^{\mathsf{RO}_{\mathsf{ts}}}$ searches T. If an entry exists, it retrieves the corresponding element variable $e_i \in \langle \mathsf{elem} \rangle$. Else, sample $a \xleftarrow{\$} \mathbb{Z}_p$ such that no $\langle \mathsf{elem} \rangle$ variable in T contains a (this can be checked by calling the type-safe oracle Eq on entries in T), and compute the element e_i (calling type-safe oracle $\textsc{Op}(\langle \mathsf{elem} \rangle \mathsf{init}, \bot, a, \bot)$). Add the mapping $T[e_i] \leftarrow \ell_i$. It then sends the element wires $\{e_i\}_{i \in [k]}$ and the string s to Ch.
4. **Random Oracle (RO) Query from $\mathcal{B}$:** When an RO query is made on a string $s \in \langle \mathsf{bits} \rangle$, $\mathcal{A}^{\mathsf{RO}_{\mathsf{ts}}}$ first checks if $s \in T_{\mathsf{RO}}$. If so, it returns the stored string $r \in \langle \mathsf{bits} \rangle$. If not, it checks whether s has the length specified in ctx:
   - If $\mathsf{len}(s) \in \mathsf{ctx}$: parse s according to the rule specified in ctx). For each parsed label ℓ_i, check T, and retrieve element $e_i \in \langle \mathsf{elem} \rangle$ if it exists. If no entry exists, sample a (as in step 3) and compute $\langle \mathsf{elem} \rangle e_i$. Store $T[e_i] \leftarrow \ell_i$.
   - Compute r by querying the underlying $\mathsf{RO}_{\mathsf{ts}}$ oracle with the sequence $(e_i, s_i)_{i \in [n]}$ generated in the previous step, and store $T_{\mathsf{RO}}[s] \leftarrow r$.
   - If $\mathsf{len}(s) \notin \mathsf{ctx}$: sample a random string $r \xleftarrow{\$} \{0,1\}^n$ and store it in $T_{\mathsf{RO}}[s] \leftarrow r$.

   Finally, it return r to $\mathcal{B}$.

Sequence of hybrids. To prove that $\mathcal{A}$ wins whenever $\mathcal{B}$ wins, except with a negligible probability, we consider the following hybrids.

Hyb_0**:** This is the real Shoup-GGM execution of Ch' with $\mathcal{B}^{\mathsf{G},\mathsf{RO}}$. By assumption, $\Pr[\mathsf{Hyb}_0 = 1] = \epsilon$.

Hyb_1**:** Injection τ in Shoup's-GGM is replaced by a lazy sampling version: fresh labels are mapped to $a \in \mathbb{Z}_p$ (without replacement). Instead of sampling the injection τ in the beginning, it is now implemented lazily; this is indistinguishable from behaviour in Hyb_0.

Hyb_2**:** Replace calls to Shoup's-GGM by calls to the type-safe oracles: Maintain table T as described in the adversary's algorithm. Forward $\textsc{Op}, \textsc{Pair}$ queries to type-safe oracles using T to translate labels to element variables. Replace $\textsc{CanonicalGen}$ query with a query to $\textsc{Gen}(1)$ oracle. Table T is updated with the results from the type-safe oracles. This is identically distributed to Hyb_1, since the change in behaviour is merely syntactic.

Hyb_3**:** Replace RO by a lazy sampled one (maintained by the table T_{RO}). Clearly, Hyb_3 and Hyb_2 are identically distributed.

Hyb_4**:** In addition to an RO, add a $\mathsf{RO}_{\mathsf{ts}}$ and process queries as: For RO queries with lengths in ctx, parse s according to the rule specified in ctx. Replace each label with corresponding element variable (maintained in table T), and forward the query to $\mathsf{RO}_{\mathsf{ts}}$. Store the response in T_{RO}. In both cases the answer is uniform and stored consistently, so Hyb_4 and Hyb_3 are identically distributed.

