The Relationship Between Idealized Models Under Computationally Bounded Adversaries

Mark Zhandry[1] and Cong Zhang[2]

[1] Princeton University & NTT
[2] University of Maryland

Abstract. The random oracle, generic group, and generic bilinear map models (ROM, GGM, GBM, respectively) are fundamental heuristics used to justify new computational assumptions and prove the security of efficient cryptosystems. While known to be invalid in some contrived settings, the heuristics generally seem reasonable for real-world applications.

In this work, we ask: *which heuristics are closer to reality?* Or conversely, which heuristics are a larger leap? We answer this question through the framework of computational indifferentiability, showing that the ROM is a strictly "milder" heuristic than the GGM, which in turn is strictly milder than the GBM. While this may seem like the expected outcome, we explain why it does not follow from prior works, and is not the a priori obvious conclusion. In order to prove our results, we develop new ideas for proving computational indifferentiable separations.

1 Introduction

The Random Oracle Model. The random oracle model (ROM) of Bellare and Rogaway [BR93] is ubiquitous in modern cryptography. The goal of this model is to justify the security of cryptosystems in cases where it is unknown how to prove the security relative to standard, widely accepted computational assumptions. In the ROM, one treats a cryptographic hash function as a uniformly random function that can only be accessed by making evaluation queries. The hope is that the ROM accurately reflects real-world attacks on the cryptosystem.

The advantage of the ROM is that it allows for designing simpler protocols than would otherwise be necessary. This helps in aiding the design of new schemes, and many feasibility results are initially proved in the ROM (e.g., identity-based encryption [BF01]). Moreover, due to their simplicity, ROM constructions are often more efficient than their standard-model counterparts. Similarly, ROM schemes can sometimes require milder cryptographic building blocks.

Now, random oracles are exponential-sized objects and therefore cannot exist in real life. To make matters worse, there are somewhat contrived examples [CGH98] of schemes that are proven secure in the ROM, but cannot be instantiated by *any* concrete procedures. Due to such theoretical limitations, a ROM proof is considered a compelling *heuristic*, but certainly less convincing

than a standard-model security proof. On the other hand, the known counter-examples tend to be contrived and, for the most part, do not reflect real-world constructions. Moreover, the best practical attacks on many hash functions and schemes built from them simply use the hash function as a black box. As a result, the trade-off between simplicity/efficiency and justification for security is often viewed as a reasonable compromise.

The Generic Group and Bilinear Map Models. Other "idealized models" for cryptographic objects are also common in the literature, two of the most common being the generic group and generic bilinear map models.

Similar to the random oracle model, the generic group model (GGM) of Shoup [Sho97] treats a cryptographic (cyclic) *group* as having uniformly random labels for the group elements. Slightly more formally, we imagine fixing a generator g of the group, and letting $L(x) = g^x$, which we will call the labeling function. The generic group is then an idealization of a group, where the labeling function L is modelled as a random injection $L : \mathbb{Z}_p \to \{0,1\}^n$, together with an oracle A computing the group operation: $A(x, y, b) \mapsto L(\ L^{-1}(x) + (-1)^b L^{-1}(y)\ )$. Analogous to the random oracle model, the generic group model captures cryptographic groups where the best practical attacks simply perform the prescribed group operations. The generic bilinear map model (GBM), as defined by Boneh and Boyen [BB04], extends the model to cryptographic pairings.

The generic group and bilinear models have been particularly useful for justifying the hardness of the many computational assumptions made on cryptographic groups and bilinear maps. For example, the discrete logarithm, decisional (bilinear) Diffie-Hellman, Diffie-Hellman inversion, Diffie-Hellman exponent, and decision linear assumptions can all be proven hard in the respective generic model. The generic bilinear map model has also been used to directly prove the security of cryptosystems, such as recent constructions of broadcast encryption [AY20] and traitor tracing [Zha20] from pairings.

This Work: Random Oracles vs. Generic Groups/Bilinear Maps. Cryptographers frequently compare various cryptographic primitives or computational assumptions—which assumptions/primitives are more plausible, and which ones are less likely to exist? Feasibility results and black box separations can be seen as giving a partial ordering to the plausibility of such objects, with "stronger" objects being less likely to exist than "weaker" ones.

Along these lines, the central goal of this work is to compare the random oracle, generic group, and generic bilinear map heuristics. In particular, we ask: which of these heuristics are closer or further from the "real" world?

In terms of "raw strength," generic groups appear strictly stronger than random oracles. After all, generic groups imply public key agreement, whereas Impagliazzo and Rudich [IR89] show that random oracles cannot be used to build public key agreement. As such, random oracles alone cannot yield generic groups. In the reverse direction, all of the typical objects implied by random oracles (one-way functions, collision resistant hashing, etc.) can readily be constructed by generic groups.

However, the relative "strength" of an idealized model in terms of implied cryptosystems is actually largely uncorrelated with the relative soundness of the heuristic. As an illustrative example, consider the case of [HSW14], who replaces a random oracle with indistinguishability obfuscation (one of the strongest cryptographic assumptions ever made) in the Full Domain Hash signature scheme [BR93]. By avoiding the random oracle, their proof is sound and free of any heuristics, even though they require substantially stronger tools to achieve security.

More abstractly, random oracle model theorems typically also make computational assumptions, in addition to relying on the random oracle heuristic. But once we make computational assumptions, we can no longer compare idealized models in terms of cryptosystems implied by the model: under the assumption that the given cryptosystem exists, both idealized models "imply" the cryptosystem, by simply ignoring the idealized model. For example, in a world where public key agreement exists (which is widely believed to be the world we live in!), random oracles actually *can* be used to build public key agreement, just by ignoring the random oracle and using the given key agreement scheme. In contrast, the separation of Impagliazzo and Rudich [IR89] explicitly requires working in a world where essentially all (cryptographically useful) computational assumptions are false, and the only source of hardness is the random oracle itself.

In a world where computational assumptions hold, perhaps a random oracle actually *can* be used to build a generic group. The recent work of Zhandry and Zhang [ZZ20] gives some hope: they combine random oracles with *standard-model* key agreement schemes satisfying certain properties, in order to "upgrade" the key agreement scheme into an "ideal" public key agreement scheme. It may therefore seem plausible that an analogous construction can upgrade a sufficiently good standard-model cryptographic group into a generic group, using only the random oracle model as the base heuristic.

We therefore view an idealized model as roughly consisting of two parts: a "standard-model" part, which tells us which types of standard-model cryptosystems are implied by the model, and a "heuristic" part, which corresponds to the features of the model that are not realizable in the standard model, and thus captures how far the model is from the real world. The "standard-model" part is where traditional black box separations come into play; for example, Impagliazzo-Rudich already tells us that generic groups are stronger than random oracles in this sense. However, the comparison between the heuristic parts is a priori entirely unclear; this is the problem we seek to answer.

1.1 Formalizing Our Setting

Before describing our results, we first must formalize what it means to compare the heuristic parts of two idealized models. In particular, we must describe what it would mean for a random oracle to imply or not imply a generic group, for our purposes. To do so, we use the indifferentiability framework of Maurer, Renner, and Holenstein [MRH04].

Consider a construction G of a cryptographic group from a random oracle H, which we will denote as G^H. G^H contains a labeling function L^H, and an

addition function A^H. We would like to define what it means for G^H to be a generic group. A first attempt would require that oracle access to L^H, A^H (but *not* H) be *indistinguishable* from the oracles of a generic group interface. Such an indistinguishable group is actually very easy to construct. Simply set L to be, say, a Feistel network with the random oracle as the round function and A the function that applies the appropriate operations to achieve the group operation. By Luby and Rackoff [LR88], a 4-round Feistel network with a random hidden round function is a strong PRP, which is enough to justify security.

The problem with the above is that a real-world attacker would be able to make queries to H as well, since H represents a concrete hash function known to everyone. But once we allow the attacker to query H, clearly, L^H, A^H, H is trivially distinguishable from a generic group: the attacker can run L^H for himself by making queries to H, and then compare that to what the L oracle outputs. In the L^H, A^H, H case, the outputs would match, while in the generic group case, they clearly would not.

The solution, as demonstrated by Maurer et al., is to use the notion of *indifferentiability*. Here, we define the "real world" as the setting where the adversary has oracle access to L^H, A^H and H. In the "ideal world", the adversary has access to a true generic group, and a "simulated" random oracle H. In the ideal world, a simulator S answers random oracle queries, potentially by keeping state and also by making queries to the generic group. In doing so, the simulator hopes to simulate a random oracle that is "consistent," so that the attacker cannot distinguish the two worlds. If such a simulator exists, we say that (L, A) is indifferentiable from a generic group.

Where do computational assumptions fit in? Suppose a construction of a generic group from a random oracle relied on a computation assumption. This would imply that indifferentiability only holds against computationally bounded distinguishers, since any computational assumption would be meaningless for an unbounded distinguisher. Therefore, we will say that $\mathcal{A}$ *computationally implies* $\mathcal{B}$ if there is a construction that is indifferentiable with respect to all computationally bounded distinguishers[3]. If indifferentiability holds also against computationally *unbounded* distinguishers, we say that $\mathcal{A}$ *statistically implies* $\mathcal{B}$.

We will therefore say that the heuristic part of $\mathcal{A}$ is *strictly milder* than the heuristic part of $\mathcal{B}$ if (1) $\mathcal{B}$ statistically implies $\mathcal{A}$, while (2) $\mathcal{A}$ does not even *computationally* imply $\mathcal{B}$. In this case, we write $\mathcal{A} \triangleleft \mathcal{B}$ [4].

As alluded to above, our notion of comparison between idealized models is largely orthogonal to what kinds of cryptosystems are implied by the models, which is the usual purview of black box separations. Indeed, as discussed above, in the setting of black box separations, the cryptographic groups are stronger than hash functions, in the sense that groups give public key agreement but hash functions cannot under black box constructions. On the other hand, Zhandry and Zhang [ZZ20] show that, under certain computational assumptions, random

[3] [MRH04] uses the terminology of "computational reducibility" for this concept.

[4] Our choice to have $\mathcal{B}$ *statistically* imply $\mathcal{A}$ means that both the standard-model part and heuristic part of $\mathcal{B}$ are stronger than $\mathcal{A}$.

oracles computationally imply both ideal public key agreement and ideal *unique* signatures. Meanwhile, cryptographic groups on their own are not known to imply unique signatures. As such, the relation, say, between random oracles and generic groups in terms of computational indifferentiability is not a priori obvious, given known separations.

1.2 Our Results

With our notation above, we can now write our main result as:

$$\text{ROM} \triangleleft \text{GGM} \triangleleft \text{GBM}$$

In other words, the random oracle model is a strictly milder heuristic than the generic group model, which is a strictly milder heuristic than the generic bilinear map model. We note that this result holds with respect to Shoup's original definition of the GGM [Sho97][5].

While this ordering mirrors intuition and what is suggested by black box separations, we will see in the following discussion that the results are incomparable. Even the "easy" directions—showing that generic bilinear maps imply generic groups, which in turn imply random oracles—are not completely trivial in this setting. For the "hard" directions, we note that there are very limited techniques for proving separations in terms of computational indifferentiability: [MRH04] gives one general approach in terms of entropy, but it cannot be applied in our setting as the computational version requires polynomially bounded entropy. In contrast, the idealized models we consider all have exponential entropy. We therefore have to develop new approaches in order to justify the claimed results.

1.3 Technical Overview

Warm-up: generic groups statistically imply random oracles. We first explain how generic groups imply random oracles. This result is not entirely trivial. The natural choice would be to set the labeling function of the generic group as the random oracle. However, this function has homomorphisms that the random oracle does not. In particular, given $L(x)$ and $L(y)$, it is possible to compute $L(x+y)$. This is not possible for a random oracle.

Our solution is simple: we just truncate the labeling function, setting $H(x)$ to be the first, say, half the bits of $L(x)$ [6]. The intuition is that, given only half the bits of $L(x)$, it is impossible to meaningfully query the addition oracle. As a result, since $L(x)$ on its own is a random oracle we obtain a random oracle[7].

[5] Another variant of the GGM is defined by Maurer [Mau05] which avoids an explicit mapping. Even though many results about the GGM hold in both models, our results hold with respect to the original definition of Shoup.

[6] Deleting any super-logarithmic number of bits will do.

[7] Technically $L(x)$ is a random injection. But a truncated random injection is indistinguishable from a random oracle.

To turn this into an indifferentiability proof, we have to explain how to simulate L and H, given a true random oracle H. The idea is that, for any query to $L(x)$, the simulator will query $H(x)$, set the first half of $L(x)$ to be $H(x)$, and then choose a random string for the second half. The addition oracle A is simulated by looking at the L queries made so far; if one of the input labels is not amongst the L queries, A rejects. The reason this simulator works, roughly, is that the distinguisher cannot make a query to A unless it knows the entire label. Since obtaining the second half of the label required querying L, this means the simulator will know the value for every label queried to A. This allows the simulator to answer queries to A correctly.

Random Oracles do not (computationally) imply Generic Groups. Suppose we have a purported construction $G^H = (L^H, A^H)$ of a cryptographic group from a random oracle H. How do we show that G^H can be differentiated from a generic group, using a computationally efficient distinguisher?

One approach might be to look at the typical security properties assumed of cryptographic groups—such as the hardness of discrete logarithms—and show that such security properties cannot hold on G. Assuming the security property is efficiently falsifiable [Nao03,GW11], this leads to a distinguisher: the distinguisher plays both the role of adversary and challenger for the security property, reporting whether the adversary wins or loses in the security experiment. In the real world, the adversary wins by design, whereas in the ideal world, the adversary cannot win. As a result, the combined adversary-challenger will distinguish the two worlds.

Remember, however, that we need the distinguisher, and hence adversary, to be efficient. Under the plausible assumption that cryptographic group exists satisfying the particular security property, one can set G to be that group, and simply ignore H. An efficient adversary for the security property is then impossible. This means we cannot actually use security properties that are true of standard-model cryptographic groups.

Our idea is to use a variant of the discrete logarithm problem, which importantly is trivially false on standard model groups but easily proved to hold on generic groups. Our variant is what we call the *discrete log identification* (DLI) problem, informally defined as: given $h = L(x)$ [8], construct a (probabilistic, efficient) circuit C such that $C(x)$ accepts with overwhelming probability, but $C(x')$ rejects with overwhelming probability on all $x' \neq x$. The DLI problem is trivially easy on standard-model groups: set $C(x')$ to be 1 if and only if $L(x') = h$, where $L(x') = g^{x'}$ is computed as part of C. On the other hand, the DLI problem is readily shown to be hard on generic groups; importantly, the circuit C is oracle-free, meaning it cannot evaluate $L(x')$ since doing so requires queries to the generic group oracles.

While the DLI problem is easily solvable on standard-model groups, it remains to show that the DLI problem is solvable on any group G^H built from a

[8] Remember that for standard-model groups, $L(x)$ denotes the value g^x for a fixed generator g.

random oracle H, where the attacker (but not C!) can also make queries to H. The difficulty is that G^H may use the random oracle, but the circuit C cannot; as a result $C(x')$ cannot readily compute $L^H(x')$ for itself, and it is therefore not obvious how to make the attack work. The natural approach, following techniques from the literature on black box separations, is to try to "compile out" the oracle H. That is, an attacker can easily construct an *oracle aided* circuit $C^H(x')$ breaking the DLI problem by computing $L^H(x')$ using the oracle H. Then, the hope is to anticipate the oracle queries C will make to H, have the adversary query on those points for itself. This task is usually accomplished by running C^H for itself on several random inputs, recording all queries that were made. Then the adversary hardcodes the query answers into C to get an oracle-free circuit, which C outputs.

Unfortunately, it is not a priori obvious that such a strategy should work, or even that the DLI problem should be easy on groups built from random oracles. To see why, consider an analogous *pre-image identification* (PI) problem for hash functions, where the goal is, given an image h, to compute a circuit that accepts only x such that $H(x) = h$. This PI problem is similarly easy for standard-model hash functions, but hard in the random oracle model. One could hope to break the PI problem on any hash function built from a random oracle by similarly compiling out the random oracle queries. But since it is trivial to build a random oracle from a random oracle (even with indifferentiability), and the PI problem is hard for random oracles, we cannot hope to build an attack on the PI problem in our setting.

Therefore, in order to develop our algorithm for the DLI problem, we must exploit the fact that G^H implements a group in order to compile out H from C^H. In particular, along with the labeling function L^H, there is an addition oracle A^H which maps $(L^H(y), L^H(z), b) \mapsto L^H(y+(-1)^b z)$. This addition oracle potentially makes queries to the random oracle H.

Consider computing $L^H(x)$ from x, which in turn makes queries to the random oracle H. Let Q_x be the set of query/answer pairs made during this process. Consider running the addition oracle $A^H(L^H(y), L^H(z), b = 0)$, where y, z are random conditioned on $y + z = x$. The output of this addition is $L^H(y + z) = L^H(x)$. For each query/answer pair $(q, a) \in Q_x$, there are roughly three possibilities:

1. With non-negligible probability over the choice of y, z, $A^H(L^H(y), L^H(z), b = 0)$ makes a query to H on q.
2. The label $L^H(x)$ does not "depend" on the answer a at all.
3. The label $L^H(x)$ depends on a, but $A^H(L^H(y), L^H(z), b = 0)$ queries q with negligible probability.

Now consider running C^H on input x. We claim that we can hardcode query answers into C to guarantee that it will reconstruct $L^H(x)$ without making any queries at all:

– In Case 1, consider the following procedure: choose a random y, compute $h_y = L^H(y)$, and then compute $h_z = A^H(h, h_y, 1) = L^H(x - y) = L^H(z)$,

where we implicitly define $z = x - y$ (we do not know it). Note that y, z are random conditioned on $y + z = x$. Then, run $A^H(h_y, h_z)$, collecting all oracle queries (q', a') made during this process into a list D. For Case 1, we know that with non-negligible probability (q, a) will be amongst the (q', a') queried. By repeating several times, we obtain that D contains (q, a) with high probability. We then include D in the description of C, using D to answer any queries on the q'.

- In Case 2, since $L^H(x)$ does not depend on a, when evaluating $L^H(x)$ we can change the response H makes on q to a random value independent of H, without affecting the ultimate labeling. Therefore, for any query not in D, we will just have C respond with a uniformly random string.
- In Case 3, it must be that $A^H(h_y, h_z, 0)$ must have, in some sense, been able to figure out a from the inputs h_y, h_z. But this in particular means that $L^H(y)$ or $L^H(z)$ must query H on q, in order for the labels to have information about a. We notice that y and z are both uniform, and in particular distributed identically, and so the probability that $L^H(y)$ resulted in query q is at least $1/2$. But then we can collect (q, a) by additionally recording in D all queries made when computing $h_y = L^H(y)$ above. By repeating the process several times, we obtain (q, a) with high probability.

Now consider evaluating $C(x')$ for $x' \neq x$. In this case, if we got lucky and correctly anticipated all queries C would make to H, we correctly compute the right label for x', which will be unequal to h; hence $C(x')$ rejects as desired. On the other hand, if we did not anticipate all the queries, we just replace the query answers with random values; while this means we compute an incorrect label for x', intuitively the random response to the oracle query should only serve to inject further randomness into the label, and it should still be the case that our incorrect label is different from g^x. The result is a procedure for compiling out the H queries made by C, resulting in an oracle-free C which accepts exactly the discrete log x.

The above sketch is imprecise and ignores numerous low-level details, which we work through in Section 3.

Generic Bilinear Maps are stronger than Generic Groups. We next turn to extend our techniques to separate generic bilinear maps from generic groups. This setting is far more challenging than the separation above, owing to the fact that generic groups have a lot more structure to exploit in order to build a bilinear map.

Since generic groups imply generic groups, we cannot rely solely on the group structure of a bilinear map in order to achieve a separation from generic groups. We instead must factor in the pairing operation. The difficulty is that the pairing operation is also in some sense a restricted group operation (in the multiplicative group $\mathbb{Z}_p^*$), and it could be that the generic group structure is used to implement the pairing. Essentially what we must do is show that a single cryptographic group cannot simultaneously be used to implement both the group operation and pairing operations. This is a very different problem than our separation

above, which showed that the group structure could not be obtained in the first place.

In terms of the proof strategy, it is possible in the generic group model to learn information about a label on a point x through the addition function, in which case one does not necessarily have to even know x. But if we try to adapt our proof technique from above, this means our procedure for collecting query/response pairs will no longer work: our procedure for generating a database D may never be able to query on the exact queries q made by $L^H(x)$, in which case we would fail to simulate the circuit C.

What we can guarantee, however, is the following. Let L be the labeling function for the generic group, and L' the supposed labeling function for a bilinear map construction. When running $L'(x)$, let Q_x denote the vector of queries made to L[9]. What we can show, essentially, is that, for $x = y + z$, given $h_y = L'(y)$ and $h_z = L'(z)$, we can compute explicit matrices M, M' such that

$$Q_x = M \cdot Q_y + M' \cdot Q_z$$

M, M' are obtained by tracing the calls to the addition oracle to the underlying generic group. Similar statements hold for the pairing operation. Our main insight is to use the fact that, for any bilinear map built from a generic group, both the group addition and the pairing operations must be linear in this way. We can then construct many equations in the various Q_x sets. This does not let us solve for the actual underlying variables, but it lets us effectively compile the oracle queries the circuit C, giving an oracle-free circuit that identifies the discrete logarithm. The details are given in Section 4.

1.4 Discussion

Our work shows that generic groups are a greater departure from the real world that random oracles, and generic bilinear maps are a still greater departure.

Random Oracles from Algebraic Tools? Our positive result shows how to (unconditionally) construct a random oracle from a generic group. To the extent which the generic group model for a given cryptographic group is believed, simply by truncating the labeling function one obtains a random oracle. Thus, it is possible to build a plausible random oracle from algebraic tools. In contrast, most of the time the random oracle heuristic is only applied to non-algebraic hash functions.

Multilinear Maps? A natural question left open by our work is whether our results extend to multilinear maps [BS02]. In other words, can it be shown that generic bilinear maps are a strictly milder heuristic than generic trilinear maps, etc.? This is an important question, since the generic multilinear map model has

[9] Technically, $L'(x)$ may make queries to the generic group addition oracle. However, we can replace such queries with queries to L, since L' can trace the origin of all labels to L queries.

been used to establish the security of various constructions, especially that of obfuscation [BR14,BGK+14]. Our separation between generic groups and bilinear maps seems challenging to extend to higher-order multilinear maps. Indeed, our separation inherently relies on solving linear equations arising from the linear structure of a generic group. When moving to the multilinear setting, one obtains higher-degree equations, and it is unclear how to adapt our proof to such equations since it is generally infeasible to solve higher-degree equations.

We note that a very coarse separation is possible using existing works: namely that ideal constant-degree multilinear maps are strictly milder than ideal multilinear maps of polynomial degree. Indeed, it is known that ideal multilinear maps of polynomial degree imply *virtual black box obfuscation* (e.g. [BR14,BGK+14]), while virtual black box is *impossible* in the generic constant-degree multilinear map model [PS16], even using computational assumptions. While these works are not phrased in terms of indifferentiability, the results taken together nonetheless imply an indifferentiability separation.

2 Preliminaries

Notation. Throughout this paper, $\lambda \in \mathbb{N}$ denotes the security parameter. For a finite set S, we denote a uniformly random sample s from S as $s \leftarrow S$.We say a function $\mu(n)$ is negligible if $\mu \in o(n^{-\omega(1)})$, and is non-negligible otherwise. We let $\mathsf{negl}(n)$ denote an arbitrary negligible function. If we say some $p(n)$ is poly, we mean that there is some polynomial q such that for all sufficiently large n, $p(n) \leq q(n)$. We say a function $\rho(n)$ is noticeable if the inverse $1/\rho(n)$ is poly.

2.1 Ideal Objects

Random Oracle Model (ROM). A hash function is a function $H : \{0,1\}^* \to \{0,1\}^n$. The Random oracle model is an idealized model proposed by Bellare and Rogaway [BR93,FS87], which assumes the existence of a truly random publicly accessible hash function H. This means $H(x)$ is chosen uniformly at random and independently for each x. H can be queried on any input x; each query has unit cost, so that a PPT algorithm can only make a polynomial number of queries.

Generic Group Model (GGM) [Sho97] . For our purposes, a cryptographic group is a set $\mathcal{G}$ of prime size p, endowed with an efficiently computable group operation. Equivalently, a cryptographic group is a (not necesarily efficient) embedding of the additive group $\mathbb{Z}_p$ into some set. The Generic Group Model is an idealized model which assumes the existence of a random embedding from $\mathbb{Z}_p$. Concretely, a generic group is a pair $\mathcal{G} = (\mathcal{G}^{\mathsf{label}}, \mathcal{G}^{\mathsf{add}})$. Here, $\mathcal{G}^{\mathsf{label}}$ is a "labeling" function that is a random injection from $\mathbb{Z}_p$ to S, giving the embedding of $\mathbb{Z}_p$ into S. $\mathcal{G}^{\mathsf{add}}$ is the induced group operation: $\mathcal{G}^{\mathsf{add}}(\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2), b) = \mathcal{G}^{\mathsf{label}}(z_1 + (-1)^b z_2)$. We will write $\mathcal{G}^{\mathsf{add}}(h_1, h_2) = \mathcal{G}^{\mathsf{add}}(h_1, h_2, 0)$. These functions can be queried on any input at unit cost.

Generic Bilinaer Maps (GBM) [BB04] . For our purposes, a bilinear map consists of three sets $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T$ of prime size p, each endowed with a group structure. Additionally, there is an efficiently computable map $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$ such that $e(g_1^{z_1}, g_2^{z_2}) = e(g_1, g_2)^{z_1 z_2}$ for some fixed generators $g_1 \in \mathcal{G}_1, g_2 \in \mathcal{G}_2$.

The Generic Bilinear Map Model is an idealized model which assumes the existence of random embeddings from $\mathbb{Z}_p$ into the groups $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T$. Concretely, a generic bilinear map is a tuple $\mathcal{G} = (\mathcal{G}_1^{\mathsf{label}}, \mathcal{G}_2^{\mathsf{label}}, \mathcal{G}_T^{\mathsf{label}} \mathcal{G}_1^{\mathsf{add}}, \mathcal{G}_2^{\mathsf{add}}, \mathcal{G}_T^{\mathsf{add}}, \mathcal{G}^{\mathsf{multi}})$. Here, $\mathcal{G}_i^{\mathsf{label}}$ and $\mathcal{G}_i^{\mathsf{add}}$ are as in the generic group case. $\mathcal{G}^{\mathsf{multi}}$ is then defined as

$$\mathcal{G}^{\mathsf{multi}}(\ \mathcal{G}_1^{\mathsf{label}}(z_1)\ ,\ \mathcal{G}_2^{\mathsf{label}}(z_2)\ ) = \mathcal{G}_T^{\mathsf{label}}(z_1 \times z_2)\ .$$

As before, all queries incur unit cost.

2.2 Indifferentiability

The following definitions in this section are taken verbatim from [ZZ20].

In [MRH04], Maurer, Renner and Holenstein (MRH) propose the indifferentiability framework, which formalizes a set of necessary and sufficient conditions for one system to securely be replaced with another one in a wide class of environments. This framework has been used to prove the structual soundness of a number of cryptographic primitives, which includes hash functions [CDMP05,DRS09], blockciphers [ABD+13,CHK+16,DSSL16], domain extenders [CDMS10] and authenticated encryption with associated data [BF18]. In the following, we first recall the definition of indifferentiability.

A random system $\Sigma := (\Sigma.\text{hon}, \Sigma.\text{adv})$ is accessible via two interfaces $\Sigma.\text{hon}$ and $\Sigma.\text{adv}$, where $\Sigma.\text{hon}$ provides a honest interface through which the system can be accessed by all parties and $\Sigma.\text{adv}$ models the adversarial access to the inner working part of Σ. Typically, a system implements either some ideal objects $\mathcal{F}$, or a construction $C^{\mathcal{F}'}$, which applies some underlying ideal objects $\mathcal{F}'$.

Definition 1 (Indifferentiability [MRH04]). *Let Σ_1 and Σ_2 be two systems and $\mathcal{S}$ be a simulator. The indifferentiability advantage of a differentiator $\mathcal{D}$ against (Σ_1, Σ_2) with respect to $\mathcal{S}$ is*

$$\text{Adv}^{\mathsf{indif}}_{\Sigma_1, \Sigma_2, \mathcal{S}, \mathcal{D}}(1^\lambda) := \Pr[\text{Real}_{\Sigma_1, \mathcal{D}}] - \Pr[\text{Ideal}_{\Sigma_2, \mathcal{S}, \mathcal{D}}],$$

where games $\text{Real}_{\Sigma_1, \mathcal{D}}$ and $\text{Ideal}_{\Sigma_2, \mathcal{S}, \mathcal{D}}$ are defined in Figure 1. We say Σ_1 is indifferentiable from Σ_2, if there exists an efficient simulator $\mathcal{S}$ such that for any probabilistic polynomial time differentiator $\mathcal{D}$, the advantage above is negligible. Moreover, we say Σ_1 is statistically indifferentiable from Σ_2, if there exists an efficient simulator such that, for any unbounded differentiator $\mathcal{D}$, the advantage above is negligible.

In the rest of the paper, we also use the notations in [BF18] and consider the definition above to two systems with interfaces as:

$$(\Sigma_1.\text{hon}(\text{X}), \Sigma_1.\text{adv}(\text{x})) := (\text{C}^{\mathcal{F}_1}(\text{X}), \mathcal{F}_1(\text{x}));$$
$$(\Sigma_2.\text{hon}(\text{X}), \Sigma_2.\text{adv}(\text{x})) := (\mathcal{F}_2(\text{X}), \mathcal{F}_2(\text{x})),$$

$\underline{\text{Real}_{\Sigma_1,\mathcal{D}}:}$	$\underline{\text{HonestR(X)}}$	$\underline{\text{Ideal}_{\Sigma_2,\mathcal{S},\mathcal{D}}:}$	$\underline{\text{HonestI(X)}}$
$b \leftarrow \mathcal{D}^{\text{HonestR,AdvR}}$	Return $\Sigma_1.\text{hon}(X)$.	$b \leftarrow \mathcal{D}^{\text{HonestI,AdvI}}$	Return $\Sigma_2.\text{hon}(X)$.
Return b.	$\underline{\text{AdvR(X)}}$	Return b.	$\underline{\text{AdvI(X)}}$
	Return $\Sigma_1.\text{adv}(X)$.		Return $\mathcal{S}^{\Sigma_2.\text{adv}(\cdot)}(X)$.

Fig. 1. Indifferentiability of Σ_1 and Σ_2, where $\mathcal{S}$ is the simulator and $\mathcal{D}$ is the adversary.

where $\mathcal{F}_1$ and $\mathcal{F}_2$ are two ideal objects sampled from their distributions and $\text{C}^{\mathcal{F}_1}$ is a construction of $\mathcal{F}_2$ by calling $\mathcal{F}_1$.

Definition 2 (Computational Indifferentiable Separation [MRH04]). *Let Σ_1, Σ_2 be two idealized models, we say Σ_2 is computational indifferentiably separated from Σ_1 if for any efficient algorithm A and any efficient simulator $\mathcal{S}$, there exist an efficient differentiator $\mathcal{D}_{A,\mathcal{S}}$ and a noticeable function $\rho(n)$ such that*

$$\text{Adv}^{\mathsf{indif}}_{\text{A}^{\Sigma_1},\Sigma_2,\mathcal{S},\mathcal{D}_{\mathcal{A},\mathcal{S}}}(1^\lambda) := |\Pr[\text{Real}_{\Sigma_1,\mathcal{D}_{\text{A},\mathcal{S}}}] - \Pr[\text{Ideal}_{\Sigma_2,\mathcal{S},\mathcal{D}_{\mathcal{A},\mathcal{S}}}]| \geq \rho(\text{n}).$$

Observe that, if an idealized model Σ_2 is computational indifferentiably separated from another idealized model Σ_1, it means that, we cannot build a scheme A^{Σ_1} such that A^{Σ_1} is indifferentiable from Σ_2, even under arbitrarily strong computational assumptions[10].

3 ROM $\triangleleft$ GGM

In this section, we give evidence that random oracles are strictly milder than generic groups. We first illustrate a separation result, that is GGM is computational indifferentiably separated from ROM, then we show how to build a statistically indifferentiable ROM from GGM.

3.1 Indifferentiable Separation between GGM and ROM

In this part, we show that generic groups are computational indifferentiably separated from random oracles.

Theorem 1 (Main Theorem). *Generic group model is computational indifferentiably separated from random oracle model.*

Proof. Let $\mathcal{G} = (\mathcal{G}^{\mathsf{label}}, \mathcal{G}^{\mathsf{add}})$ be a generic group model, subject to $\mathcal{G}^{\mathsf{label}} : \mathbb{Z}_p \to \{0,1\}^m$ (p is a sufficiently large prime and $m \geq \log p + \lambda$), and let H be a random oracle model that $H : \{0,1\}^* \to \{0,1\}^n$. To establish the proof, we first identify a hard problem P_{hard} for $\mathcal{G}$ and then show that if we can build an indifferentiable scheme (L^H, A^H) (this indicates that there is an efficient simulator $\mathcal{S}$), we can

[10] Unless, of course, those assumptions are false.

break P_{hard} by using $\mathcal{S}$. According to [Sho97], the discrete logarithm problem is hard; specifically for any unbounded adversary $\mathcal{A}$,

$$\Pr[\mathcal{A}^{\mathcal{G}(\cdot)}(\mathcal{G}^{\mathsf{label}}(x)) \to x : x \xleftarrow{\$} \mathbb{Z}_p] \leq \frac{q^2}{2p},$$

where $\mathcal{A}$ is making q queries to $\mathcal{G}(\cdot)$ (either $\mathcal{G}^{\mathsf{label}}(\cdot)$ or $\mathcal{G}^{\mathsf{add}}(\cdot,\cdot,\cdot)$). For ease of exposition, we give a modified hard problem DL_{hard} as follows. Given two group elements, $\mathcal{G}^{\mathsf{label}}(x), \mathcal{G}^{\mathsf{label}}(y)$, any query efficient adversary cannot identify $x+y$ with good probability, more concretely,

$$\Pr_{x,y \xleftarrow{\$} \mathbb{Z}_p}\left[\mathcal{A}^{\mathcal{G}(\cdot)}(\mathcal{G}^{\mathsf{label}}(x), \mathcal{G}^{\mathsf{label}}(y)) \to S : (x+y \in S) \wedge \left(\frac{|S|}{p} \leq \frac{1}{q^3}\right)\right] \leq \frac{q^2|S|}{2p} \leq \frac{1}{q},$$

This variant discrete logarithms tells us that, for any query efficient adversary, it is infeasible to output a set S that covers $x+y$ with a good probability, such that S is a tiny fraction of $\mathbb{Z}_p$. The hardness of DL_{hard} follows immediately from [Sho97] above, by computing S and then sampling a random element from S; assuming the discrete log is in S, the result will be the discrete log with probability $1/|S|$.

We use the hardness of DL_{hard} to build our contradiction as follows. Assuming there is an indifferentiable scheme (L^H, A^H), which indicates that there exists an efficient simulator $\mathcal{S}$ such that no computational differentiator can tell the difference of the real world and ideal world, we then can build a query efficient adversary that breaks DL_{hard} by accessing to $\mathcal{S}$. Here we first specify some parameters:

- A makes at most q_{ind} queries to H when running $A_1^H(x)$;
- A makes at most q_{ind} queries to H when running $A_2^H(A_1^H(x), A_1^H(y), b)$;
- $\mathcal{S}$ makes at most q_{sim} queries to $\mathcal{G}$ when responding to a single query to H;
- advantage of any efficient differentiator is bounded by $\epsilon \leq \mathsf{negl}(\lambda)$;
- q^* is a polynomial such that $q^*(\lambda) \geq \max(\lambda, 10)$;
- $\bar{q} = 6q_{\mathsf{ind}} * q^*, s = \bar{q}^2 * q_{\mathsf{ind}},$;
- $q_{\mathcal{G}} = (s+1) * q_{\mathsf{sim}} * q_{\mathsf{ind}}, f = 10q_{\mathcal{G}}^3$.

Now we are ready to illustrate our attacker for DL_{hard}. Our intuitive strategy is the following: (1) we build an algorithm in the real world within two specific properties (details shown below); (2) we leverage the framework of indifferentiability to transfer this algorithm into the ideal world via the simulator $\mathcal{S}$, and show that the transferred algorithm also achieves the same property; (3) using the transferred algorithm to break DL_{hard}. Next we establish our proof step by step.

Step 1. In this part, we build an algorithm in the real world that achieves the following properties. Concretely, we build an efficient algorithm B_{real}^H that takes inputs $(L^H(x), L^H(y))$ and outputs a circuit P_{real} which has no access to H. This circuit here plays role of a predicate function that takes $z \in \mathbb{Z}_p$ as input and outputs a bit. And we argue that P_{real} would identify $x+y$ with high probability, which means it satisfies the following two properties:

1. $\Pr[P_{\mathsf{real}}(x+y)=1] \geq 1 - \frac{1}{2q^*}$;
2. $\Pr_{z \neq x+y}[P_{\mathsf{real}}(z)=1] \leq 2\sqrt{\epsilon} \leq \mathsf{negl}$,

where the probability is over the distribution of sampling x, y, z and generating P_{real}. For ease of exposition, we denote $(t_1^x, \ldots, t_{q_{\mathsf{ind}}}^x) \xleftarrow{\mathsf{q}} L^H(x)$ as the queries made during the encoding procedure, and similarly $(t_1^{(L(x),L(y))}, \ldots, t_{q_{\mathsf{ind}}}^{(L(x),L(y))}) \xleftarrow{\mathsf{q}} A^H(L^H(x), L^H(y))$ as the queries made when adding the labels for x and y. Next, we give the description of B_{real}^H and P_{real} in Figure 2.

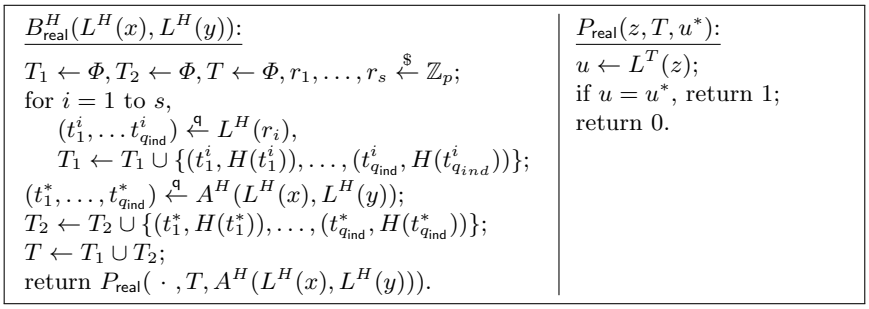

Fig. 2. Algorithms in the real world.

B_{real}^H takes as input two labels $(L^H(x), L^H(y))$. It first runs L^H on s random points r_i, collecting all queries made to H. Then it adds the two input labels together using A^H, collecting all queries made during this process. Then it outputs the program P_{real}, with the list of queries hardcoded, and a target u^* which is the sum of the input labels.

P_{real} takes the input z, and computes the label u on z, checking if the result is u^*. Here, P_{real} is not allowed to make oracle queries; instead it relies on the list of queries computed by B_{real}. Concretely, it computes $u \leftarrow L^T(z)$, which works the exactly as $L^H(z)$ except that on a query t, it does:

- if there is a tuple of the form $(t, a) \in T$, then it responds to the query with a;
- else it samples a random string str, inserts (t, str) into T, and responds with str.

The hope is that by the time B_{real} finishes, T will contain all "relevant" queries, such that P_{real} will satisfy the two desired properties. We now prove this:

First property. Here we prove that, $\Pr[P_{\mathsf{real}}(x+y)] \geq 1 - \frac{1}{2q^*}$. We denote by Q_{x+y} the set of queries made during the encoding procedure $L^H(x+y)$; in other words $Q_{x+y} = \{(t_1^{x+y}, H(t_1^{x+y})), \ldots, (t_1^{x+y}, H(t_1^{x+y}))\}$, where $(t_1^{x+y}, \ldots, t_{q_{\mathsf{ind}}}^{x+y}) \xleftarrow{\mathsf{q}} L^H(x+y)$. We define Q_x, Q_y as the same way as above and denote Q^{add} as the set of queries that are made during $A^H(L^H(x), L^H(y))$.

We immediately observe that, if $Q_{x+y} \subseteq T$, then $L^T(x+y) = L^H(x+y)$, referring to $P_{\mathsf{real}}(x+y) = 1$. Hence it suffices to show that T covers Q_{x+y} with high probability. Unfortunately, this certainly is not true in general as some query in Q_{x+y} may be completely ignored when running $L^H(x+y)$. However, suppose that the label on $x+y$ depends on a for some query (t,a) in Q_{x+y}. If we run A^H on the labels for x, y, then A^H must also output the label on $x+y$, else we can easily differentiate the construction. Thus, A^H must somehow recover a, since the label depends on a. In turn, this means either A^H must have queried on t, or that $L^H(x)$ or $L^H(y)$ must have queried on t.

More abstractly, let U be the label on $x+y$. Then there exist two functions $\mathsf{func}_1, \mathsf{func}_2$ such that $\mathsf{U} = \mathsf{func}_1(x+y, Q_{x+y}) = \mathsf{func}_2(x, y, Q_x \cup Q_y \cup Q^{\mathsf{add}})$, where func_1 corresponds computing the label through $L^H(x+y)$, and func_2 corresponds to computing the label through $A^H(L^H(x), L^H(y))$. As a result, we argue that the only queries we care about are actually $Q_{x+y} \cap (Q_x \cup Q_y \cup Q^{\mathsf{add}})$ (we denote it as Q^*_{x+y} below), because U is in fact independent of the queries in $Q_{x+y} \setminus Q^*_{x+y}$[11].

By the description of B^H_{real}, we note that $Q^{\mathsf{add}} \subseteq T_2 \subseteq T$, hence it is sufficient to show that T also covers $Q_{x+y} \cap (Q_x \cup Q_y)$. However, the fact is that, both (x, Q_x) and (y, Q_y) are unknown to $B^H_{\mathsf{real}}(\cdot,\cdot)$, and to resolve this barrier, our strategy is that, we show that with high probability the queries in $Q_{x+y} \cap (Q_x \cup Q_y)$ are all frequent queries (define below). Thus, we can draw sufficiently large samples, say $r_1, \ldots, r_s$, and record all the queries made in $L^H(r_i)$, and capture all the frequent queries. In other words, T_2 records Q^{add} and T_1 covers $Q_{x+y} \cap (Q_x \cup Q_y)$.