Hyb_5**:** Move the whole simulation inside $\mathcal{A}^{\mathsf{RO}_{\mathsf{ts}}}$ while $\mathcal{A}$ interacts with the type-safe challenger Ch. Messages from Ch to $\mathcal{A}$ are translated to labels before being given to $\mathcal{B}$, and messages from $\mathcal{B}$ are translated to element variables before being forwarded to Ch. The view of $\mathcal{B}$ is the same as in Hyb_4.

In Hyb_5, the machine interacting with Ch is a type-safe PPT adversary with access only to $\mathsf{RO}_{\mathsf{ts}}$. Hence $\mathcal{A}$ wins Ch with probability $\Pr[\mathsf{Hyb}_5 = 1] \geq \epsilon$. This contradicts the assumed type-safe security of Ch and proves the theorem.

H Plonk and Marlin Typed Simulation Extractability

The work of [29]. Obtaining zkSNARKs by compiling a polynomial interactive oracle proof (PIOP) via an extractable polynomial commitment scheme (PCS) is a common design paradigm [15, 23]. A very popular choice of PCS is the KZG commitment scheme [41] as it leads to constant size proofs. To optimize the compilation process towards getting smaller proof sizes, a common approach is to use the homomorphic properties of PCS for the so-called *linearization trick*. This is in fact the version that several real-world implementations consider.

The work of [29] proved that KZG polynomial commitment scheme (with the linearization trick) is simulation-extractable in the algebraic group model and non-programmable random oracle model. Starting from this, they show that using KZG for the optimized PIOP compilation results in a simulation-extractable zkSNARK in the same model. As a corollary, they obtain that Plonk and Marlin are SIMEXT in the AGM + non-programmable ROM.

Concluding UC security. For us to conclude UC security for Plonk and Marlin given [29], all we need to do is type the result. For information theoretic reduction, notice that the algorithm consists of (1) parsing a list of bitstrings and deriving polynomials, (2) forward these polynomials, therefore it is trivially type-safe. For computational reductions, we must do the following:

- Identify the underlying assumptions in the theorem statement. For computational assumptions, show that they are generically hard.
- Identify the sequence of hybrids for which indistinguishability reduces to one of the computational assumption identified before.
- For these hybrids, type the reduction, and check if it is model-preserving (it will usually be efficiency preserving).

For the specific case of [29], we will have to consider three main theorems – Theorem 4 (states NIZK and SIMEXT for the compiled zkSNARK), and Theorem 2 (states NIZK and SIMEXT for KZG with linearization trick, i.e., protocol Π_{lin}), and Theorem 1 (states SIMEXT for KZG with multiple evaluations, i.e., protocol $\Pi_{m\text{-}eval}$).

For Theorem 4, the computational assumptions are (1) Protocol Π_{lin} is NIZK and SIMEXT (which actually refers to Theorem 2) and (2) uniqueness of KZG proofs (hard under SDH). We can list the sequence of hybrids where these assumptions are used simply by observation. The hybrids using assumption (1) almost always build the reduction following the same outline: run the main adversary internally, forward queries to oracles of the underlying assumption if needed (eg. simulation queries), run the KZG extractor. This means that as long as the KZG extractor is type-safe (which it is as it performs no group operations), the reduction is model-preserving. Also notice that at some point during the reduction, the adversary has to program the random oracle while the simulator at the end doesn't. In this case programmability is a proof artefact and can be seen as something happening internally in the adversary.

For Theorem 2, the main computational assumption is OMSDH, Affine-MDH, and that protocol $\Pi_{m\text{-}eval}$ is SIMEXT (which relates to Theorem 1). For the adaptive version of Theorem 1, the computation assumptions are: OMSDH, and affine-MDH. Finally, we remark that assumptions SDH and OMSDH are already shown to be generically hard in [29], while affine-MDH reduces to discrete-logarithm. It can be observed that the reductions to these problems are model-preserving since any group element computed in the reduction is done via group operations (and not by bit manipulation). To be explicit, we concretely type one of the main reductions, i.e., to OMSDH (Theorem 1, Lemma 3 in [29]) in Figure 6.