Now we define the frequent query as follows: we say a query $(t, H(t))$ is a frequent query, if

$$\Pr[(t, H(t)) \in Q_z : z \xleftarrow{\$} \mathbb{Z}_p] \geq \frac{1}{\bar{q}}.$$

Then we show that, T_1 captures all of the frequent queries with high probability. In fact, there are at most $v = q_{\mathsf{ind}} * \bar{q}$ frequent queries (we denote those queries as $t_1, \ldots, t_v$), and for t_i, we have

$$\Pr[(t_i, H(t_i)) \notin T_1] \leq \left(1 - \frac{1}{\bar{q}}\right)^s = \left(1 - \frac{1}{\bar{q}}\right)^{\bar{q} * (q_{\mathsf{ind}} * \bar{q})} \leq e^{-q_{\mathsf{ind}} * \bar{q}}.$$

Thus, by union bound, it's apparent that

$$\Pr[(t_i, H(t_i)) \in T_1 : \forall i \in [v]] \geq 1 - q_{\mathsf{ind}} * \bar{q} * e^{-q_{\mathsf{ind}} * \bar{q}} \geq 1 - \lambda e^{-\lambda}.$$

Now, it remains to show that, with high probability, the queries in $Q_{x+y} \cap (Q_x \cup Q_y)$ are all frequent queries. First, note that x and $x+y$ are independent random values, if we ignore y. Therefore, for all queries $(t^x_1, \ldots, t^x_{q_{\mathsf{ind}}}) \xleftarrow{\mathsf{q}} L^H(x)$, if t^x_i is not a frequent query, then by definition, we have

$$\Pr[(t^x_i, H(t^x_i)) \in Q_{x+y}] \leq \frac{1}{\bar{q}}.$$

[11] We can view those queries as insensitive queries and replacing the responses to random strings would not affect the encoding value with high probability

By a union bound, we have that

$$\Pr[(Q_{x+y} \cap Q_x) \text{ contains non-frequent queries}] \leq \frac{q_{\mathsf{ind}}}{\bar{q}} = \frac{1}{6q^*}.$$

We can symmetrically conclude that $(Q_{x+y} \cap Q_y)$ contains frquent queries with probability at most $1/6q^*$. Then we have that

$$\Pr[Q_{x+y} \cap (Q_x \cup Q_y) \text{ are all frequent queries}] \geq 1 - \frac{1}{3q^*},$$

This then implies that

$$\Pr[P_{\mathsf{real}}(x+y) = 1] \geq 1 - \frac{1}{3q^*} - \lambda e^{-\lambda} - \epsilon \geq 1 - \frac{1}{2q^*}.$$

Second property. Here we show that, for other $z \neq x + y$, $P_{\mathsf{real}}(z) = 0$ except with negligible probability. We split $\mathbb{Z}_p$ into two disjoint set: S_1 and S_2, and we say that $z \in S_1$ if and only if $L^H(z) = L^H(x + y)$. Then we show that: (1) $\frac{|S_1|}{p} \leq \sqrt{\epsilon}$; (2) $\Pr[P_{\mathsf{real}}(z) = 1 | z \in S_2] \leq \sqrt{\epsilon}$.

It is straightforward that $\frac{|S_1|}{p}$ must be negl, or else we can build a differentiator runs L^H on random inputs looking for collisions. Now consider a $z \in S_2$. We see that the distribution of $L^H(z)$ and $L^T(z)$ are identical, since L^T just answers all queries not in T with random outputs, matching the distribution of H. Hence if $\Pr[P_{\mathsf{real}}(z) = 1 | z \in S_2] > \sqrt{\epsilon}$, then we have that

$$\Pr[L^T(z_1) = L^T(z_2) | z_1, z_2 \in S_2] > \epsilon \Rightarrow \Pr[L^H(z_1) = L^H(z_2) | z_1, z_2 \in S_2] > \epsilon$$

which contradicts indifferentiability security. Thus, we have

$$\Pr[P_{\mathsf{real}}(z) = 1] \leq 2\sqrt{\epsilon}.$$

Step 2. In this part, we leverage the simulator for the indifferentiability to transfer $(B^H_{\mathsf{real}}, P_{\mathsf{real}})$ to $(B^{\mathcal{S}^{\mathcal{G}}}_{\mathsf{ideal}}, P_{\mathsf{ideal}})$. Concretely, P_{ideal} works identically to P_{real} and $B^{\mathcal{S}^{\mathcal{G}}}_{\mathsf{ideal}}$ works the same as B^H_{real} except for answering queries. Essentially, when B_{real} makes a query to H, say $H(t)$, B_{ideal} calls the simulator $\mathcal{S}$ to answer $H(t)$ by calling at most q_{sim} queries to $\mathcal{G}(\cdot)$. Then we show P_{ideal} also achieves the two properties above with only a tiny loss, specifically:

$$\Pr[P_{\mathsf{ideal}}(x+y) = 1] \geq 1 - \frac{1}{2q^*} - \frac{1}{2f} > 1 - \frac{1}{q^*} > \frac{9}{10};$$
$$\Pr_{z \neq x+y}[P_{\mathsf{ideal}}(z) = 1] \leq \mathrm{negl}(\lambda) + \frac{1}{2f} < \frac{1}{f}.$$

Toward that end, we show that if either of these two properties does not hold in P_{ideal}, then we can build an differentiator D, illustrated in Figure 3 , with advantage at least $\frac{1}{20} > \epsilon$.

Differentiator in real world $\mathcal{D}_{\mathsf{real}}$	Differentiator in ideal world $\mathcal{D}_{\mathsf{ideal}}$

$x, y, z_1, \ldots, z_f \xleftarrow{\$} \mathbb{Z}_p$;
$\mathsf{U}_1 \leftarrow L^H(x), \mathsf{U}_2 \leftarrow L^H(y)$,
$\mathsf{U}_3 \leftarrow L^H(x+y)$;
$P^* \leftarrow B^H_{\mathsf{real}}(\mathsf{U}_1, \mathsf{U}_2)$;
if $P^*(x+y) = 1 \wedge (\forall i \in [f], P^*(z_i) = 0)$,
  return 1;
return 0.
	$x, y, z_1, \ldots, z_f \xleftarrow{\$} \mathbb{Z}_p$;
	$\mathsf{U}_1 \leftarrow \mathcal{G}^{\mathsf{label}}(x), \mathsf{U}_2 \leftarrow \mathcal{G}^{\mathsf{label}}(y)$,
	$\mathsf{U}_3 \leftarrow \mathcal{G}^{\mathsf{label}}(x+y)$;
	$P^* \leftarrow B^{\mathcal{S}^{\mathcal{G}}}_{\mathsf{ideal}}(\mathsf{U}_1, \mathsf{U}_2)$;
	if $P^*(x+y) = 1 \wedge (\forall i \in [f], P^*(z_i) = 0)$,
	  return 1;
	return 0.

Fig. 3. Differentiator for failed properties.

We note that, in the real world, the differentiator outputs 1 with high probability, in fact, we have that

$$\Pr[\mathcal{D}_{\mathsf{real}} = 1] \geq 1 - \frac{1}{2q^*} - 2f\sqrt{\epsilon} \geq \frac{19}{20}.$$

However in the ideal world, we argue (assuming either of the two properties does not hold) that the differentiator $\mathcal{D}_{\mathsf{ideal}}$ outputs 1 with noticeably smaller probability. In the following, we give the analysis case by case:

The first property fails. In this case, we have that $\Pr[P^*(x+y) = 1] \leq \frac{9}{10}$, which immediately refers to that

$$\Pr[\mathcal{D}_{\mathsf{ideal}} = 1] \leq \Pr[P^*(x+y) = 1] \leq \frac{9}{10}.$$

The second property fails. In this case we have that $\forall i, \Pr[P^*(z_i) = 1] \geq \frac{1}{f}$. Moreover, $z_1, \ldots, z_f$ are uniformly sampled, which means $P^*(z_1), \ldots, P^*(z_f)$ are independent, meaning

$$\Pr[\mathcal{D}_{\mathsf{ideal}} = 1] \leq 1 - \left(1 - \frac{1}{f}\right)^f \leq 1 - \frac{1}{e} \approx 0.6 \ .$$

Hence, if one of the two properties fails, then there exists a differentiator that breaks the indifferentiability with advantage at least $\frac{1}{20}$.

Step 3. In this part, we show that how to use $(B^{\mathcal{S}^{\mathcal{G}}}_{\mathsf{ideal}}, P_{\mathsf{ideal}})$ to break DL_{hard}. Recall that, in generic group model, for any query efficient adversary $\mathcal{A}$,

$$\Pr_{x,y \xleftarrow{\$} \mathbb{Z}_p}\left[\mathcal{A}^{\mathcal{G}(\cdot)}(\mathcal{G}^{\mathsf{label}}(x), \mathcal{G}^{\mathsf{label}}(y)) \to S : (x+y \in S) \wedge \left(\frac{|S|}{p} \leq \frac{1}{q_{\mathcal{G}}^3}\right)\right] \leq \frac{1}{q_{\mathcal{G}}} < \frac{1}{10^6},$$

where $\mathcal{A}$ makes at most $q_{\mathcal{G}}$ queries to $\mathcal{G}(\cdot)$. To present the contradiction, we next build a query efficient adversary $\mathcal{A}^*$ in figure 4, which outputs a very small set S such that $x + y \in S$ with high probability.

$\mathcal{A}^*(\mathcal{G}^{\mathsf{label}}(x), \mathcal{G}^{\mathsf{label}}(y))$:
$S \leftarrow \Phi$; $P \leftarrow B^{S^{\mathcal{G}}}$;
for $i = 0$ to $p-1$,
  if $P(i) = 1$, $S \leftarrow S \cup \{i\}$;
return S.

Fig. 4. Adversary to break DL_{hard}.

We immediately observe that $\mathcal{A}^*$ only makes $q_{\mathcal{G}}$ queries to $\mathcal{G}(\cdot)$, hence it suffices to show that $\mathcal{A}^*$ can output a set S, such that $x + y \in S$ and $\frac{|S|}{p} \leq \frac{1}{q_{\mathcal{G}}^3}$, with a good probability, say $\geq \frac{1}{2}$.

In fact, by the analysis in step 2, it immediately follows that $\Pr[x+y \in S] \geq \frac{9}{10}$. In order to bound the size of S, we denote $S^{P_{\mathsf{ideal}}}$ to be the set such that $z \in S^{P_{\mathsf{ideal}}}$ iff $P_{\mathsf{ideal}}(z) = 1$. Due to the analysis in step 2, we have that

$$\Pr_{x,y,P_{\mathsf{ideal}}}[\frac{|S^{P_{\mathsf{ideal}}}|}{p} > \frac{10}{f}] \leq \frac{1}{10} \Rightarrow \Pr_{x,y,P_{\mathsf{ideal}}}[\frac{|S^{P_{\mathsf{ideal}}}|}{p} > \frac{1}{q_{\mathcal{G}}^3}] \leq \frac{1}{10},$$

meaning

$$\Pr[\mathcal{A}^* \text{ wins}] \geq 1 - \frac{1}{10} - \frac{1}{10} = \frac{4}{5}.$$

Combing together, we establish the entire proof. □

3.2 GGM implies ROM

In this part, we show how to build an indifferentiable ROM from GGM. Although it is simple, the result appears to be previously unknown.

Building Blocks. Our construction consists of two building blocks:

- $\mathcal{G} = (\mathcal{G}^{\mathsf{label}}, \mathcal{G}^{\mathsf{add}})$ is a generic group model that maps $\mathbb{Z}_p$ to $\{0,1\}^s$;
- Trunc_n^s is a truncation function that takes s-bit string as inputs and outputs its first n bits.

where $s \geq 2\log p + \lambda$ and $n := \lfloor \log p \rfloor$. Note that, Trunc_n^s truncates any $\mathcal{G}^{\mathsf{label}}(x)$ into an n-bit string. Now, we build an indifferentiable random oracle $\mathcal{H}$ that maps $\{0,1\}^n \to \{0,1\}^n$ as follows:

$$\mathcal{H}(x) = \mathsf{Trunc}_n^s(\mathcal{G}^{\mathsf{label}}(x)).$$

We note that every bit of $\mathcal{H}(x)$ is random and independent, referring to the outputs are well-distributed. Next we show this construction achieves indifferentiability.

Theorem 2 (Indifferentiable ROM). *$\mathcal{H}$ is indifferentiable from a random oracle model. More precisely, there is a simulator $\mathcal{S}$ such that for all $(q_{\mathcal{G}^{\mathsf{label}}}, q_{\mathcal{G}^{\mathsf{add}}})$-query differentiator $\mathcal{D}$ with $q_{\mathcal{G}^{\mathsf{label}}} + q_{\mathcal{G}^{\mathsf{add}}} \leq q$, we have*

$$\mathrm{Adv}^{\mathsf{indif}}_{\mathcal{H},\mathcal{S},\mathcal{D}} \leq \frac{q^2 + q}{2^\lambda}.$$

The simulator makes at most q queries to its oracles.

Proof Sketch. According to the definition of indifferentiability, the adversary has one honest interface $\mathcal{H}$ and two adversarial interfaces $\mathcal{G}^{\mathsf{label}}$ and $\mathcal{G}^{\mathsf{add}}$[12]. Therefore, we need to build an efficient simulator $\mathcal{S}$ that can simulate $(\mathcal{G}^{\mathsf{label}}, \mathcal{G}^{\mathsf{add}})$ properly, which means, for any (even computationally unbounded) differentiator $\mathcal{D}$, the view of $\mathcal{D}$ in the real game is close to the view in the ideal game. In the following, we illustrate the description of our simulator in Figure 5 and then we give the high-level intuition of our proof strategy (due to space limit, we give the full proof in Appendix A).

Algo.$\mathcal{S}.\mathcal{G}^{\mathsf{label}}(x)$:	Algo.$\mathcal{S}.\mathcal{G}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$:

if $\exists(x, y) \in T_{\mathsf{label}}$,
  return y;
$r \leftarrow \{0,1\}^{s-n}$,
$T_{\mathsf{label}} \leftarrow T_{\mathsf{label}} \cup (x, \mathcal{O}(x)\|\|r)$,
  return $\mathcal{O}(x)\|\|r$.
	if $\exists((\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{Z}_2) \vee (\mathsf{Z}_1, \mathsf{Z}_0, \mathsf{Z}_2)) \in T_{\mathsf{add}}$, return Z_2;
	if $\exists((x_0, \mathsf{Z}_0) \wedge (x_1, \mathsf{Z}_1) \wedge (x_0 + x_1, \mathsf{Z}_2)) \in T_{\mathsf{label}}$, return Z_2;
	if $\exists((x_0, \mathsf{Z}_0) \wedge (x_1, \mathsf{Z}_1)) \in T_{\mathsf{label}}$ and $(x_0 + x_1, \mathsf{Z}_2) \notin T_{\mathsf{label}}$,
	  $r \leftarrow \{0,1\}^{s-n}$, $T_{\mathsf{label}} \leftarrow T_{\mathsf{label}} \cup (x_0 + x_1, \mathcal{O}(x_0 + x_1)\|\|r)$,
	  return $\mathcal{O}(x_0 + x_1)\|\|r$;
	return $\perp$.

Fig. 5. Simulator for ROM in terms of two sub-simulators associated with oracle $\mathcal{O}$. These two sub-simulators share two tables $(T_{\mathsf{label}}, T_{\mathsf{add}})$ as joint state (which are initialized empty).

We immediately observe that, our simulator makes at most q queries to $\mathcal{O}$, and it keeps two tables and the size of each table is at most q, referring to $\mathcal{S}$ is efficient. In the following, we present the intuitive idea that why $\mathcal{S}$ works. Note that, $\mathcal{G}$ is a generic group model, hence the responses of a proper simulator should follow the following rules:

1. The responses of $\mathcal{G}^{\mathsf{label}}$ are statistically uniform in $\{0,1\}^s$;
2. There do not exist $x_0 \neq x_1$ such that $\mathcal{G}^{\mathsf{label}}(x_0) = \mathcal{G}^{\mathsf{label}}(x_1)$;
3. $\mathcal{H}(x) = \mathsf{Trunc}_n^s(\mathcal{G}^{\mathsf{label}}(x))$;
4. $\mathcal{G}^{\mathsf{label}}(x_0 + x_1) = \mathcal{G}^{\mathsf{add}}(\mathcal{G}^{\mathsf{label}}(x_0), \mathcal{G}^{\mathsf{label}}(x_1))$,
5. $\forall \mathsf{Z} \notin \{\mathcal{G}^{\mathsf{label}}(x)\}_{x \in \mathbb{Z}_p}$, $\mathcal{G}^{\mathsf{add}}(\mathsf{Z}, \cdot) = \perp$.

Next, we show our simulator achieves those five rules. Observe that rule 1 and 3 trivially holds.

Rule 2. The only way to break this rule is if a collision occurs. As r is uniformly sampled, this bad event is trivially bounded by $\frac{q^2}{2^{s-n}} \leq \frac{q^2}{2^\lambda}$.

Rule 4. Note that, when running $\mathcal{G}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$, if $\mathsf{Z}_0, \mathsf{Z}_1$ are the known valid encoding ($\mathsf{Z}_0, \mathsf{Z}_1$ have already been put into T_{label}), then this equation holds for

[12] For ease of exposition, we here only illustrate the simulation for the addition procedure, and subtraction can be simulated identically.

free. The only bad event that breaks this rule is when $\mathcal{A}$ makes the addition query before the labeling queries (in the real world, the response is $\mathcal{G}^{\mathsf{label}}(x_0+x_1)$, while in the ideal world, the simulator outputs $\perp$ instead).

However, if this bad event occurs, then it means that the adversary needs to predict a valid encoding $\mathcal{G}^{\mathsf{label}}(x)$. Of course, the adversary could have the first n bits of $\mathcal{G}^{\mathsf{label}}(x)$, which it has to outputs the last $s-n$ bits, which in fact is independent of the adversary's view. Thus this bad event is apparently bounded by $\frac{q*p}{2^{s-n}} \leq \frac{q}{2^{\lambda}}$.

Rule 5. By the description of $\mathcal{S}.\mathcal{G}^{\mathsf{add}}$, we know that, if and only if $\mathsf{Z}_0, \mathsf{Z}_1$ are the known valid encoding , the response is not $\perp$. Hence, if Z is an invalid encoding, $\mathcal{S}.\mathcal{G}^{\mathsf{add}}$ always outputs $\perp$. □

4 GGM ◁ GBM

In this section, we give evidence that generic group model is strictly milder than generic bilinear map. We first illustrate a separation result, that is GBM is computational indifferentiably separated from GGM, then we show how to build an indifferentiable GGM from GBM.

4.1 Indifferentiable Separation between GBM and GGM

In this part, we show that generic bilinear maps are computational indifferentiably separated from generic groups.

An intuitive idea is to apply the same method above, since the discrete logarithm problem is also hard in the GBM. Concretely, we build an algorithm $B^{\mathcal{G}}_{\mathsf{real}}$, in the real world, that takes two source group labels $(\mathcal{B}_1^{\mathsf{label}}(x), \mathcal{B}_1^{\mathsf{label}}(y))$ as inputs and outputs a "query-free" circuit P_{real} that can identify xy with high probability. Then we leverage the indifferentiability framework to transfer the algorithms to the ideal world, we build $(B^{\mathcal{S}^{\mathcal{G}}}_{\mathsf{ideal}}, P_{\mathsf{ideal}})$. After that, we break discrete log by P_{ideal}. Unfortunately, in generic group model, this does not work anymore. Below, we illustrate why it fails and propose our new solutions.

First Attempt. We apply the same method as Theorem 1. Let $\mathcal{B} = (\mathcal{B}_1^{\mathsf{label}}, \mathcal{B}_1^{\mathsf{add}}, \mathcal{B}_1^{\mathsf{multi}}, \mathcal{B}_2^{\mathsf{label}}, \mathcal{B}_2^{\mathsf{add}})$ be a generic bilinear map (assuming identical source groups for ease), with $\mathcal{B}_1^{\mathsf{label}}, \mathcal{B}_2^{\mathsf{label}}$ mapping $\mathbb{Z}_p \to \{0,1\}^s$. Let $\mathcal{G} = (\mathcal{G}^{\mathsf{label}}, \mathcal{G}^{\mathsf{add}})$ be a generic group model. To show the impossibility, we use an analous hard problem for the generic bilinear map model. Specifically, we can adapt the techniques of Section 3 to show:

$$\Pr_{x,y \xleftarrow{\$} \mathbb{Z}_p}[\mathcal{A}^{\mathcal{G}(\cdot)}(\mathcal{B}_1^{\mathsf{label}}(x), \mathcal{B}_1^{\mathsf{label}}(y)) \to S \text{ s.t. } (xy \in S) \wedge \left(\frac{|S|}{p} \leq \frac{1}{q^3}\right)] \leq \frac{q^2|S|}{2p} \leq \frac{1}{q}.$$

Following the same strategy, we assume there exists an indifferentiable GBM construction $(L_1^{\mathcal{G}}, A_1^{\mathcal{G}}, M_1^{\mathcal{G}}, L_2^{\mathcal{G}}, A_2^{\mathcal{G}})$, and then build the algorithm $B^{\mathcal{G}}_{\mathsf{real}}$, in figure 6, which takes $L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y)$ as inputs and output a circuit P_{real}. Moreover, we hope P_{real} identifies xy and rejects others with high probability.

$\underline{B^{\mathcal{G}}_{\mathsf{real}}(L^{\mathcal{G}}_1(x), L^{\mathcal{G}}_1(y)):}$

$T_1 \leftarrow \Phi, T_2 \leftarrow \Phi, T \leftarrow \Phi, r_1, \ldots, r_s \xleftarrow{\$} \mathbb{Z}_p$;
for $i = 1$ to s,
  $(t^i_1, \ldots t^i_{q_{\mathsf{ind}}}) \xleftarrow{\mathsf{q}} L^{\mathcal{G}}_1(r_i)$,
  $T_1 \leftarrow T_1 \cup \{(t^i_1, \mathcal{G}(t^i_1)), \ldots, (t^i_{q_{\mathsf{ind}}}, \mathcal{G}(t^i_{q_{ind}}))\}$;
$(t^*_1, \ldots, t^*_{q_{\mathsf{ind}}}) \xleftarrow{\mathsf{q}} M^{\mathcal{G}}_1(L^{\mathcal{G}}_1(x), L^{\mathcal{G}}_1(y))$;
$T_2 \leftarrow T_2 \cup \{(t^*_1, \mathcal{G}(t^*_1)), \ldots, (t^*_{q_{\mathsf{ind}}}, \mathcal{G}(t^*_{q_{\mathsf{ind}}}))\}$;
$T \leftarrow T_1 \cup T_2$;
return $P_{\mathsf{real}}(\cdot, T, M^{\mathcal{G}}_1(L^{\mathcal{G}}_1(x), L^{\mathcal{G}}_1(y)))$.

$\underline{P_{\mathsf{real}}(z, T, u^*):}$

$u \leftarrow L^T_2(z)$;
if $u = u^*$, return 1;
return 0.

Fig. 6. Algorithms in the real world.

Note that, in the algorithm B_{real}, T_1 records the frequent queries and T_2 records the queries in the multiplication procedure. Hence, following the analysis in Theorem 1, T indeed covers all the relevant queries with high probability. However, because the underlying model here is generic group model, rather than random oracle model, this strategy is insufficient.

Why it fails in GGM? It is true that T covers all the relevant queries with high probability. However, the queries recorded in T might not be have sufficient information, and as a result P_{real} fails to respond to the queries properly. Indeed, there are two types of queries in the GGM: labeling queries and adding queries. We write $(z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z))$ as the labeling query and $(\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2), \mathsf{add}, \mathcal{G}^{\mathsf{label}}(z_1 + z_2))$ as the adding query. We observe that, when T records $(z_1 + z_2, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z_1 + z_2))$, it knows $z_1 + z_2$ explicitly. Meanwhile if T *only* records $(\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2), \mathsf{add}, \mathcal{G}^{\mathsf{label}}(z_1 + z_2))$, it has no knowledge of z_1, z_2 or $z_1 + z_2$.

This unfortunately means that, after running B_{real}, some of the tuples recorded in T_2 are the adding queries, while P_{real} needs the corresponding labelling queries. Concretely, when $L^T_2(z)$ makes a relevant labeling query $(z^*_1 + z^*_2, \mathsf{label}, \cdot)$, P_{real} has to respond with $\mathcal{G}^{\mathsf{label}}(z^*_1 + z^*_2)$, while T_2 has only recorded $(\mathcal{G}^{\mathsf{label}}(z^*_1), \mathcal{G}^{\mathsf{label}}(z^*_2), \mathsf{add}, \mathcal{G}^{\mathsf{label}}(z^*_1 + z^*_2))$, which means P_{real} cannot tell whether or not $\mathcal{G}^{\mathsf{label}}(z^*_1 + z^*_2)$ is the proper response. Thus, P_{real} fails to identify xy.

Note that the main barrier showed above is that adding queries themselves might not be sufficient for the algorithm L_2, as the corresponding labelling queries are needed when running L_2. To solve this obstacle, we utilize a new technique. At a high level, we construct two extractor algorithms that "transfer" those adding queries to the corresponding labeling queries with high probability, and then apply the proof framework of Theorem 1 to complete the proof. Essentially, we observe that, although the adding queries (e.g. $(\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2), \mathsf{add}, \mathcal{G}^{\mathsf{label}}(z_1 + z_2))$) do not explicitly give out the underlying value (e.g. $(z_1, z_2, z_1 + z_2)$), they leak *something*, in particular linear constraints on the values, e.g. $(\mathcal{G}^{\mathsf{label}}(z_1) \oplus \mathcal{G}^{\mathsf{label}}(z_2) = \mathcal{G}^{\mathsf{label}}(z_1 + z_2))$ (here we use $\oplus$ as the group addition). Our goal is to collect sufficiently many constraints so as to compute $(z_1, z_2, z_1 + z_2)$ by solving a linear system. Unfortunately, adding queries alone are insufficient to generate enough equations, since generic groups can of course be used to build

generic groups. We then leverage the power of the pairing to form an additional linear system, which lets us solve the equations, transferring all adding queries into labeling queries with high probability.

Theorem 3 (Main Theorem). *Generic bi-linear maps are indifferentiably separated from the generic group model.*

Proof. Applying the same strategy in Theorem 1, we immediately observe that it suffices to build an efficient algorithm $B^{\mathcal{G}}_{\mathsf{real}}$ such that $B^{\mathcal{G}}_{\mathsf{real}}$ takes $L^{\mathcal{G}}_1(x)$ as input and outputs a "query-free" circuit P_{real} that satisfying the following:

1. $\Pr[P_{\mathsf{real}}(x) = 1] \geq \frac{9}{10}$;
2. $\Pr_{z \neq x}[P_{\mathsf{real}}(z) = 1] \leq \mathsf{negl}(\lambda)$.

Hence, in the rest of proof, we focus on $(B^{\mathcal{G}}_{\mathsf{real}}, P_{\mathsf{real}})$. Here are some parameters:

- all algorithms makes at most q_{ind} queries to $\mathcal{G}$;
- advantage of any efficient differentiator is bounded by $\epsilon \leq \mathsf{negl}(\lambda)$;
- q^* is a polynomial such that $q^*(\lambda) \geq \max(\lambda, 30)$;
- $\bar{q} = q_{\mathsf{ind}} * q^*$, $s = q_{\mathsf{ind}} * \bar{q}^2$;

Next we define five sets, $(Q^1_x, D^1_x, F_x, Q^2_x, D^2_x)$, which record *(query, response)* tuples:

- *All-query set on source group:* Q^1_x is a table that records all the labeling queries when running $L^{\mathcal{G}}_1(x)$.
- *Relevant-query set on source group:* D^1_x is a table that only records all the relevant labeling queries for $L^{\mathcal{G}}_1(x)$.
- *Frequently-extractable-query set on source group:* F_x is a table that records tuples with form of $(*, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z))$ as computed by Extractor-1 (Figure. 7). At a high level, F_x records all the frequently extractable queries (defined below) of $L^{\mathcal{G}}_1(x)$, when Extractor-1 runs group operations (e.g. addition or multiplication), on $L^{\mathcal{G}}_1(x)$ and $L^{\mathcal{G}}_1(y)$, where $y \leftarrow \mathbb{Z}_p$ is sampled by Extractor-1.
- *All-query set on target group:* Analogously, Q^2_x is a table that records all the labeling queries when running $L^{\mathcal{G}}_2(x)$.
- *Relevant-query set on target group:* Analogously, D^2_x is a table that only records all the relevant labeling queries for $L^{\mathcal{G}}_2(x)$.

Observe that $D^1_x \subset Q^1_x$ and $D^2_x \subset Q^2_x$. Moreover, as in Theorem 1, we have that if we can build $B^{\mathcal{G}}_{\mathsf{real}}$ that takes $L^{\mathcal{G}}_1(x)$ as input and outputs D^1_x or D^2_x, then P_{real} can be constructed straightforwardly. In fact, one we have fixed D^1_x (or D^2_x), it is apparent that

$$\Pr[L^{\mathcal{G}}_1(x) = L^{D^1_x}_1(x)] \geq 9/10 \ \wedge \ \Pr_{y \neq x}[L^{\mathcal{G}}_1(y) = L^{D^1_x}_1(y)] \leq \mathrm{negl},$$

$$\Pr[L^{\mathcal{G}}_2(x) = L^{D^2_x}_2(x)] \geq 9/10 \ \wedge \ \Pr_{y \neq x}[L^{\mathcal{G}}_2(y) = L^{D^2_x}_2(y)] \leq \mathrm{negl}.$$

Thus in the rest of the proof, we show that how to transfer those adding queries to D_x^1 (or D_x^2) by making use of F_x and the pairing.

As in Theorem 1, we define the frequent queries: a query $(z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z))$ is a frequent query on the source group if

$$\Pr[(z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z)) \in Q_y^1 : y \xleftarrow{\$} \mathbb{Z}_p] \geq \frac{1}{\bar{q}},$$

and a query $(z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z))$ is a frequent query on the target group if

$$\Pr[(z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z)) \in Q_y^2 : y \xleftarrow{\$} \mathbb{Z}_p] \geq \frac{1}{\bar{q}}.$$

Below, we let T_1 and T_2 be the set of frequent queries on the source group and the target group, respectively. For ease of exposition, all the queries that the algorithm calls to $\mathcal{G}$ in the addition procedure $A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y))$ are denoted as

$$(at_1^{L_1(x),L_1(y)}, \ldots, at_{q_{\mathsf{ind}}}^{L_1(x),L_1(y)}) \xleftarrow{q} A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y)).$$

Similarly, all the queries that the algorithm calls to $\mathcal{G}$ in the multiplication procedure $M_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y))$ are denoted as

$$(mt_1^{L_1(x),L_1(y)}, \ldots, mt_{q_{\mathsf{ind}}}^{L_1(x),L_1(y)}) \xleftarrow{q} M_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y)),$$

Finally, we consider a scaling procedure Scale, which takes $(L_1^{\mathcal{G}}(x), u)$ as inputs and outputs $L_1^{\mathcal{G}}(ux)$. Scale is computed by at most $2\log p$ addition procedures. All the queries that the algorithm calls to $\mathcal{G}$ in $\mathsf{Scale}^{\mathcal{G}}(L_1^{\mathcal{G}}(x), u)$ are denoted as:

$$(st_1^{L_1(x),u}, \ldots, st_{2\log p q_{\mathsf{ind}}}^{L_1(x),u}) \xleftarrow{q} \mathsf{Scale}^{\mathcal{G}}(L_1^{\mathcal{G}}(x), u).$$

Next, we define a new notion called "extractable queries", a component of constructing $\mathsf{Extractor\text{-}1}$ and computing F_x. For fixed $L_1^{\mathcal{G}}(x)$, consider $\mathsf{Extractor\text{-}1}$ sampling y and computing $L_1^{\mathcal{G}}(y)$. Write $(at_1, \ldots, at_{q_{\mathsf{ind}}}) \xleftarrow{q} A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y))$. Then we say query $(*, \mathsf{label}, \mathcal{G}(z))$ is an extractable query of $L_1^{\mathcal{G}}(x)$ in the addition procedure $A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y))$, denoted as $(*, \mathsf{label}, \mathcal{G}(z)) \overset{\mathsf{ext}}{=} A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y))$, if:

1. $\mathcal{G}(z) \notin Q_y^1 \cup T_1 \cup T_2$;
2. $\exists\ at_i, i \in [1, q_{\mathsf{ind}}]$ s.t. $at_i = (\mathcal{G}(z), \mathcal{G}(z'), \mathsf{add}, \mathcal{G}(z + z'))$;
3. $\mathcal{G}(z)$ never appears in the previous addition queries $at_1, \ldots, at_{i-1}$.

The extractable queries of $L_1^{\mathcal{G}}(x)$ in the multiplication and scaling procedures can be defined analogously.

Now, we are ready to describe the frequently extractable query of $L_1^{\mathcal{G}}(x)$, and thus build F_x. Specifically, we say $(*, \mathsf{label}, \mathcal{G}(z))$ is a frequently extractable query of $L_1^{\mathcal{G}}(x)$ for the addition procedure if

$$\Pr[(*, \mathsf{label}, \mathcal{G}(z)) \overset{\mathsf{ext}}{=} A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y)) : y \xleftarrow{\$} \mathbb{Z}_p] \geq \frac{1}{\bar{q}}.$$

Similarly, $(*, \mathsf{label}, \mathcal{G}(z))$ is a frequently extractable query of $L_1^{\mathcal{G}}(x)$ for scaling if:

$$\Pr[(*, \mathsf{label}, \mathcal{G}(z)) \stackrel{\mathsf{ext}}{=} \mathsf{Scale}^{\mathcal{G}}(L_1^{\mathcal{G}}(x), u) : u \stackrel{\$}{\leftarrow} \mathbb{Z}_p] \geq \frac{1}{q},$$

and $(*, \mathsf{label}, \mathcal{G}(z))$ is a frequently used query of $L_1^{\mathcal{G}}(x)$ for multiplication if:

$$\Pr[(*, \mathsf{label}, \mathcal{G}(z)) \stackrel{\mathsf{ext}}{=} M_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y)) : y \stackrel{\$}{\leftarrow} \mathbb{Z}_p] \geq \frac{1}{q}.$$

Next, we give the formal description of Extractor-1, in Figure 7, which takes $L_1^{\mathcal{G}}(x)$ as inputs and outputs F_x and C_G, where C_G denotes the set of the linear constraints.

$\mathsf{Extractor\text{-}1}(L_1^{\mathcal{G}}(x))$:
$T_1, T_2 \leftarrow \{(1, \mathcal{G}(1))\}; F_x \leftarrow \{(1, \mathcal{G}(1))\}, G \leftarrow \{\mathcal{G}(1)\}, C_G \leftarrow (\mathcal{G}(1) = 1);$
$y_1, \ldots y_s, w_1, \ldots, w_s, u_1, \ldots, u_s, v_1, \ldots, v_s \stackrel{\$}{\leftarrow} \mathbb{Z}_p;$
for $i = 1$ to s
  $Q^1_{y_i} = (t_1, \ldots, t_{q_{\mathsf{ind}}}) \stackrel{q}{\leftarrow} L_1^{\mathcal{G}}(y_i), Q^2_{w_i} = (t_{q_{\mathsf{ind}}+1}, \ldots, t_{2q_{\mathsf{ind}}}) \stackrel{q}{\leftarrow} L_2^{\mathcal{G}}(w_i);$
  $T_1 \leftarrow T_1 \cup Q^1_{y_i}, T_2 \leftarrow T_2 \cup Q^2_{w_i}$ *//computes T_1 and T_2*
  for $j = 1$ to $2q_{\mathsf{ind}}$; *//collects all the group elements that appears into G*
    $t_j = (z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z)), G \leftarrow G \cup \{\mathcal{G}^{\mathsf{label}}(z)\}, C_G \leftarrow C_G \cup (\mathcal{G}^{\mathsf{label}}(z) = z);$
  $(t_{2q_{\mathsf{ind}}+1}, \ldots, t_{3q_{\mathsf{ind}}}) \stackrel{q}{\leftarrow} A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y_i)),$
  $(t_{3q_{\mathsf{ind}}+1}, \ldots, t_{4q_{\mathsf{ind}}}) \stackrel{q}{\leftarrow} M_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(v_i))$
  $(t_{4q_{\mathsf{ind}}+1}, \ldots, t_{(2\log p+4)q_{\mathsf{ind}}}) \stackrel{q}{\leftarrow} \mathsf{Scale}(L_1^{\mathcal{G}}(x), u_i),$
  for $j = 2q_{\mathsf{ind}}+1$ to $(2\log p+4)q_{\mathsf{ind}}$ *//identifies all the extractable queries that ever appears*
    if $t_j = (z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z)), G \leftarrow G \cup \{\mathcal{G}^{\mathsf{label}}(z)\}, C_G \leftarrow C_G \cup (\mathcal{G}^{\mathsf{label}}(z) = z);$
    if $t_j = (\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2), \mathsf{add}, \mathcal{G}^{\mathsf{label}}(z_1 + z_2)),$
      $G \cup \{\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2), \mathcal{G}^{\mathsf{label}}(z_1 + z_2)\},$
      $C_G \leftarrow C_G \cup (\mathcal{G}^{\mathsf{label}}(z_1) + \mathcal{G}^{\mathsf{label}}(z_2) = \mathcal{G}^{\mathsf{label}}(z_1 + z_2))$
      if $\mathcal{G}^{\mathsf{label}}(z_1) \stackrel{\mathsf{used}}{=} L_1^{\mathcal{G}}(x)$, $F_x \leftarrow F_x \cup \mathcal{G}^{\mathsf{label}}(z_1)$
      if $\mathcal{G}^{\mathsf{label}}(z_2) \stackrel{\mathsf{used}}{=} L_1^{\mathcal{G}}(x)$, $F_x \leftarrow F_x \cup \mathcal{G}^{\mathsf{label}}(z_2)$
return F_x, T_1, T_2, G, C_G.

Fig. 7. Extractor for F_x.

Applying the same analysis in Theorem 1, we have for sufficiently large s that T_1 and T_2 will record all the frequent queries for the source and target groups (with probability $\geq 1 - \frac{2\lambda}{e^\lambda}$), and and F_x will record all the frequently extractable queries of $L_1^{\mathcal{G}}(x)$ (with probability $\geq 1 - \frac{(2\log p+2)\lambda}{e^\lambda}$) (here we abuse the notation that F_x only records the group element $\mathcal{G}(z)$, rather than the tuple $(*, \mathsf{label}, \mathcal{G}(z))$).

Next, we claim that $F_x \setminus \{\mathcal{G}(1)\} \subseteq D_x$ with overwhelming probability. Indeed, $L_1^{\mathcal{G}}(x)$ is independent of any query $q \notin D_x$, and by the description of Extractor-1,

every element in F_x is extracted in the addition, multiplication and scaling procedure. Thus, as long as the generic group model ($\mathbb{Z}_p \to S$) has long outputs, say $\log|S| \geq \log p + 2\lambda$ (which means it is hard to predict a valid encoding in S without making a labeling query), those procedures would not involve any independent queries (without making labeling query) except for negligible probability ($\leq \frac{1}{2^\lambda}$). Hence, every element in F_x would be relevant to $L_1^{\mathcal{G}}(x)$, implying that $F_x \setminus \{\mathcal{G}(1)\} \subseteq D_x$.

Moreover, by the description of Extractor-1, we have that Extractor-1 also outputs C_G, a set of linear constraints on G. Based on C_G, we can compute some linear constraints on F_x, denoted as $C^1_{F_x}$. Write $|C^1_{F_x}|$ to mean the total number of *linearly independent* equations on F_x. Then if $|C^1_{F_x}| = |F_x|$, we can trivially decode F_x by solving the linear system. For example, if $F_x = \{(*, \mathcal{G}(z_1)), (*, \mathcal{G}(z_2))\}$ and $C^1_{F_x} = \{(\mathcal{G}(z_1)) = \mathcal{G}(z_2)) \oplus \mathcal{G}(z_2))), (\mathcal{G}(6)) = \mathcal{G}(z_1)) \oplus \mathcal{G}(z_2)))\}$, then we can solve the linear system and get the values $z_1 = 4$ and $z_2 = 2$. We use F_x^* to denote the decoded table of F_x and note that every tuple in F_x^* has the form $(z_1, \mathcal{G}(z_1))$.

Next, we show that if F_x^* can be efficiently computed, then we can construct $(B^{\mathcal{G}}_{\mathsf{real}}, P_{\mathsf{real}})$ straightforwardly. Here is the description, in Figure 8, which takes input $(L_1^{\mathcal{G}}(x), F_x^*, T_1, T_2)$.

$\underline{B^{\mathcal{G}}_{\mathsf{real}}(L_1^{\mathcal{G}}(x), F_x^*, T_1, T_2)}$:
$y \xleftarrow{\$} \mathbb{Z}_p, T \leftarrow T_1 \cup T_2, Q_y \xleftarrow{\mathsf{q}} L_1^{\mathcal{G}}(y)$,
$(t_1, \ldots, t_{q_{\mathsf{ind}}}) \xleftarrow{\mathsf{q}} A_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x), L_1^{\mathcal{G}}(y))$;
for $i = 1$ to q_{ind},
  if $t_i = (z, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z)), T \leftarrow T \cup t_i$,
  if $t_i = (\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2), \mathsf{add}, \mathcal{G}^{\mathsf{label}}(z_1 + z_2))$,
    if $\{\mathcal{G}^{\mathsf{label}}(z_1), \mathcal{G}^{\mathsf{label}}(z_2)\} \not\subset Q_y \cup F_x^* \cup T_1 \cup T_2$, abort,
    else $t_i \leftarrow (z_1 + z_2, \mathsf{label}, \mathcal{G}^{\mathsf{label}}(z_1 + z_2)), T \leftarrow T \cup t_i$;
    //every tuple recorded in T is labeling query
Return $P_{\mathsf{real}}(\cdot, T, y, L_1^{\mathcal{G}}(x + y))$.

$\underline{P_{\mathsf{real}}(z, T, y, u^*)}$:
$u \leftarrow L_1^T(z + y)$;
if $u = u^*$, return 1;
return 0.

Fig. 8. Algorithms in the real world.

Immediately observe that $B^{\mathcal{G}}_{\mathsf{real}}$ aborts only if:

1. some extractable queries of $L_1^{\mathcal{G}}(x)$ appears in $(t_1, \ldots, t_{q_{\mathsf{ind}}})$;
2. one of them is not frequently extractable.

By the definition of frequently-extractable and F_x^*, this bad event is trivially bounded by $\leq \frac{q_{\mathsf{ind}}}{\bar{q}} = \frac{1}{q^*} = \frac{1}{30}$. Thus, applying the analysis in Theorem 1, we have that

$$\Pr[P_{\mathsf{real}}(x) = 1] \geq 1 - \frac{1}{30} - \frac{1}{q^*} - \frac{(2\log p + 4)\lambda}{e^\lambda} - \frac{1}{2^\lambda} - \epsilon \geq \frac{9}{10},$$
$$\Pr_{z \neq x}[P_{\mathsf{real}}(z) = 1] \leq 2\sqrt{\epsilon} \leq \mathsf{negl}.$$

However, it is possible that F_x^* is *not* efficiently computable; for instance, $C^1_{F_x}$ might only have a few linear constraints or none. This is inherent, since so far we have not used the pairing at all. To compute F_x^*, we extract more equations using the algorithm Extractor-2, which makes use of the pairing.

First, in the scaling procedure $\mathsf{Scale}(L_1^{\mathcal{G}}(x), u)$, if u is known but x is unknown, then the group elements in D^1_{ux} can be represented as a linear system as follows (with overwhelming probability):

$$D^1_{ux} = \bar{M}^1_{ux} F_x + \bar{M}^2_{ux}(T_1 \cup T_2),$$

Here, we here abuse the notation D^1_{ux} as a vector of the group elements), and $\bar{M}^1_{ux}$ and $\bar{M}^2_{ux}$ are explicitly known. This follows straightforwardly by the same analysis in Theorem 1. By this linear system, we know that every group element in D^1_{ux} would be represented by a linear combination of the elements in $F_x \cup T_1 \cup T_2$, with high probability. Moreover, $(F_{ux} \setminus \{\mathcal{G}(1)\} \subseteq D_{ux}$ and $(1, \mathcal{G}(1)) \in T_1 \cup T_2$, thus we have:

$$F_{ux} = M^1_{ux} F_x + M^2_{ux}(T_1 \cup T_2),$$

where M^1_{ux} and M^2_{ux} can be computed from $\bar{M}^1_{ux}$ and $\bar{M}^2_{ux}$, by cutting off some rows.