Reduction to OMSDH. The reduction maintains a local table T to record all queries to $\mathcal{O}_f$ (Figure 3) made by the underlying adversary $\mathcal{A}$ for performing group operations, and uses this to obtain the coefficients

for any group element present in the output of $\mathcal{A}$. All group operations are performed by querying $\mathcal{O}_f$ and is not explicitly mentioned in the reduction below. On receiving the crs from the OMSDH challenger, it simply forwards it to $\mathcal{A}$, while at the same time recording these group elements in T. On receiving simulation query from $\mathcal{A}$ on (c, x, y), it computes $\pi := (\mathsf{c} - [y]_1)(s-x)^{-1}$, where $(s-x)^{-1}$ is computed by querying the $\mathcal{O}_S$ oracle provided by challenger in the OMSDH game. On receiving a forgery (c^*, x^*, y^*, π^*), given that all prior group operations were recorded in T, the reduction derives the group representation for c^* (represented as the polynomial F^*), then computes Q, r such that $F(X) := Q(X) + r(X - x*)$. Finally, it computes $\mathsf{y}^* := r^{-1}(\pi^* - [q(s)]_1)$ by querying $\mathcal{O}_f$ and submits this to the OMSDH challenger.

Figure 6: OMSDH reduction: $R^{\mathcal{O}}[\mathcal{A}](1^\lambda, \mathsf{crs})$

State:

T: A table to store algebraic representation of each $\langle\mathsf{elem}\rangle$ variable.

Setup:

Parse $\mathsf{crs} = \langle\mathsf{elem}\rangle(g_0, \ldots, g_d)$.
for $i \in [0, d]$ **do**
    Store $T[g_i] \leftarrow 1$
end for
Forward crs to $\mathcal{A}$.

Simulation: $\langle\mathsf{elem}\rangle\ \mathsf{c}, \langle\mathsf{bits}\rangle\ x, \langle\mathsf{bits}\rangle\ y$

$\langle\mathsf{elem}\rangle[(s-x)^{-1}]_1 \leftarrow \mathsf{Ch}.\mathcal{O}_S(1)$
$\langle\mathsf{elem}\rangle\pi := \langle\mathsf{elem}\rangle(\mathsf{c} - [y]_1) \cdot [(s-x)^{-1}]_1$
return π

Group query: $\langle\mathsf{elem}\rangle\ h_1, \langle\mathsf{elem}\rangle\ h_2, \langle\mathsf{bits}\rangle\ a_1, \langle\mathsf{bits}\rangle\ a_2$

$\langle\mathsf{elem}\rangle\ h \leftarrow \mathcal{O}_f(\langle\mathsf{elem}\rangle\ h_1, \langle\mathsf{elem}\rangle\ h_2, \langle\mathsf{bits}\rangle\ a_1, \langle\mathsf{bits}\rangle\ a_2)$.
$T[h] \leftarrow a_1 \cdot T[h_1] + a_2 \cdot T[h_2]$
return h

Forgery Extraction:

$(\langle\mathsf{elem}\rangle\ c^*, \langle\mathsf{bits}\rangle\ x^*, \langle\mathsf{bits}\rangle\ y^*, \langle\mathsf{elem}\rangle\ \pi^*) \xleftarrow{\$} \mathcal{A}$
Define $F^*(X) := T[c^*]$
Define $(Q(X), r) := F(X)/(X - x^*)$
$\langle\mathsf{elem}\rangle\pi_1 := \langle\mathsf{elem}\rangle([q(s)]_1)$
$\langle\mathsf{elem}\rangle\pi_2 := \langle\mathsf{elem}\rangle(\pi^* - \pi_1)$
$\langle\mathsf{elem}\rangle\mathsf{y}^* := \mathcal{O}_f(\langle\mathsf{elem}\rangle\pi_2, \langle\mathsf{elem}\rangle g, \langle\mathsf{bits}\rangle r^{-1}, 0)$
return $(\langle\mathsf{bits}\rangle x^*, \langle\mathsf{elem}\rangle\mathsf{y}^*)$