Next, we leverage the power of the pairing to gain more linear constraints. In generic bilinear maps, $L_2(xyz) = M_1^{\mathcal{G}}(L_1(xy), L_1(z)) = M_1^{\mathcal{G}}(L_1(x), L_1(yz))$. Hence, with high probability we can represent the elements in D^2_{xyz} as follows:

$$\begin{aligned} D^2_{xyz} &= M_1^{xy} \boxed{F_{xy}} + M_2^z Q_z + \bar{M}_3^{x,y,z}(T_1 \cup T_2) \\ &= M_1^{xy} \boxed{M_{yx} F_x} + M_2^z Q_z + M_3^{x,y,z}(T_1 \cup T_2) \\ &= \boxed{M_1^{xy} M_{yx}} F_x + M_2^z Q_z + M_3^{x,y,z}(T_1 \cup T_2) \\ &= \boxed{M_4^x} F_x + M_5^{yz} Q_{yz} + M_6^{x,y,z}(T_1 \cup T_2). \end{aligned}$$

Note that, if $M_1^{xy} M_{yx} \neq M_4^x$, then we can gain constraints on F_x as follows:

$$(M_1^{xy} M_{yx} - M_4^x) F_x = M_5^{yz} Q_{yz} - M_2^z Q_z + (M_6^{x,y,z} - M_3^{x,y,z})(T_1 \cup T_2). \quad (1)$$

Moreover, y, z are chosen by the algorithm itself, it means that every term on the right side is explicitly known, by which we have that the new constraints can be efficiently computed. In the following, we denote $(c_1, \ldots, c_k) \xleftarrow{\mathsf{ext}} \mathsf{Ext}(L_1(x), F_x, y, z)$ as the linear constraints that extracted by Equation 1. Next, we present the description of Extractor-2, in Figure 9.

Observe that the goal of Extractor-2 is to gain as many linear constraints on F_x as possible. A more interesting property is that, when Extractor-2 finishes, then it would be hard to extract an additional linear-independent constraint, specifically,

$$\Pr_{y,z}[\mathsf{Ext}(L_1^{\mathcal{G}}(x), F_x, y, z) \text{ extracts a new constraint on } F_x] \leq \frac{\lambda}{e^\lambda} + \frac{1}{q}.$$

$\underline{\mathsf{Extractor\text{-}2}(L_1^{\mathcal{G}}(x), F_x, C^1_{F_x})}$:

$y_1, \ldots, y_s, z_1 \ldots, z_s \xleftarrow{\$} \mathbb{Z}_p, C^2_{F_x} \leftarrow C^1_{F_x}$;
for $i = 1$ to s,
  $(c_1^i, \ldots, c_k^i) \xleftarrow{\mathsf{ext}} \mathsf{Ext}(L_1^{\mathcal{G}}(x), F_x, y_i, z_i)$; $C^2_{F_x} \leftarrow C^2_{F_x} \cup \{c_1^i, \ldots, c_k^i\}$;
Return $C^2_{F_x}$

Fig. 9. Additional extractor for F_x.

We now show how to use this property to build $(B_{\mathsf{real}}, P_{\mathsf{real}})$ in case that $C^2_{F_x}$ is still not good enough. Again, if we can compute F_x^* by $C^2_{F_x}$, then we are done, as in Figure 8. Unfortunately, even with $C^2_{F_x}$, F_x^* might not be efficiently computed. In such a situation, we then propose a new method that builds $(B^{\mathcal{G}}_{\mathsf{real}}, P_{\mathsf{real}})$ in an alternative way, by using the property we illustrated above.

Let $f_1 = |F_x|, f_2 = |C^2_{F_x}|$. Given F_x and $C^2_{F_x}$, it is easy to compute an $(f_1 - f_2) \times f_1$ matrix P^x_{ext} and a new set of group elements $\hat{F}_x$ such that:

1. $|\hat{F}_x| = f_1 - f_2$; $F_x = P^x_{\mathsf{ext}} \hat{F}_x$;
2. elements in $\hat{F}_x$ are linearly independent.

Note that $\hat{F}_x$ is a projection of F_x based on $C^2_{F_x}$. Moreover, we have that extracting a linear constraint on $\hat{F}_x$ is identical to extracting an additional linear-independent constraint on F_x after $\mathsf{Extractor\text{-}2}$ finishes. Thus, we have that, for any $w, v \in \mathbb{Z}_p$,

$$\begin{aligned} D^2_{xwv} &= M_1^{xw} M_{xw} F_x + M_2^v Q_v + M_3^{x,w,v}(T_1 \cup T_2) \\ &= M_1^{xw} M_{xw} P^x_{\mathsf{ext}} \hat{F}_x + M_2^v Q_v + M_3^{x,w,v}(T_1 \cup T_2) \\ &= M_4^x F_x + M_5^{wv} Q_{wv} + M_6^{x,w,v}(T_1 \cup T_2) \\ &= M_4^x P^x_{\mathsf{ext}} \hat{F}_x + M_5^{wv} Q_{wv} + M_6^{x,w,v}(T_1 \cup T_2). \end{aligned}$$

Observe that with high probability, we must have

$$M_1^{xw} M_{xw} P_{\mathsf{ext}} = M_4^x P_{\mathsf{ext}}. \tag{2}$$

In fact, if Equation 2 does not hold, it immediately gives at least one linear constraint on $\hat{F}_x$, which would have resulted in an additional linear-independent constraint on F_x after $\mathsf{Extractor\text{-}2}$ finishes.

Of course, Equation 2 would not directly help us to compute F_x^*, but it opens up an alternative way to build $(B^{\mathcal{G}}_{\mathsf{real}}, P_{\mathsf{real}})$. Let $L_1^{\mathcal{G}}(x)$ be the challenge input. We first sample x', y', and in our new strategy, we would then attempt to compute $F^*_{x'x}$, rather than F_x^* directly. Specifically, given $L_1^{\mathcal{G}}(x')$, we compute $F_{x'}, C^2_{F_{x'}}$, and $P^{x'}_{\mathsf{ext}}$ as above (here we know everything of $F_{x'}$ as x' is sampled). Then we have that, with high probability

$$F_{x'x} = M_{x'x} F_{x'} + M'(T_1 \cup T_2) = M_{x'x} P^{x'}_{\mathsf{ext}} \hat{F}_{x'} + M'(T_1 \cup T_2)$$

$$M_1^{x'x} M_{x'x} P^{x'}_{\mathsf{ext}} = M_4^{x'} P^{x'}_{\mathsf{ext}}$$

where we treat the unknown challenge term x as the scaling parameter. Now we see that, if we can compute the matrix $M_{x'x}P_{\mathsf{ext}}^{x'}$, then $F_{x'x}^*$ can be efficiently computed (as x' is sampled by the algorithm, even Q_x^1 is explicitly known). Again, once we compute $F_{x'x}^*$, we are done.

However, $M_{x'x}P_{\mathsf{ext}}^{x'}$ is computable if and only if $M_1^{x'x}$ is a row full rank matrix, which might not be true. It remains therefore to handle the scenario that $M_1^{x'x}$ is not row full rank. Observe that if $M_1^{x'x}$ has a kernel (not a row full rank matrix), it means that the multiplication procedure $M_1^{\mathcal{G}}(L_1^{\mathcal{G}}(x'x), L_1^{\mathcal{G}}(y'))$ only needs partial information about $F_{x'x}$. Thus, instead of computing $F_{x'x}^*$ (which is impossible in this case), we only compute this kind of partial information of $F_{x'x}^*$. Concretely, let K^* be the kernel of $M_1^{x'x}$, an $f_1 \times n$ matrix ($n \leq f_1$). We then compute K such that $[K|K^*]$ is a square ($f_1 \times f_1$) and invertable matrix and $U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$[13] as its inverse. Then

$$\begin{aligned} M_1^{x'x}M_{x'x}P_{\mathsf{ext}}^{x'} &= M_1^{x'x}[K|K^*]\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} M_{x'x}P_{\mathsf{ext}}^{x'} \\ &= [M_1^{x'x}K|0]\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} M_{x'x}P_{\mathsf{ext}}^{x'} \\ &= (M_1^{x'x}K)U_1M_{x'x}P_{\mathsf{ext}}^{x'}. \end{aligned}$$

Now we see that $M_1^{x'x}K$ is definitely row full rank, which means that we can efficiently compute $U_1M_{x'x}P_{\mathsf{ext}}^{x'}$. And now we specify the partial information as $\tilde{F}_{x'x} := U_1F_{x'x}^*$ and prove that $\tilde{F}_{x'x}$ itself is sufficient to transfer every adding query into labeling query. Concretely,

$$\begin{aligned} D_{xx'y'}^2 &= M_1^{x'x}F_{x'x} + M_2^{y'}F_{y'} + M_3^{x,x',y'}(T_1 \cup T_2) \\ &= (M_1^{x'x}K)U_1F_{x'x} + M_2^{y'}F_{y'} + M_3^{x,x',y'}(T_1 \cup T_2) \\ &= (M_1^{x'x}K)\tilde{F}_{x'x} + M_2^{y'}F_{y'} + M_3^{x,x',y'}(T_1 \cup T_2). \end{aligned}$$

We know everything about $\hat{F}_{x'}$ since x' is chosen by the algorithm itself, so we can compute $\tilde{F}_{x'x}$ as

$$\tilde{F}_{x'x} = U_1M_{x'x}P_{\mathsf{ext}}^{x'}\hat{F}_{x'}.$$

And again, once we have $\tilde{F}_{x'x}$, we are done. Combining together, we establish the entire proof. □

Due to space limit, we give our indifferentiable GGM from GBM in Appendix C. In fact, we illustrate a more general result in Appendix C, which states the following: for any integer d_1, d_2, if d_1 divides d_2, then we can build an indifferentiable d_1-linear MMaps from an idealized d_2-linear MMaps statistically.

[13] U_1 is an $(f_1 - n) \times f_1$ matrix.

Acknowledgements

We thank Prof. Mohammad Mahmoody, Prof. David Wu and Fermi Ma for the insightful discussions and comments on this paper. Prof. Mark Zhandry is supported by an NSF CAREER award. Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF.

References

ABD+13. Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the indifferentiability of key-alternating ciphers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 531–550. Springer, Heidelberg, August 2013.

AY20. Shweta Agrawal and Shota Yamada. Optimal broadcast encryption from pairings and LWE. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 13–43. Springer, Heidelberg, May 2020.

BB04. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Heidelberg, May 2004.

BF01. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

BF18. Manuel Barbosa and Pooya Farshim. Indifferentiable authenticated encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 187–220. Springer, Heidelberg, August 2018.

BGK+14. Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 221–238. Springer, Heidelberg, May 2014.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

BR14. Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 1–25. Springer, Heidelberg, February 2014.

BS02. Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2002.

CDMP05. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Heidelberg, August 2005.

CDMS10. Jean-Sébastien Coron, Yevgeniy Dodis, Avradip Mandal, and Yannick Seurin. A domain extender for the ideal cipher. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 273–289. Springer, Heidelberg, February 2010.

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

CHK+16. Jean-Sébastien Coron, Thomas Holenstein, Robin Künzler, Jacques Patarin, Yannick Seurin, and Stefano Tessaro. How to build an ideal cipher: The indifferentiability of the Feistel construction. *Journal of Cryptology*, 29(1):61–114, January 2016.

DRS09. Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for practical applications. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 371–388. Springer, Heidelberg, April 2009.

DSSL16. Yevgeniy Dodis, Martijn Stam, John P. Steinberger, and Tianren Liu. Indifferentiability of confusion-diffusion networks. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 679–704. Springer, Heidelberg, May 2016.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GW11. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

HSW14. Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 201–220. Springer, Heidelberg, May 2014.

IR89. Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM STOC*, pages 44–61. ACM Press, May 1989.

LR88. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2), 1988.

Mau05. Ueli Maurer. Abstract models of computation in cryptography. In *IMA International Conference on Cryptography and Coding*, pages 1–12. Springer, 2005.

MRH04. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

Nao03. Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, August 2003.

PS16. Rafael Pass and Abhi Shelat. Impossibility of vbb obfuscation with ideal constant-degree graded encodings. In *Theory of Cryptography Conference*, pages 3–17. Springer, 2016.

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

Zha20. Mark Zhandry. New techniques for traitor tracing: Size $N^{1/3}$ and more from pairings. In Daniele Micciancio and Thomas Ristenpart, editors,

CRYPTO 2020, Part I, volume 12170 of *LNCS*, pages 652–682. Springer, Heidelberg, August 2020.

ZZ20. Mark Zhandry and Cong Zhang. Indifferentiability for public key cryptosystems. In *Annual International Cryptology Conference*, pages 63–93. Springer, 2020.

A Proof of Theorem 2

Proof. According to the definition of indifferentiability, the adversary has one honest interface $\mathcal{H}$ and two adversarial interfaces $\mathcal{G}^{\mathsf{label}}$ and $\mathcal{G}^{\mathsf{add}}$. Therefore we need build an efficient simulator $\mathcal{S}$ in the ideal world that can simulate the adversarial interfaces properly. We will go through a sequence of hybrid games where in each game, there exists a system that responds to all of the queries (both honest and adversarial) in a slightly different way and then we build our simulator $\mathcal{S}$ as the system in the last game. Before the description of the games, we first specify some parameters:

- There are three types of queries: $(x;\mathcal{H})$, $(y,\mathsf{label};\mathcal{G})$ and $(\mathsf{Z}_0,\mathsf{Z}_1,\mathsf{add};\mathcal{G})$, where $x \leftarrow \{0,1\}^n, y \leftarrow \mathbb{Z}_p$ and $\mathsf{Z}_0,\mathsf{Z}_1 \leftarrow S$.
- The adversary only makes q queries to the system, where $q = \mathsf{poly}(\lambda)$.
- The oracles used in the real world are $\tilde{\mathcal{H}}$, $\tilde{\mathcal{G}}^{\mathsf{label}}$ and $\tilde{\mathcal{G}}^{\mathsf{add}}$;
- In each game, the system's responses are denoted as $\mathcal{H}^{\mathsf{r}}$, $\mathcal{G}^{\mathsf{rlabel}}$ and $\mathcal{G}^{\mathsf{radd}}$. For instance, $\mathcal{G}^{\mathsf{radd}}(\mathsf{Z}_0,\mathsf{Z}_1)$ denotes the system's response when adversary makes a query $Q = (\mathsf{Z}_0,\mathsf{Z}_1,\mathsf{add};\mathcal{G})$.

The hybrid games are as follows.

Game 0. This game is identical to the real game except that the system maintains two table, referring to L-table and A-table(for labeling and addition queries, respectively). Specifically, the system responds to the queries the same as in the real world, for instance, $\mathcal{H}^{\mathsf{r}}(x) = \tilde{\mathcal{H}}(x)$ and $\mathcal{G}^{\mathsf{radd}}(\mathsf{Z}_0,\mathsf{Z}_1) = \tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0,\mathsf{Z}_1)$. For the tables, the system maintains them as follows:

- L-table: Initially empty, consists of tuples with form of (x,Z). Once the adversary makes a label query, say $(x,\mathsf{Label};\mathcal{G})$, which does not exist in L-table yet(no tuple such that the first element is x), the system inserts $(x,\tilde{\mathcal{G}}^{\mathsf{label}}(x))$ into L-table.
- A-table: Initially empty, consists of tuples with form of $(\mathsf{Z}_0,\mathsf{Z}_1,\mathsf{Z})$. Once the adversary makes an addition query, say $(\mathsf{Z}_0,\mathsf{Z}_1,\mathsf{add};\mathcal{G})$, which does not exist in A-table yet(no tuple such that the first elements are $(\mathsf{Z}_0,\mathsf{Z}_1)$ or $(\mathsf{Z}_1,\mathsf{Z}_0)$), the system inserts $(\mathsf{Z}_0,\mathsf{Z}_1,\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0,\mathsf{Z}_1))$ into A-table. Moreover, if there exist (x_0,Z_0) and (x_1,Z_1) in the previous queries, then the system also inserts $(x_0+x_1,\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0,\mathsf{Z}_1))$ into L-table.

Note that at this point all the queries are answered by the real oracles and these tables are just keeping track of information related to adversary's queries (to the adversarial interfaces) and completely hidden to the adversary. Hence the view in real game is identical to the one in Game 0. Next, we illustrate an alternative way to responds to part of the queries, by using these tables and accessing to the honest interface.

For ease of exposition, we here define a relation between the query Q and the table Tab. Specifically, if Q is a labeling query, for instance $Q = (x,\mathsf{label};\mathcal{G})$, we say $Q \in \mathsf{Tab}$ if there exists a tuple $T = (T_1,T_2) \in \mathsf{Tab}$ such that $T_1 = x$. Analogously, if Q is an addition query, for instance $Q = (\mathsf{Z}_0,\mathsf{Z}_1,\mathsf{add};\mathcal{G})$, we say

$Q \in \mathsf{Tab}$, if there exists a tuple $T = (T_1, T_2, T_3) \in \mathsf{Tab}$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$.

Game 1. This game is identical to Game 0, except the way of maintaining the tables and responding to the queries. Specifically,
Labeling query. Suppose $Q_k = (x, \mathsf{label}; \mathcal{G})$(the k-th query), then

- Case 1. If $Q_k \in \mathsf{L}$, which means there exists a tuple $T = (T_1, T_2) \in \mathsf{L}$ such that $T_1 = x$, then the system responds to the query with T_2.
- Case 2. Otherwise, the system responds with $\tilde{\mathcal{G}}^{\mathsf{label}}(x)$ and inserts $(x, \tilde{\mathcal{G}}^{\mathsf{label}}(x))$ into L-table.

Addition query. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, then

- Case 1. If $Q_k \in \mathsf{A}$, which means there exist a tuple $T = (T_1, T_2, T_3)$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$, then the system responds to the query with T_3.
- Case 2. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T_1', T_2')$ such that $T_2 = \mathsf{Z}_0$ and $T_2' = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T_2' = \mathsf{Z}_0$), and $T_1 + T_1' \in \mathsf{L}$, then the system responds with the corresponding record.
- Case 3. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T_1', T_2')$ such that $T_2 = \mathsf{Z}_0$ and $T_2' = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T_2' = \mathsf{Z}_0$), but $T_1 + T_1' \notin \mathsf{L}$, then the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$ and inserts $(T_1 + T_1', \tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1))$ into L-table.
- Case 4. Otherwise, the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$.

Note that, in Game 1 the system keeps a longer table, and for part of the queries, the system responds to them in an alternative way, which is *only* using the tables and the honest interfaces. Moreover, in Game 1, the tuples stored in the tables correspond to the response of queries that are answered by the real oracles. Hence, in either game, the response of any query is identical, which refers to that the view in Game 1 is identical to the one in Game 0. However, the system can only answer *part of* the queries by tables and honest interfaces, and for the rest it has to call the real oracles. Thus, in the following hybrid games, we will illustrate additional alternative ways(not calling the real oracles) to respond to the rest queries, without changing the view significantly.

Game 2. This game is identical to Game 1, except for responding to the addition queries. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{A}$, which means there exist a tuple $T = (T_1, T_2, T_3)$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$, then same as in Game 1.
- Case 2. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T_1', T_2')$ such that $T_2 = \mathsf{Z}_0$ and $T_2' = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T_2' = \mathsf{Z}_0$), and $T_1 + T_1' \in \mathsf{L}$, then same as in Game 1.
- Case 3. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T_1', T_2')$ such that $T_2 = \mathsf{Z}_0$ and $T_2' = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T_2' = \mathsf{Z}_0$), but $T_1 + T_1' \notin \mathsf{L}$, then same as in Game 1.
- Case 4. Otherwise, the system responds with $\perp$.

Note that the only difference between Game 1 and Game 2 occurs in Case 4, where one of $(\mathsf{Z}_0, \mathsf{Z}_1)$ never appears in the labeling query, say Z_0. We immediately observe that, if Z_0 is not a valid group element, then Game 1 and Game 2 are identical. In fact, the honest interface only gives the adversary the first n bits and due to $\log |S| \geq \log p + n + \lambda$, Z_0 is non-valid except with negligible probability.
Game 3. This game is identical to Game 2, except for responding to the addition queries. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{A}$, which means there exist a tuple $T = (T_1, T_2, T_3)$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$, then same as in Game 2.
- Case 2. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), and $T_1 + T'_1 \in \mathsf{L}$, then same as in Game 2.
- Case 3. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), but $T_1 + T'_1 \notin \mathsf{L}$, then the system makes a query $(T_1 + T'_1; \mathcal{H})$ and samples a random element $\mathsf{Z} \in S$ conditioned that $\mathsf{Trunc}_{\mathsf{n}}(Z) = \mathcal{H}^{\mathsf{r}}(T_1 + T'_1)$, and responds the query with Z. After that, the system inserts $(T_1 + T'_1, \mathsf{Z})$ into L-table.
- Case 4. Otherwise, same as in Game 2.

Note that the only difference between Game 2 and Game 3 occurs in Case 3. In this case, Z_0 and Z_1, of course, are valid group element and the system is obligated to output a proper response, which should be consistent to the honest interface. Essentially, we observe that, as $T_1 + T'_1 \in \mathsf{L}$-table, the adversary can only learn the first n bits of $\tilde{\mathcal{G}}^{\mathsf{label}}(T_1 + T'_1)$, and the rest bits are independent of the adversary's view. Thus, it is okay to replace them with random strings, as long as Z never appears before Q_k, which can only happen with negligible probability.
Game 4. This game is identical to Game 3, except for responding to the labeling queries. Suppose $Q_k = (\mathsf{x}, \mathsf{label}; \mathcal{G})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{L}$, which means there exists a tuple $T = (T_1, T_2) \in \mathsf{L}$ such that $T_1 = x$, then same as in Game 3.
- Case 2. Otherwise, the system makes a query $(x; \mathcal{H})$ and samples a random element $\mathsf{Z} \in S$ conditioned that $\mathsf{Trunc}_{\mathsf{n}}(Z) = \mathcal{H}^{\mathsf{r}}(x)$, and responds the query with Z. After that, the system inserts (x, Z) into L-table.

Trivial to note that, Game 4 is close to Game 3 within the same reason above(Game 3 $\approx$ Game 2).
Game 5. In Game 4, the queries to the adversarial interfaces are answered by the tables which are maintained by the system and by making queries to honest interface. The system never makes queries directly to $\tilde{\mathcal{G}}^{\mathsf{label}}$ and $\tilde{\mathcal{G}}^{\mathsf{add}}$; these oracles are *only* used to answer the $\mathcal{H}$ queries (either generated by the adversary or by the system's response to labeling or addition queries). At this point, it is straightforward to show that we can replace $\mathcal{H}$ with a random oracle $\mathcal{O}$, resulting in Game 5.

We note that in Game 5, the system is efficient, and it responds to the adversarial interfaces just by keeping several tables and calling the honest interfaces. Thus, we can build a simulator that responds to the honest and adversarial queries precisely as the system does in Game 5. The result is that the view in Game 5 is identical to the ideal world and it suffices to prove that any adjacent games are indistinguishable.

In the following, we first give the full description of the simulator and then complete the proof by showing that every pair of adjacent games are indistinguishable.

Simulator in Ideal Game. Let $\mathcal{O}$ be a random oracle, and we define the simulator $\mathcal{S}$ works as follows. Like the system in Game 5, the simulator also maintains two tables: the labeling table L-table and the addition table A-table. Concretely:

- L-table: Initially empty and consists of tuples with form of (x, Z);
- A-table: Initially empty and consists of tuples with form of $(\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{Z})$ or $(\mathsf{Z}_0, \mathsf{Z}_1, \perp)$.

By definition, $\mathcal{S}$ has access to the honest interfaces $\mathcal{O}$. And for the adversarial queries, $\mathcal{S}$ works as the system in Game 5, by just using the tables and calling the honest interfaces.

Labeling query. Suppose $Q_k = (x,)$(the k-th query), then

- Case 1. If $Q_k \in \mathsf{L}$, which means there exists a tuple $T = (T_1, T_2) \in \mathsf{L}$ such that $T_1 = x$, then the simulator responds to the query with T_2.
- Case 2. Otherwise, the simulator makes a query $(x; \mathcal{H})$ and samples a random element $\mathsf{Z} \in S$ conditioned that $\mathsf{Trunc}_\mathsf{n}(Z) = \mathcal{H}^\mathsf{r}(x)$, and responds the query with Z. After that, the simulator inserts (x, Z) into L-table.

Addition query. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, then

- Case 1. If $Q_k \in \mathsf{A}$, which means there exist a tuple $T = (T_1, T_2, T_3)$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$, then the simulator responds to the query with T_3.
- Case 2. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), and $T_1 + T'_1 \in \mathsf{L}$, then the simulator responds with the corresponding record.
- Case 3. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), but $T_1 + T'_1 \notin \mathsf{L}$, then the simulator makes a query $(T_1 + T'_1; \mathcal{H})$ and samples a random element $\mathsf{Z} \in S$ conditioned that $\mathsf{Trunc}_\mathsf{n}(Z) = \mathcal{H}^\mathsf{r}(T_1 + T'_1)$, and responds the query with Z. After that, the simulator inserts $(T_1 + T'_1, \mathsf{Z})$ into L-table.
- Case 4. Case 4. Otherwise, the simulator responds with $\perp$.

Next we prove the indistinguishability between any adjacent games, by showing that the adversary's view on both games are close.

Claim. Game Real $\approx$ Game 0.

Proof. The only difference between Game Real and Game 0 is that, in Game 0 the system additionally maintains two tables that are completely hidden from the adversary, hence we have

$$\Pr[\text{Game Real}] = \Pr[\text{Game 0}].$$

Claim. Game 0 $\approx$ Game 1.

Proof. By definition, we note that in Game 1, the system maintains longer tables and it responds to part of the queries (type A query) by just using those tables and calling the honest interface. For the other queries (type B query), the system responds by calling the real oracles. Moreover, the items stored in those tables are always consistent with the real oracles, and in either games, the honest interfaces correspond to the real oracles, which means the response by calling the honest interfaces is identical to the one by calling real oracles. Hence the responses of type A queries by either the real oracles(Game 0) or by honest interfaces plus tables(Game 1) are identical, which refers to

$$\Pr[\text{Game 0}] = \Pr[\text{Game 1}].$$

Claim. Game 1 $\approx$ Game 2.

Proof. Recalling that the only difference between Game 1 and Game 2 occurs in case 4, where one of the elements in $(\mathsf{Z}_0, \mathsf{Z}_1)$ never appears in the previous queries, say Z_0. In Game 1, the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$, while in Game 2, the system replaces it with $\perp$. To prove the indistinguishability, we first formalize the adversary's view in Game 1. By definition, we immediately observe that,

- The system responds to $(x; \mathcal{H})$ with $\tilde{\mathcal{H}}(x)$.
- The system responds to $(x, \mathsf{label}; \mathcal{G})$ with $\tilde{\mathcal{G}}^{\mathsf{label}}(x)$.
- The system responds to $(\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$ with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$.

Hence, in the adversary's view, under the consistency conditions listed below, every bit of the response(if not $\perp$) should be random and independent. And the consistency conditions are:

- $\mathcal{H}^{\mathsf{r}}(x) = \mathsf{Trunc}_{\mathsf{n}}(\mathcal{G}^{\mathsf{rlabel}}(x))$;
- $\mathcal{G}^{\mathsf{rlabel}}(x_0 + x_1) = \mathcal{G}^{\mathsf{radd}}(\mathcal{G}^{\mathsf{rlabel}}(x_0), \mathcal{G}^{\mathsf{rlabel}}(x_1))$.
- There exists no $x \neq x'$ such that $\mathcal{G}^{\mathsf{rlabel}}(x) = \mathcal{G}^{\mathsf{rlabel}}(x')$.

We immediately observe that, if Z_0 is not a valid group element, then the view on Q_k are identical in both games, as $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1) = \perp$. Hence it suffice to prove that Z_0 is a non-valid group element except with negligible probability. In fact, the adversary might know some x^* such that $\mathcal{H}^{\mathsf{r}}(x^*) = \mathsf{Trunc}^{\mathsf{s}}_{\mathsf{n}}(\tilde{\mathcal{G}}^{\mathsf{L}}(x))$, but those bits clearly are the only information it has. To output a valid element, the adversary is obligated to guess the rest $n + \lambda$ bits, which the probability is bounded by

$$\Pr[\mathsf{Bad}] \leq \frac{2^n}{2^{n+\lambda}} = \frac{1}{2^\lambda} \leq \mathsf{negl}(\lambda).$$

Thus, we have

$$|\Pr[\text{Game 1}] - \Pr[\text{Game 2}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 2 $\approx$ Game 3.

Proof. Recalling that the only difference between Game 2 and Game 3 occurs in case 3, where both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ appear in L-table(say $(x_0, \mathsf{Z_0})$ and (x_1, Z_1)), but $x_0 + x_1 \notin \mathsf{L}$. In game 2, the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z_0}, \mathsf{Z_1})$, while in Game 3, it replaces with a random $\mathsf{Z} \in S$ such that $\mathsf{Trunc}^{\mathsf{s}}_{\mathsf{n}}(\mathsf{Z}) = \mathcal{H}^{\mathsf{r}}(x_0 + x_1)$. And after Q_k, the system implicitly sets $\mathcal{G}^{\mathsf{rlabel}}(x_0 + x_1) = \mathsf{Z}$.

Analogously, we know that, under the condition the bad events never happen, the adversary's view on $\mathcal{H}$ and $\mathcal{G}$ are random strings. Hence, the response in Game 3 are well distributed. Next, we show that, except with negligible probability, the consistency conditions also hold.

Observe that the first two equations hold trivially. And for the third one, note that Z is uniformly sampled and as long as it does not appear before Q_k (the first n bits might be known) and there exists no x such that $\tilde{\mathcal{G}}^{\mathsf{label}}(x) = \mathsf{Z}$, then the equation holds. And we can bound the union of the bad events by

$$\Pr[\mathsf{Bad}] \leq \frac{k-1}{2^{n+\lambda}} + \frac{2^n}{2^{n+\lambda}} \leq \mathsf{negl}(\lambda),$$

which refers to

$$|\Pr[\text{Game 2}] - \Pr[\text{Game 3}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 3 $\approx$ Game 4.

Proof. Applying the same analysis above(Game 2 $\approx$ Game 3), it is trivial that

$$|\Pr[\text{Game 3}] - \Pr[\text{Game 4}]| \leq |\Pr[\text{Game 2}] - \Pr[\text{Game 3}]|.$$

Claim. Game 4 $\approx$ Game 5.

Proof. Let $\mathcal{O}$ be a random oracle model, and we note that the difference between Game 4 and Game 5 is that: in Game 4 the system responds to all the queries by calling $(x; \mathcal{H})$, while in Game 5, it makes queries with form of $(x; \mathcal{O})$. In fact, as $\mathcal{H}^r(x) = \mathsf{Trunc}^{\mathsf{s}}_{\mathsf{n}}(\tilde{\mathcal{G}}^{\mathsf{label}}(x))$, and $\mathcal{G}$ is a generic group model, hence every bit of $\mathcal{H}^r(x)$ is random and independent, which means the distribution of $\mathcal{H}^r(x)$ and $\mathcal{O}^r(x)$ are identical, referring to

$$\Pr[\text{Game 4}] = \Pr[\text{Game 5}].$$

Combing together, we establish the entire proof.

B General GGM implies ROM

In this section, we consider about a more general version of GGM and show that even the encoding of GGM is not random string, we still can build an indifferentiable ROM from it. To simplify the construction, we here only give out a one-bit random oracle construction[14].
Building Blocks. Our construction consists of two building blocks:

- $\mathcal{G} = (\mathcal{G}^{\mathsf{label}}, \mathcal{G}^{\mathsf{add}})$ is a generic group model that maps $\mathbb{Z}_p$ to S, subject to distribution D;
- Ext is a (k, ϵ) strong seeded extractor.

where $\log |S| \geq 2 \log p + 2\lambda$, the min-entropy $H_\infty(D) \geq k \geq 2 \log p + \lambda$. Note that Ext is a strong seeded extractor, taking the encoding $\mathcal{G}^{\mathsf{label}}(x)$ and a seed s as inputs, and outputs a bit. Then, we can build a one-bit indifferentiable ROM $\mathcal{H}$ that maps $\{0,1\}^n \to \{0,1\}$ as follows:

$$\mathcal{H}(x) = \mathsf{Ext}(s, \mathcal{G}^{\mathsf{label}}(x)),$$

where s is a random seed as a public parameter. Observe that, for any efficient adversary that makes at most q queries, the distribution of $\mathcal{H}(x_1), \ldots, \mathcal{H}(x_q)$ is $q\epsilon$-close to uniform.

Theorem 4. *$\mathcal{H}$ is indifferentiable from a random oracle model. More precisely, there is a simulator $\mathcal{S}$ such that for all $(q_{\mathcal{H}}, q_{\mathcal{G}^{\mathsf{label}}}, q_{\mathcal{G}^{\mathsf{add}}})$-query PPT differentiator $\mathcal{D}$ with $q_{\mathcal{H}} \leq q_1$ $q_{\mathcal{G}^{\mathsf{label}}} + q_{\mathcal{G}^{\mathsf{add}}} \leq q_2$, we have*

$$\mathrm{Adv}^{\mathsf{indif}}_{\mathcal{H},\mathcal{S},\mathcal{D}} \leq \frac{q_2^2 + q_2}{2^\lambda} + (q_1 + q_2\lambda)\epsilon + \frac{q_2}{3^\lambda}.$$

The simulator makes at most $q_2\lambda$ queries to its oracles.

Proof. According to the definition of indifferentiability, the adversary has one honest interface $\mathcal{H}$ and two adversarial interfaces $\mathcal{G}^{\mathsf{label}}$ and $\mathcal{G}^{\mathsf{add}}$. Therefore we need build an efficient simulator $\mathcal{S}$ in the ideal world that can simulate the adversarial interfaces properly. We will go through a sequence of hybrid games where in each game, there exists a system that responds to all of the queries (both honest and adversarial) in a slightly different way and then we build our simulator $\mathcal{S}$ as the system in the last game. In the following, we illustrate the description of our simulator in Figure 10 and then we give the high-level intuition of our proof strategy and then complete the proof.

We immediately observe that, our simulator makes at most q queries to $\mathcal{O}$, and it keeps two tables and the size of each table is at most q, referring to $\mathcal{S}$ is efficient. In the following, we present the intuitive idea that why $\mathcal{S}$ works. Note that, $\mathcal{G}$ is a generic group model, hence the responses of a proper simulator should follow the following rules:

[14] For random oracle model, once having one-bit scheme, it's straightforwards to extend to many bits

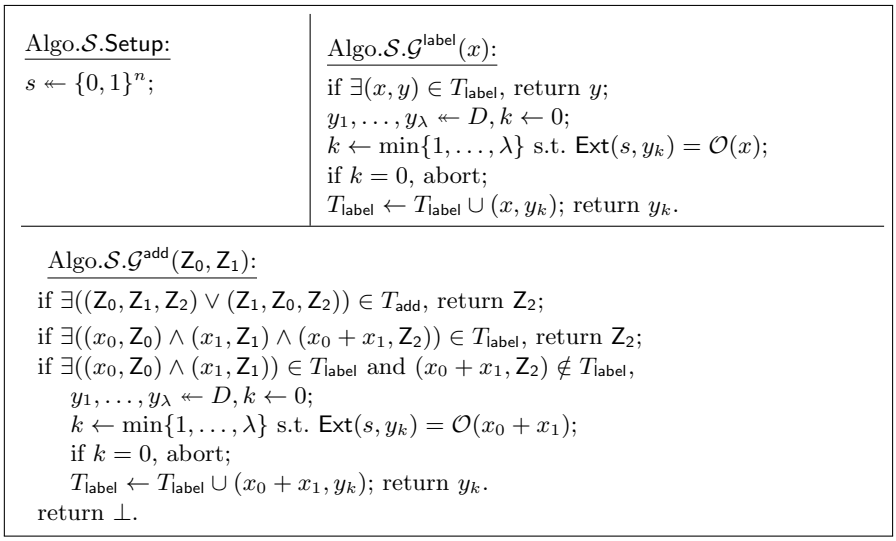

Fig. 10. Simulator for ROM in terms of two sub-simulators associated with oracle $\mathcal{O}$. These two sub-simulators share two tables $(T_{\mathsf{label}}, T_{\mathsf{add}})$ as joint state (which are initialized empty).

1. The responses of $\mathcal{G}^{\mathsf{label}}$ obey to D;
2. There do not exist $x_0 \neq x_1$ such that $\mathcal{G}^{\mathsf{label}}(x_0) = \mathcal{G}^{\mathsf{label}}(x_1)$;
3. $\mathcal{O}(x) = \mathsf{Ext}(s, \mathcal{G}^{\mathsf{label}}(x))$;
4. $\mathcal{G}^{\mathsf{label}}(x_0+x_1) = \mathcal{G}^{\mathsf{add}}(\mathcal{G}^{\mathsf{label}}(x_0), \mathcal{G}^{\mathsf{label}}(x_1))$,
5. $\forall \mathsf{Z} \notin \{\mathcal{G}^{\mathsf{label}}(x)\}_{x\in\mathbb{Z}_p}$, $\mathcal{G}^{\mathsf{add}}(\mathsf{Z},\cdot) = \perp$;
6. $\mathcal{S}$ never aborts.

Next, we show our simulator achieves those six rules.

Rule 1. By definition, it's trivial that the simulator samples the response of $\mathcal{G}^{\mathsf{label}}(x)$ by distribution D.

Rule 2. The only chance that breaks this rule is collisions occurs. As D has high min-entropy and Ext is a strong seeded extractor, this bad event is trivially bounded by $\frac{q_2^2}{2^{H_\infty(D)*(\frac{1}{2}-\epsilon)}} \leq \frac{q_2^2}{2^\lambda}$.

Rule 3. This rule holds straightforwardly under the condition that $\mathcal{S}$ does not abort.

Rule 4. Note that, when running $\mathcal{G}^{\mathsf{add}}(\mathsf{Z_0},\mathsf{Z_1})$, if $\mathsf{Z_0},\mathsf{Z_1}$ are the known valid encoding ($\mathsf{Z_0},\mathsf{Z_1}$ have already been put into T_{label}), then this equation holds for free. And the only bad event that breaks this rule is that, $\mathcal{A}$ makes the addition query before the labeling queries (in the real world, the responses is $\mathcal{G}^{\mathsf{label}}(x_0 + x_1)$, while in the ideal world, the simulator outputs $\perp$ instead). However, if this bad event occurs, then it means that the adversary needs to predict a valid encoding $\mathcal{G}^{\mathsf{label}}(x)$. Of course, the adversary could have $\mathsf{Ext}(s, \mathcal{G}^{\mathsf{label}}(x))$, but other information is lost. Thus this bad event is apparently bounded by $\frac{q_2}{2^\lambda}$.

Rule 5. By the description of $\mathcal{S}.\mathcal{G}^{\mathsf{add}}$, we know that, if and only if $\mathsf{Z}_0, \mathsf{Z}_1$ are the known valid encoding , the response is not $\perp$. Hence, if Z is an invalid encoding, $\mathcal{S}.\mathcal{G}^{\mathsf{add}}$ always outputs $\perp$.
Rule 6. By definition, we see that the only chance $\mathcal{S}$ abort is that for all y_i, $\mathsf{Ext}(s, y_i) \neq \mathcal{O}(x)$, which is trivially bounded by $\frac{q_2}{3^\lambda}$.

Next, we complete our proof. Before the description of the games, we first specify some parameters:

- There are three types of queries: $(x; \mathcal{O})$, $(y, \mathsf{label}; \mathcal{G})$ and $(\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, where $x \leftarrow \{0,1\}^n, y \leftarrow \mathbb{Z}_p$ and $\mathsf{Z}_0, \mathsf{Z}_1 \leftarrow S$.
- The adversary only makes q_1 queries to $\mathcal{H}$ and q_2 queries to $\mathcal{G}$, where $q_1, q_2 = \mathsf{poly}(\lambda)$.
- The oracles used in the real world are $\tilde{\mathcal{H}}$, $\tilde{\mathcal{G}}^{\mathsf{label}}$ and $\tilde{\mathcal{G}}^{\mathsf{add}}$;
- In each game, the system's responses are denoted as $\mathcal{O}^{\mathsf{r}}$, $\mathcal{G}^{\mathsf{rlabel}}$ and $\mathcal{G}^{\mathsf{radd}}$. For instance, $\mathcal{G}^{\mathsf{radd}}(\mathsf{Z}_0, \mathsf{Z}_1)$ denotes the system's response when adversary makes a query $Q = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$.

The hybrid games are as follows.
Game 0. This game is identical to the real game except that the system maintains two table, referring to L-table and A-table(for labeling and addition queries, respectively). Specifically, the system responds to the queries the same as in the real world, for instance, $\mathcal{O}^{\mathsf{r}}(x) = \tilde{\mathcal{O}}(x)$ and $\mathcal{G}^{\mathsf{radd}}(\mathsf{Z}_0, \mathsf{Z}_1) = \tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$. For the tables, the system maintains them as follows:

- L-table: Initially empty, consists of tuples with form of (x, Z). Once the adversary makes a label query, say $(x, \mathsf{Label}; \mathcal{G})$, which does not exist in L-table yet(no tuple such that the first element is x), the system inserts $(x, \tilde{\mathcal{G}}^{\mathsf{label}}(x))$ into L-table.
- A-table: Initially empty, consists of tuples with form of $(\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{Z})$. Once the adversary makes an addition query, say $(\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, which does not exist in A-table yet(no tuple such that the first elements are $(\mathsf{Z}_0, \mathsf{Z}_1)$ or $(\mathsf{Z}_1, \mathsf{Z}_0)$), the system inserts $(\mathsf{Z}_0, \mathsf{Z}_1, \tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1))$ into A-table. Moreover, if there exist (x_0, Z_0) and (x_1, Z_1) in the previous queries, then the system also inserts $(x_0 + x_1, \tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1))$ into L-table.

Note that at this point all the queries are answered by the real oracles and these tables are just keeping track of information related to adversary's queries (to the adversarial interfaces) and completely hidden to the adversary. Hence the view in real game is identical to the one in Game 0. Next, we illustrate an alternative way to responds to part of the queries, by using these tables and accessing to the honest interface.

For ease of exposition, we here define a relation between the query Q and the table Tab. Specifically, if Q is a labeling query, for instance $Q = (x, \mathsf{label}; \mathcal{G})$, we say $Q \in \mathsf{Tab}$ if there exists a tuple $T = (T_1, T_2) \in \mathsf{Tab}$ such that $T_1 = x$. Analogously, if Q is an addition query, for instance $Q = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, we say $Q \in \mathsf{Tab}$, if there exists a tuple $T = (T_1, T_2, T_3) \in \mathsf{Tab}$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$.

Game 1. This game is identical to Game 0, except the way of maintaining the tables and responding to the queries. Specifically,
Labeling query. Suppose $Q_k = (x, \mathsf{label}; \mathcal{G})$(the k-th query), then

- Case 1. If $Q_k \in \mathsf{L}$, which means there exists a tuple $T = (T_1, T_2) \in \mathsf{L}$ such that $T_1 = x$, then the system responds to the query with T_2.
- Case 2. Otherwise, the system responds with $\tilde{\mathcal{G}}^{\mathsf{label}}(x)$ and inserts $(x, \tilde{\mathcal{G}}^{\mathsf{label}}(x))$ into L-table.

Addition query. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, then

- Case 1. If $Q_k \in \mathsf{A}$, which means there exist a tuple $T = (T_1, T_2, T_3)$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$, then the system responds to the query with T_3.
- Case 2. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), and $T_1 + T'_1 \in \mathsf{L}$, then the system responds with the corresponding record.
- Case 3. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), but $T_1 + T'_1 \notin \mathsf{L}$, then the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$ and inserts $(T_1 + T'_1, \tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1))$ into L-table.
- Case 4. Otherwise, the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$.

Note that, in Game 1 the system keeps a longer table, and for part of the queries, the system responds to them in an alternative way, which is *only* using the tables and the honest interfaces. Moreover, in Game 1, the tuples stored in the tables correspond to the response of queries that are answered by the real oracles. Hence, in either game, the response of any query is identical, which refers to that the view in Game 1 is identical to the one in Game 0. However, the system can only answer *part of* the queries by tables and honest interfaces, and for the rest it has to call the real oracles. Thus, in the following hybrid games, we will illustrate additional alternative ways(not calling the real oracles) to respond to the rest queries, without changing the view significantly.
Game 2. This game is identical to Game 1, except for responding to the addition queries. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{A}$, which means there exist a tuple $T = (T_1, T_2, T_3)$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$, then same as in Game 1.
- Case 2. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), and $T_1 + T'_1 \in \mathsf{L}$, then same as in Game 1.
- Case 3. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T'_1, T'_2)$ such that $T_2 = \mathsf{Z}_0$ and $T'_2 = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T'_2 = \mathsf{Z}_0$), but $T_1 + T'_1 \notin \mathsf{L}$, then same as in Game 1.
- Case 4. Otherwise, the system responds with $\perp$.

Note that the only difference between Game 1 and Game 2 occurs in Case 4, where one of $(\mathsf{Z}_0, \mathsf{Z}_1)$ never appears in the labeling query, say Z_0. We immediately

observe that, if Z_0 is not a valid group element, then Game 1 and Game 2 are identical. In fact, the honest interface only gives the adversary $\mathsf{Ext}(s, \mathcal{G}_x^{\mathsf{label}})$ and due to D has high min-entropy and Ext is strong extractor, Z_0 is non-valid except with negligible probability.

Game 3. This game is identical to Game 2, except for responding to the addition queries. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{A}$, which means there exist a tuple $T = (T_1, T_2, T_3)$ such that $T_1 = \mathsf{Z}_0, T_2 = \mathsf{Z}_1$ or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0$, then same as in Game 2.
- Case 2. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T_1', T_2')$ such that $T_2 = \mathsf{Z}_0$ and $T_2' = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T_2' = \mathsf{Z}_0$), and $T_1 + T_1' \in \mathsf{L}$, then same as in Game 2.
- Case 3. If there exist two tuples $T = (T_1, T_2)$ and $T' = (T_1', T_2')$ such that $T_2 = \mathsf{Z}_0$ and $T_2' = \mathsf{Z}_1$(or $T_2 = \mathsf{Z}_1$ and $T_2' = \mathsf{Z}_0$), but $T_1 + T_1' \notin \mathsf{L}$, then the system makes a query $(T_1 + T_1'; \mathcal{O})$ and samples $y_1, \ldots, y_\lambda$ and find y_i such that $\mathsf{Ext}(s, y_i) = \mathcal{O}^{\mathsf{r}}(T_1 + T_1')$, and responds to the query with y_i. After that, the system inserts $(T_1 + T_1', \mathsf{y_i})$ into L-table.
- Case 4. Otherwise, same as in Game 2.

Note that the only difference between Game 2 and Game 3 occurs in Case 3. In this case, Z_0 and Z_1, of course, are valid group element and the system is obligated to output a proper response, which should be consistent to the honest interface. Essentially, we observe that, as $T_1 + T_1' \in \mathsf{L}$-table, the adversary can only learn $\mathsf{Ext}(s, \tilde{\mathcal{G}}^{\mathsf{label}}(T_1 + T_1'))$, and all other information is lost. Thus, it is okay to replace them with other samples conditioned on they have the same extracted value and y_i never appears before.

Game 4. This game is identical to Game 3, except for responding to the labeling queries. Suppose $Q_k = (\mathsf{x}, \mathsf{label}; \mathcal{G})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{L}$, which means there exists a tuple $T = (T_1, T_2) \in \mathsf{L}$ such that $T_1 = x$, then same as in Game 3.
- Case 2. Otherwise, the system makes a query $(T_1 + T_1'; \mathcal{O})$ and samples $y_1, \ldots, y_\lambda$ and find y_i such that $\mathsf{Ext}(s, y_i) = \mathcal{O}^{\mathsf{r}}(T_1 + T_1')$, and responds to the query with y_i. After that, the system inserts $(T_1 + T_1', \mathsf{y_i})$ into L-table.

Trivial to note that, Game 4 is close to Game 3 within the same reason above(Game 3 $\approx$ Game 2).

Game 5. In Game 4, the queries to the adversarial interfaces are answered by the tables which are maintained by the system and by making queries to honest interface. The system never makes queries directly to $\tilde{\mathcal{G}}^{\mathsf{label}}$ and $\tilde{\mathcal{G}}^{\mathsf{add}}$; these oracles are *only* used to answer the $\mathcal{O}$ queries (either generated by the adversary or by the system's response to labeling or addition queries). At this point, due to Ext is a strong extractor, it is straightforward to show that we can replace $\mathcal{O}$ with a random oracle $\mathcal{H}$, resulting in Game 5.

We note that in Game 5, the system is efficient, and it responds to the adversarial interfaces just by keeping several tables and calling the honest interfaces. Thus, we can build a simulator that responds to the honest and adversarial

queries precisely as the system does in Game 5. The result is that the view in Game 5 is identical to the ideal world and it suffices to prove that any adjacent games are indistinguishable.

Next we prove the indistinguishability between any adjacent games, by showing that the adversary's view on both games are close.

Claim. Game Real $\approx$ Game 0.

Proof. The only difference between Game Real and Game 0 is that, in Game 0 the system additionally maintains two tables that are completely hidden from the adversary, hence we have

$$\Pr[\text{Game Real}] = \Pr[\text{Game 0}].$$

Claim. Game 0 $\approx$ Game 1.

Proof. By definition, we note that in Game 1, the system maintains longer tables and it responds to part of the queries (type A query) by just using those tables and calling the honest interface. For the other queries (type B query), the system responds by calling the real oracles. Moreover, the items stored in those tables are always consistent with the real oracles, and in either games, the honest interfaces correspond to the real oracles, which means the response by calling the honest interfaces is identical to the one by calling real oracles. Hence the responses of type A queries by either the real oracles(Game 0) or by honest interfaces plus tables(Game 1) are identical, which refers to

$$\Pr[\text{Game 0}] = \Pr[\text{Game 1}].$$

Claim. Game 1 $\approx$ Game 2.

Proof. Recalling that the only difference between Game 1 and Game 2 occurs in case 4, where one of the elements in $(\mathsf{Z}_0, \mathsf{Z}_1)$ never appears in the previous queries, say Z_0. In Game 1, the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$, while in Game 2, the system replaces it with $\perp$. To prove the indistinguishability, we first formalize the adversary's view in Game 1. By definition, we immediately observe that,

- The system responds to $(x; \mathcal{O})$ with $\tilde{\mathcal{O}}(x)$.
- The system responds to $(x, \mathsf{label}; \mathcal{G})$ with $\tilde{\mathcal{G}}^{\mathsf{label}}(x)$.
- The system responds to $(\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{add}; \mathcal{G})$ with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z}_0, \mathsf{Z}_1)$.

Hence, in the adversary's view, under the consistency conditions listed below, every bit of the response(if not $\perp$) should be random and independent. And the consistency conditions are:

- $\mathcal{O}^{\mathsf{r}}(x) = \mathsf{Ext}(s, \mathcal{G}^{\mathsf{rlabel}}(x))$;
- $\mathcal{G}^{\mathsf{rlabel}}(x_0 + x_1) = \mathcal{G}^{\mathsf{radd}}(\mathcal{G}^{\mathsf{rlabel}}(x_0), \mathcal{G}^{\mathsf{rlabel}}(x_1))$.
- There exists no $x \neq x'$ such that $\mathcal{G}^{\mathsf{rlabel}}(x) = \mathcal{G}^{\mathsf{rlabel}}(x')$.

We immediately observe that, if $\mathsf{Z_0}$ is not a valid group element, then the view on Q_k are identical in both games, as $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z_0}, \mathsf{Z_1}) = \perp$. Hence it suffice to prove that Z_0 is a non-valid group element except with negligible probability. In fact, the adversary might know some x^* such that $\mathcal{O}^{\mathsf{r}}(x^*) = \mathsf{Ext}(s, \tilde{\mathcal{G}}^{\mathsf{label}}(x))$, but this is the only information that adversary gain. To outputs a valid group element, it has to recover the other information, and this is bounded by

$$\Pr[\mathsf{Bad}] \leq \frac{p}{2^{H_\infty(D)*(\frac{1}{2}-\epsilon)}} \leq \frac{1}{2^\lambda} \leq \mathrm{negl}(\lambda).$$

Thus, we have

$$|\Pr[\text{Game 1}] - \Pr[\text{Game 2}]| \leq q_2 \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 2 $\approx$ Game 3.

Proof. Recalling that the only difference between Game 2 and Game 3 occurs in case 3, where both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ appear in L-table(say $(x_0, \mathsf{Z_0})$ and (x_1, Z_1)), but $x_0 + x_1 \notin \mathsf{L}$. In game 2, the system responds with $\tilde{\mathcal{G}}^{\mathsf{add}}(\mathsf{Z_0}, \mathsf{Z_1})$, while in Game 3, it replaces with a random $y_i \twoheadleftarrow D$ such that $\mathsf{Ext}(s, y_i) = \mathcal{H}^{\mathsf{r}}(x_0 + x_1)$. And after Q_k, the system implicitly sets $\mathcal{G}^{\mathsf{rlabel}}(x_0 + x_1) = y_i$.

Analogously, we know that, under the condition the bad events never happen, the adversary's view on $\mathcal{H}$ and $\mathcal{G}$ are random strings. Hence, the response in Game 3 are well distributed. Next, we show that, except with negligible probability, the consistency conditions also hold.

Observe that the first two equations hold trivially. And for the third one, note that Z is uniformly sampled and as long as it does not appear before Q_k (the first n bits might be known) and there exists no x such that $\tilde{\mathcal{G}}^{\mathsf{label}}(x) = \mathsf{Z}$, then the equation holds. And we can bound the union of the bad events by

$$\Pr[\mathsf{Bad}] \leq \frac{k-1}{2^{n+\lambda}} + \frac{2^n}{2^{n+\lambda}} \leq \mathsf{negl}(\lambda),$$

which refers to

$$|\Pr[\text{Game 2}] - \Pr[\text{Game 3}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 3 $\approx$ Game 4.

Proof. Applying the same analysis above(Game 2 $\approx$ Game 3), it is trivial that

$$|\Pr[\text{Game 3}] - \Pr[\text{Game 4}]| \leq |\Pr[\text{Game 2}] - \Pr[\text{Game 3}]|.$$

Claim. Game 4 $\approx$ Game 5.

Proof. Let $\mathcal{H}$ be a random oracle model, and we note that the difference between Game 4 and Game 5 is that: in Game 4 the system responds to all the queries by calling $(x; \mathcal{O})$, while in Game 5, it makes queries with form of $(x; \mathcal{H})$. In fact, as $\mathcal{O}^r(x) = \mathsf{Ext}(s, \mathcal{G}^{\mathsf{label}}(x))$, where $\mathcal{G}^{\mathsf{label}}(x)$ obeys the distribution D. Moreover, the adversary only makes q_1 queries to $\mathcal{H}$ and the simulator only makes $q_2\lambda$ queries to $\mathcal{H}$, hence by hybrid argument, it's trivial the statistical distance of Game 4 and 5 is bounded by $(q_1 + q_2\lambda)\epsilon$.

Combing together, we establish the proof.

C Constructing *d*-linear Multi-linear Map from *dl*-linear Multi-linear Map

In this part, we give a more general result, which states that, for any integer d, l, we can build an indifferentiable d-linear MMaps from a dl-linear MMaps.

Construction. Let $\tilde{\mathsf{M}}$ be an ideal dl-linear maps, where the encoding of α at level $k \in [dl]$ is denoted as $\widetilde{[\![\alpha]\!]}_k$. Moreover, $\tilde{\mathsf{M}}$ allows addition and multiplication $(\widetilde{\mathsf{Add}}, \widetilde{\mathsf{Mul}})$ of the encodings, as well as zero-test($\widetilde{\mathsf{ZT}}$) of the top-level elements. Now, we build an ideal d-linear map M as follows:

- Encode algorithm: for any α and $k \in [d]$, we define $[\![\alpha]\!]_k = \widetilde{[\![\alpha]\!]}_{kl}$;
- Addition algorithm: $\mathsf{Add}([\![\alpha]\!]_k, [\![\beta]\!]_k) = \widetilde{\mathsf{Add}}(\widetilde{[\![\alpha]\!]}_{kl}, \widetilde{[\![\alpha]\!]}_{kl})$;
- Multiplication algorithm: $\mathsf{Mul}([\![\alpha]\!]_{k_1}, [\![\beta]\!]_{k_2}) = \widetilde{\mathsf{Add}}(\widetilde{[\![\alpha]\!]}_{k_1 l}, \widetilde{[\![\alpha]\!]}_{k_2 l})$;
- Zero-test: $\mathsf{ZT}([\![\alpha]\!]_d) = \widetilde{\mathsf{ZT}}([\![\alpha]\!]_{dl})$.

Correctness easily follows, and next we prove the indifferentiability for our construction.

Theorem 5 (dl-linear to d-linear). M *is indifferentiable from an ideal d-linear mulitilinear map.*

Proof. According to the definition of indifferentiability, the adversary has four honest interface $\mathsf{Encd}, \mathsf{Add}, \mathsf{Mul}, \mathsf{ZT}$ and four adversarial interfaces $\tilde{\mathsf{Encd}}, \tilde{\mathsf{Add}}, \tilde{\mathsf{Mul}}$, $\tilde{\mathsf{ZT}}$. Therefore we need build an efficient simulator $\mathcal{S}$ in the ideal world that can simulate the adversarial interfaces properly. We will go through a sequence of hybrid games where in each game, there exists a system that responds to all of the queries (both honest and adversarial) in a slightly different way and then we build our simulator $\mathcal{S}$ as the system in the last game. Before the description of the games, we first specify some parameters:

- There are four types of honest queries: $(x, k, \mathsf{label}; \mathsf{M}), (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{k}, \mathsf{add}; \mathsf{M}), (\mathsf{Z}_0, \mathsf{k}_0, \mathsf{Z}_1, \mathsf{k}_1, \mathsf{Mul}; \mathsf{M}), (\mathsf{Z}, \mathsf{ZT}, \mathsf{M})$;
- There are four type of adverserial queries: $(x, k, \mathsf{label}; \tilde{\mathsf{M}}), (\mathsf{Z}_0, \mathsf{Z}_1, \mathsf{k}, \mathsf{add}; \tilde{\mathsf{M}}), (\mathsf{Z}_0, \mathsf{k}_0, \mathsf{Z}_1, \mathsf{k}_1, \mathsf{Mul}; \tilde{\mathsf{M}}), (\mathsf{Z}, \mathsf{ZT}, \tilde{\mathsf{M}})$;
- The adversary only makes q queries to the system, where $q = \mathsf{poly}(\lambda)$.
- The oracles used in the real world are $\mathsf{M}^{\mathsf{L}}_{\mathsf{real}}, \mathsf{M}^{\mathsf{A}}_{\mathsf{real}}, \mathsf{M}^{\mathsf{M}}_{\mathsf{real}}, \mathsf{M}^{\mathsf{ZT}}_{\mathsf{real}}, \tilde{\mathsf{M}}^{\mathsf{L}}_{\mathsf{real}}, \tilde{\mathsf{M}}^{\mathsf{A}}_{\mathsf{real}}, \tilde{\mathsf{M}}^{\mathsf{M}}_{\mathsf{real}}$ and $\tilde{\mathsf{M}}^{\mathsf{ZT}}_{\mathsf{real}}$;
- In each game, the system's responses are denoted as $\mathsf{M}^{\mathsf{rL}}, \mathsf{M}^{\mathsf{rA}}, \mathsf{M}^{\mathsf{rM}}, \mathsf{M}^{\mathsf{rZT}}$ and $\tilde{\mathsf{M}}^{\mathsf{rL}}, \tilde{\mathsf{M}}^{\mathsf{rA}}, \tilde{\mathsf{M}}^{\mathsf{rM}}, \tilde{\mathsf{M}}^{\mathsf{rZT}}$.

The hybrid games are as follows.

Game 0. This game is identical to the real game except that the system maintains $3dl$ table, referring to L_{i}-table and A_{i}-table(for encoding, addition and multiplication queries of each individual level , respectively). Specifically, the system responds to the queries the same as in the real world, for instance, $\mathsf{M}^{\mathsf{rL}}(x) = \mathsf{M}^{\mathsf{L}}_{\mathsf{real}}(x)$ and $\tilde{\mathsf{M}}^{\mathsf{rA}}(\mathsf{Z}_0, \mathsf{Z}_1, S) = \tilde{\mathsf{M}}^{\mathsf{A}}_{\mathsf{real}}(\mathsf{Z}_0, \mathsf{Z}_1, S) =$. For the tables, the system maintains them as follows:

- $\mathsf{L_i}$-table: Initially empty, consists of tuples with form of (x, i, Z). Once the adversary makes a label query for level $i \in [dl]$, say $(x, i\mathsf{Label}; \tilde{\mathsf{M}})$, which does not exist in $\mathsf{L_i}$-table yet(no tuple such that the first element is x and the second element is i), the system inserts $(x, i, \tilde{\mathsf{M}}^{\mathsf{L}}_{\mathsf{real}}(x)$ into $\mathsf{L_i}$-table.
- $\mathsf{A_i}$-table: Initially empty, consists of tuples with form of $(\mathsf{Z_0}, \mathsf{Z_1}, \mathsf{i}, \mathsf{Z})$. Once the adversary makes an addition query, say $(\mathsf{Z_0}, \mathsf{Z_1}, \mathsf{i}, \mathsf{add}; \tilde{\mathsf{M}})$, which does not exist in $\mathsf{A_i}$-table yet(no tuple such that the first two elements are $(\mathsf{Z_0}, \mathsf{Z_1})$ or $(\mathsf{Z_1}, \mathsf{Z_0})$ for level i), the system inserts $(\mathsf{Z_0}, \mathsf{Z_1}, i, \tilde{\mathsf{M}}^{\mathsf{A}}_{\mathsf{real}}(\mathsf{Z_0}, \mathsf{Z_1})$ into $\mathsf{A_i}$-table. Moreover, if there exist $(x_0, i, \mathsf{Z_0})$ and $(x_1, i, \mathsf{Z_1})$ in the previous queries, then the system also inserts $(x_0 + x_1, i, \tilde{\mathsf{M}}^{\mathsf{A}}_{\mathsf{real}}(\mathsf{Z_0}, \mathsf{Z_1}, \mathsf{i}))$ into $\mathsf{L_i}$-table.
- $\mathsf{M_i}$-table: Initially empty, consists of tuples with form of $(\mathsf{Z_0}, \mathsf{i_0}, \mathsf{Z_1}, \mathsf{i_1}, \mathsf{Z})$, where $i_0 + i_1 = i$. Once the adversary makes an addition query, say $(\mathsf{Z_0}, \mathsf{i_0}, \mathsf{Z_1}, \mathsf{i_1}, \mathsf{mul}; \mathsf{M})$, which does not exist in $\mathsf{M_i}$-table yet, then the system inserts $(\mathsf{Z_0}, \mathsf{i_0}, \mathsf{Z_1}, \mathsf{i_1}, \tilde{\mathsf{M}}^{\mathsf{M}}_{\mathsf{real}}(\mathsf{Z_0}, \mathsf{i_0}, \mathsf{Z_1}, \mathsf{i_1}))$ into $\mathsf{M_i}$-table. Moreover, if $(x_0, i_0, \mathsf{Z_0})$ and $(x_1, i_1, \mathsf{Z_1})$ appear in the previous queries, then the system also inserts $(x_0 + x_1, i_0 + i_1, \tilde{\mathsf{M}}^{\mathsf{M}}_{\mathsf{real}}(\mathsf{Z_0}, \mathsf{Z_1}))$ into $\mathsf{L_i}$-table.

Note that at this point all the queries are answered by the real oracles and these tables are just keeping track of information related to adversary's queries (to the adversarial interfaces) and completely hidden to the adversary. Hence the view in real game is identical to the one in Game 0. Next, we illustrate an alternative way to responds to part of the queries, by using these tables and accessing to the honest interface.

To ease of exposition, we here introduce a new notion, called "*upgradable*". We say an group element Z is upgradable if: 1) $\mathsf{Z} \in \mathsf{L_i} \cup \mathsf{A_i} \cup \mathsf{M_i} (i = al + b)$ where $a < d$ and $a \geq b > 0$; 2) we can represent $\mathsf{Z} = \sum [\![\alpha_1]\!]_{k_1} \cdot \ldots \cdot [\![\alpha_s]\!]_{k_s}$ where $[\![\alpha_\ell]\!]_{k_\ell}$ is either a valid group element in M or in L_{k_ℓ}-table. Thus, if Z is upgradable, it's trivial to upgrade Z to a valid element $\mathsf{up}(\mathsf{Z})$ on level $(a+1)l$, such that Z and $\mathsf{up}(\mathsf{Z})$ share the same plaintext and the upgrade algorithm only needs access to the honest interfaces.

Game 1. This game is identical to Game 0, except the way of maintaining the tables and responding to the queries. Specifically,

Zero-test query. Suppose $Q_k = (\mathsf{Z}, \mathsf{ZT}; \tilde{\mathsf{M}})$, then the system just makes a query $(\mathsf{Z}, \mathsf{ZT}, \mathsf{M})$ and outputs the corresponding response.

Labeling query. Suppose $Q_k = (x, i, \mathsf{label}, \tilde{\mathsf{M}})$, then

- Case 1. If $Q_k \in \mathsf{L_i}$, which means there exists a tuple $T = (T_1, T_2, T_3) \in \mathsf{L}$ such that $T_1 = x$ and $T_2 = i$, then the system responds to the query with T_3.
- Case 2. If l divides i, then the system makes a query $(x, i/l, \mathsf{label}; \mathsf{M})$ and outputs the corresponding response. Then it inserts $(x, i, \mathsf{M}^{\mathsf{rL}}(\mathsf{x}, \mathsf{i/l}))$ into $\mathsf{L_i}$-table.
- Case 3. Else, we have that $i = al + b$ where $0 < b < a < d$. Then if there exists a upgradable element $\mathsf{Z} \in \mathsf{A_i} \cup \mathsf{M_i}$ such that $\mathsf{M}^{\mathsf{rL}}(x, a+1) = \mathsf{up}(\mathsf{Z})$, then the system responds to the query with Z and inserts (x, i, Z) into $\mathsf{L_i}$-table.

– Case 4. Otherwise, it responds with $\tilde{\mathsf{M}}^{\mathsf{rL}}_{\mathsf{real}}(x,i)$ and inserts $(x,i,\tilde{\mathsf{M}}^{\mathsf{rL}}_{\mathsf{real}}(x,i))$ into $\mathsf{L_i}$-table.

Addition query. Suppose $Q_k = (\mathsf{Z_0},\mathsf{Z_1},i,\mathsf{add};\tilde{\mathsf{M}})$, then

– Case 1. If $Q_k \in \mathsf{A_i}$, which means there exists a tuple $T=(T_1,T_2,T_3,T_4) \in \mathsf{A_i}$ such that $T_1 = \mathsf{Z_0}, \mathsf{T_2} = \mathsf{Z_1}$ and $T_3 = i$(or $T_1 = \mathsf{Z_1}, T_2 = \mathsf{Z_0}, T_3 = i$), then the system responds to the query with T_4.
– Case 2. If l divides i, then the system makes a query $(\mathsf{Z_0},\mathsf{Z_i},i/l,\mathsf{add};\mathsf{M})$ and outputs the corresponding response.
– Case 3. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable and there exists Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rL}}(\mathsf{up}(\mathsf{Z_0}),\mathsf{up}(\mathsf{Z_1}),(a+1))$, then the system responds to the query with Z and inserts $(\mathsf{Z_0},\mathsf{Z_1},i,\mathsf{Z})$ into $\mathsf{A_i}$-table.
– Case 4. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable but there is no upgradable Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rL}}_{\mathsf{real}}(\mathsf{up}(\mathsf{Z_0}),\mathsf{up}(\mathsf{Z_1}))$, then the system responds to the query with $\tilde{\mathsf{M}}^{\mathsf{rA}}_{\mathsf{real}}(\mathsf{Z_0},\mathsf{Z_1},i)$.
– Case 5. Otherwise, it responds the query with $\tilde{\mathsf{M}}^{\mathsf{rA}}_{\mathsf{real}}(\mathsf{Z_0},\mathsf{Z_1},i)$ and inserts $(\mathsf{Z_0},\mathsf{Z_1},i,\tilde{\mathsf{M}}^{\mathsf{rA}}_{\mathsf{real}}(\mathsf{Z_0},\mathsf{Z_1},i))$ into $\mathsf{A_i}$-table.

Multiplication query. Suppose $Q_k = (\mathsf{Z_0},i_0,\mathsf{Z_1},i_1,\mathsf{mul};\tilde{\mathsf{M}})$, then

– Case 1. If $Q_k \in \mathsf{M}_{\mathsf{i_0+i_1}}$-table, then the system responds with the corresponding item.
– Case 2. If l divides both i_0 and i_1, then the system makes a query $(\mathsf{Z_0},i_0/l,\mathsf{Z_1},i_1/l,\mathsf{mul};\mathsf{M})$, and outputs the corresponding response.
– Case 3. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable, and there is a $\mathsf{Z} \in \mathsf{L}_{\mathsf{i_0+i_1}} \cup \mathsf{A}_{\mathsf{i_0+i_1}}$ such that $\mathsf{up}(\mathsf{Z_0}\cdot\mathsf{Z_1}) = \mathsf{up}(\mathsf{Z})$, then the system responds to the query with Z.
– Case 4. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable, but there is no $\mathsf{Z} \in \mathsf{L}_{\mathsf{i_0+i_1}} \cup \mathsf{A}_{\mathsf{i_0+i_1}}$ such that $\mathsf{up}(\mathsf{Z_0}\cdot\mathsf{Z_1}) = \mathsf{up}(\mathsf{Z})$, then the system responds to the query with $\tilde{\mathsf{M}}^{\mathsf{rM}}_{\mathsf{real}}(\mathsf{Z_0},i_0,\mathsf{Z_1},i_1)$.
– Case 5. Otherwise, the system responds to the query with $\tilde{\mathsf{M}}^{\mathsf{rM}}_{\mathsf{real}}(\mathsf{Z_0},i_0,\mathsf{Z_1},i_1)$.

Note that, in Game 1 the system keeps a longer table, and for part of the queries, the system responds to them in an alternative way, which is *only* using the tables and the honest interfaces. Moreover, in Game 1, the tuples stored in the tables correspond to the response of queries that are answered by the real oracles. Hence, in either game, the response of any query is identical, which refers to that the view in Game 1 is identical to the one in Game 0. However, the system can only answer *part of* the queries by tables and honest interfaces, and for the rest it has to call the real oracles. Thus, in the following hybrid games, we will illustrate additional alternative ways(not calling the real oracles) to respond to the rest queries, without changing the view significantly.

Game 2. This game is identical to Game 1, except for responding to the addition queries. Suppose $Q_k = (\mathsf{Z_0},\mathsf{Z_1},i,\mathsf{add};\tilde{\mathsf{M}})$, then the system responds:

– Case 1. If $Q_k \in \mathsf{A_i}$, then same as in Game 1.

– Case 2. If l divides i, thensame as in Game 1.
– Case 3. If both Z_0 and Z_1 are upgradable and there exists Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rL}}(\mathsf{up}(\mathsf{Z}_0), \mathsf{up}(\mathsf{Z}_1), (a+1))$, then same as in Game 1.
– Case 4. If both Z_0 and Z_1 are upgradable but there is no upgradable Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rL}}_{\mathsf{real}}(\mathsf{up}(\mathsf{Z}_0), \mathsf{up}(\mathsf{Z}_1))$, then same as in Game 1.
– Case 5. Otherwise, it responds the query with $\perp$.

Note that the only difference between Game 1 and Game 2 occurs in Case 5, where at least one of $(\mathsf{Z}_0, \mathsf{Z}_1)$ is not upgradable, say Z_0. We immediately observe that, if Z_0 is not a valid group element, then Game 1 and Game 2 are identical. In fact, the honest interfaces only give the adversary the encoding at level $al, a \in [d]$, and if Z_0 is not upgradable, then the adversary has to guess an encoding correctly at other levels, which is of course bounded by negligible probability.

Game 3. This game is identical to Game 2, except for responding to the multiplication queries. Suppose $Q_k = (\mathsf{Z}_0, i_0, \mathsf{Z}_1, i_1, \mathsf{mul}; \tilde{\mathsf{M}})$, then the system responds:

– Case 1. If $Q_k \in \mathsf{M}_{\mathsf{i}_0+\mathsf{i}_1}$-table, then same as in Game 2.
– Case 2. If l divides both i_0 and i_1, then same as in Game 2.
– Case 3. If both Z_0 and Z_1 are upgradable, and there is a $\mathsf{Z} \in \mathsf{L}_{\mathsf{i}_0+\mathsf{i}_1} \cup \mathsf{A}_{\mathsf{i}_0+\mathsf{i}_1}$ such that $\mathsf{up}(\mathsf{Z}_0 \cdot \mathsf{Z}_1) = \mathsf{up}(\mathsf{Z})$, then same as in Game 2.
– Case 4. If both Z_0 and Z_1 are upgradable, but there is no $\mathsf{Z} \in \mathsf{L}_{\mathsf{i}_0+\mathsf{i}_1} \cup \mathsf{A}_{\mathsf{i}_0+\mathsf{i}_1}$ such that $\mathsf{up}(\mathsf{Z}_0 \cdot \mathsf{Z}_1) = \mathsf{up}(\mathsf{Z})$, then same as in Game 2.
– Case 5. Otherwise, the system responds to the query with $\perp$.

Same as above, we see that if one of $(\mathsf{Z}_0, \mathsf{Z}_1)$(say Z_0) is not upgradable, then with overwhelming probability, Z_0 is not a valid group element.

Game 4. This game is identical to Game 3, except for responding to the addition queries. Suppose $Q_k = (\mathsf{Z}_0, \mathsf{Z}_1, i, \mathsf{add}; \tilde{\mathsf{M}})$, then the system responds:

– Case 1. If $Q_k \in \mathsf{A}_{\mathsf{i}}$, which means there exists a tuple $T = (T_1, T_2, T_3, T_4) \in \mathsf{A}_{\mathsf{i}}$ such that $T_1 = \mathsf{Z}_0, \mathsf{T}_2 = \mathsf{Z}_1$ and $T_3 = i$(or $T_1 = \mathsf{Z}_1, T_2 = \mathsf{Z}_0, T_3 = i$), then same as in Game 3.
– Case 2. If l divides i, then same as in Game 3.
– Case 3. If both Z_0 and Z_1 are upgradable and there exists Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rA}}(\mathsf{up}(\mathsf{Z}_0), \mathsf{up}(\mathsf{Z}_1), (a+1))$, then same as in Game 3.
– Case 4. If both Z_0 and Z_1 are upgradable but there is no upgradable Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rL}}_{\mathsf{real}}(\mathsf{up}(\mathsf{Z}_0), \mathsf{up}(\mathsf{Z}_1))$, then the system responds to the query with a random string.
– Case 5. Otherwise, it responds the query with $\perp$.

Note that the only difference between Game 3 and Game 4 occurs in Case 4. In this case, Z_0 and Z_1, of course, are valid group element and the system is obligated to output a consistent response. While, as there is no Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rL}}_{\mathsf{real}}(\mathsf{up}(\mathsf{Z}_0), \mathsf{up}(\mathsf{Z}_1))$, which means the encoding is independent of the adversary's view. Thus it is okay to replace them with random strings, as long as those strings never appear before Q_k, which can only happen with negligible probability.

Game 5. This game is identical to Game 4, except for responding to the multiplication queries. Suppose $Q_k = (\mathsf{Z_0}, i_0, \mathsf{Z_1}, i_1, \mathsf{mul}; \tilde{\mathsf{M}})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{M}_{\mathsf{i}_0+\mathsf{i}_1}$-table, then same as in Game 4.
- Case 2. If l divides both i_0 and i_1, then same as in Game 4.
- Case 3. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable, and there is a $\mathsf{Z} \in \mathsf{L}_{\mathsf{i}_0+\mathsf{i}_1} \cup \mathsf{A}_{\mathsf{i}_0+\mathsf{i}_1}$ such that $\mathsf{up}(\mathsf{Z_0} \cdot \mathsf{Z_1}) = \mathsf{up}(\mathsf{Z})$, then same as in Game 4.
- Case 4. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable, but there is no $\mathsf{Z} \in \mathsf{L}_{\mathsf{i}_0+\mathsf{i}_1} \cup \mathsf{A}_{\mathsf{i}_0+\mathsf{i}_1}$ such that $\mathsf{up}(\mathsf{Z_0} \cdot \mathsf{Z_1}) = \mathsf{up}(\mathsf{Z})$, then the system responds to the query with a random string.
- Case 5. Otherwise, the system responds to the query with $\perp$.

Trivial to note that, Game 5 is close to Game 4 within the same reason above(Game 4 $\approx$ Game 3).

Game 6. This game is identical to Game 5, except for responding to the labeling queries. Suppose $Q_k = (x, i, \mathsf{label}; \tilde{\mathsf{M}})$, then the system responds:

- Case 1. If $Q_k \in \mathsf{L_i}$, then same as in Game 5.
- Case 2. If l divides i, then same as in Game 5.
- Case 3. If there exists a upgradable element $\mathsf{Z} \in \mathsf{A_i} \cup \mathsf{M_i}$ such that $\mathsf{M}^{\mathsf{rL}}(x, a+1) = \mathsf{up}(\mathsf{Z})$, then same as in Game 5.
- Case 4. Otherwise, it responds with a random string.

The argument that Game 6 is close to Game 5 easily follows as above(Game 4 $\approx$ Game 3).

Game 7. In Game 6, the queries to the adversarial interfaces are answered by the tables which are maintained by the system and by making queries to honest interface. The system never makes queries directly to the adversarial interfaces; these oracles are *only* used to answer the honest queries (either generated by the adversary or by the system's response to labeling or addition queries). At this point, it is straightforward to show that we can replace M with an ideal d-linear MMaps, resulting in Game 7.

We note that in Game 7, the system is efficient, and it responds to the adversarial interfaces just by keeping several tables and calling the honest interfaces. Thus, we can build a simulator that responds to the honest and adversarial queries precisely as the system does in Game 7. The result is that the view in Game 7 is identical to the ideal world and it suffices to prove that any adjacent games are indistinguishable.

In the following, we first give the full description of the simulator and then complete the proof by showing that every pair of adjacent games are indistinguishable.

Simulator in Ideal Game. Let MM be an ideal d-linear MMaps, and we define the simulator $\mathcal{S}$ works as follows. Like the system in Game 7, the simulator also maintains $3dl$ tables. Concretely:

- $\mathsf{L_i}$-table: Initially empty, consists of tuples with form of (x, i, Z).
- $\mathsf{A_i}$-table: Initially empty, consists of tuples with form of $(\mathsf{Z_0}, \mathsf{Z_1}, \mathsf{i}, \mathsf{Z})$.

– $\mathsf{M_i}$-table: Initially empty, consists of tuples with form of $(\mathsf{Z_0}, \mathsf{i_0}, \mathsf{Z_1}, \mathsf{i_1}, \mathsf{Z})$, where $i_0 + i_1 = i$.

By definition, $\mathcal{S}$ has access to the honest interfaces MM. And for the adversarial queries, $\mathcal{S}$ works as the system in Game 7, by just using the tables and calling the honest interfaces.

Zero-test query. Suppose $Q_k = (\mathsf{Z}, \mathsf{ZT}; \tilde{\mathsf{M}})$, then $\mathcal{S}$ just makes a query $(\mathsf{Z}, \mathsf{ZT}, \mathsf{MM})$ and outputs the corresponding response.

Labeling query. Suppose $Q_k = (x, i, \mathsf{label}, \tilde{\mathsf{M}})$, then

– Case 1. If $Q_k \in \mathsf{L_i}$, which means there exists a tuple $T = (T_1, T_2, T_3) \in \mathsf{L}$ such that $T_1 = x$ and $T_2 = i$, then $\mathcal{S}$ responds to the query with T_3.
– Case 2. If l divides i, then the simulator makes a query $(x, i/l, \mathsf{label}; \mathsf{M})$ and outputs the corresponding response. Then it inserts $(x, i, \mathsf{MM^{rL}(x, i/l)})$ into $\mathsf{L_i}$-table.
– Case 3. Else if there exists a upgradable element $\mathsf{Z} \in \mathsf{A_i} \cup \mathsf{M_i}$ such that $\mathsf{MM^{rL}}(x, a+1) = \mathsf{up}(\mathsf{Z})$, then the simulator responds to the query with Z and inserts (x, i, Z) into $\mathsf{L_i}$-table.
– Case 4. Otherwise, $\mathcal{S}$ responds with a random string.

Addition query. Suppose $Q_k = (\mathsf{Z_0}, \mathsf{Z_1}, i, \mathsf{add}; \tilde{\mathsf{M}})$, then

– Case 1. If $Q_k \in \mathsf{A_i}$, which means there exists a tuple $T = (T_1, T_2, T_3, T_4) \in \mathsf{A_i}$ such that $T_1 = \mathsf{Z_0}, \mathsf{T}_2 = \mathsf{Z_1}$ and $T_3 = i$(or $T_1 = \mathsf{Z_1}, T_2 = \mathsf{Z_0}, T_3 = i$), then $\mathcal{S}$ responds to the query with T_4.
– Case 2. If l divides i, then the simulator makes a query $(\mathsf{Z_0}, \mathsf{Z_i}, i/l, \mathsf{add}; \mathsf{MM})$ and outputs the corresponding response.
– Case 3. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable and there exists Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{MM^{rL}}(\mathsf{up}(\mathsf{Z_0}), \mathsf{up}(\mathsf{Z_1}), (a+1))$, then the simulator responds to the query with Z and inserts $(\mathsf{Z_0}, \mathsf{Z_1}, i, \mathsf{Z})$ into $\mathsf{A_i}$-table.
– Case 4. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable but there is no upgradable Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{MM^{rL}}(\mathsf{up}(\mathsf{Z_0}), \mathsf{up}(\mathsf{Z_1}))$, then the simulator responds to the query with a random string.
– Case 5. Otherwise, it responds the query with $\perp$.

Multiplication query. Suppose $Q_k = (\mathsf{Z_0}, i_0, \mathsf{Z_1}, i_1, \mathsf{mul}; \tilde{\mathsf{M}})$, then

– Case 1. If $Q_k \in \mathsf{M_{i_0+i_1}}$-table, then the simulator responds with the corresponding item.
– Case 2. If l divides both i_0 and i_1, then the simulator makes a query $(\mathsf{Z_0}, i_0/l, \mathsf{Z_1},$
$i_1/l, \mathsf{mul}; \mathsf{MM})$, and outputs the corresponding response.
– Case 3. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable, and there is a $\mathsf{Z} \in \mathsf{L_{i_0+i_1}} \cup \mathsf{A_{i_0+i_1}}$ such that $\mathsf{up}(\mathsf{Z_0} \cdot \mathsf{Z_1}) = \mathsf{up}(\mathsf{Z})$, then the simulator responds to the query with Z.
– Case 4. If both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable, but there is no $\mathsf{Z} \in \mathsf{L_{i_0+i_1}} \cup \mathsf{A_{i_0+i_1}}$ such that $\mathsf{up}(\mathsf{Z_0} \cdot \mathsf{Z_1}) = \mathsf{up}(\mathsf{Z})$, then the system responds to the query with a random string.

– Case 5. Otherwise, the system responds to the query with $\perp$.

Next we prove the indistinguishability between any adjacent games, by showing that the adversary's view on both games are close.

Claim. Game Real $\approx$ Game 0.

Proof. The only difference between Game Real and Game 0 is that, in Game 0 the system additionally maintains two tables that are completely hidden from the adversary, hence we have

$$\Pr[\text{Game Real}] = \Pr[\text{Game 0}].$$

Claim. Game 0 $\approx$ Game 1.

Proof. By definition, we note that in Game 1, the system maintains longer tables and it responds to part of the queries (type A query) by just using those tables and calling the honest interface. For the other queries (type B query), the system responds by calling the real oracles. Moreover, the items stored in those tables are always consistent with the real oracles, and in either games, the honest interfaces correspond to the real oracles, which means the response by calling the honest interfaces is identical to the one by calling real oracles. Hence the responses of type A queries by either the real oracles(Game 0) or by honest interfaces plus tables(Game 1) are identical, which refers to

$$\Pr[\text{Game 0}] = \Pr[\text{Game 1}].$$

Claim. Game 1 $\approx$ Game 2.

Proof. Recalling that the only difference between Game 1 and Game 2 occurs in case 5, where one of the elements in $(\mathsf{Z}_0, \mathsf{Z}_1)$, say Z_0, is not upgradable. In Game 1, the system responds with $\tilde{\mathsf{M}}^{\mathsf{L}}_{\mathsf{real}}(\mathsf{Z}_0, \mathsf{Z}_1, i)$, while in Game 2, the system replaces it with $\perp$. To prove the indistinguishability, we first formalize the adversary's view in Game 1. By definition, we immediately observe that,

– The system responds to $(x, i, \mathsf{label}; \mathsf{M})$ with $\tilde{\mathsf{M}}^{\mathsf{L}}_{\mathsf{real}}(x, il)$.
– The system responds to $(\mathsf{Z}_0, \mathsf{Z}_1, i, \mathsf{add}; ; \mathsf{M})$ with $\tilde{\mathsf{M}}^{\mathsf{A}}_{\mathsf{real}}(\mathsf{Z}_0, \mathsf{Z}_1, il)$.
– The system responds to $(\mathsf{Z}_0, i_0, \mathsf{Z}_1, i_1, \mathsf{mul}; \mathsf{M})$ with $\tilde{\mathsf{M}}^{\mathsf{M}}_{\mathsf{real}}(\mathsf{Z}_0, i_0 l, \mathsf{Z}_1, i_1 l)$.
– The system responds to $(x, i, \mathsf{label}; \tilde{\mathsf{M}})$ with $\tilde{\mathsf{M}}^{\mathsf{L}}_{\mathsf{real}}(x, i)$.
– The system responds to $(\mathsf{Z}_0, \mathsf{Z}_1, i, \mathsf{add}; ; \tilde{\mathsf{M}})$ with $\tilde{\mathsf{M}}^{\mathsf{A}}_{\mathsf{real}}(\mathsf{Z}_0, \mathsf{Z}_1, i)$.
– The system responds to $(\mathsf{Z}_0, i_0, \mathsf{Z}_1, i_1, \mathsf{mul}; \tilde{\mathsf{M}})$ with $\tilde{\mathsf{M}}^{\mathsf{M}}_{\mathsf{real}}(\mathsf{Z}_0, i_0, \mathsf{Z}_1, i_1)$.

Hence, in the adversary's view, under the consistency conditions listed below, every bit of the response(if not $\perp$) should be random and independent. And the consistency conditions are:

– $\mathsf{M}^{\mathsf{rL}}(x, i) = \tilde{\mathsf{M}}^{\mathsf{rL}}(x, il)$;
– $\mathsf{M}^{\mathsf{rA}}(\mathsf{M}^{\mathsf{rL}}(x_0, i), \mathsf{M}^{\mathsf{rL}}(x_1, i)) = \mathsf{M}^{\mathsf{rL}}(x_0 + x_1, i)$;
– $\mathsf{M}^{\mathsf{rM}}(\mathsf{M}^{\mathsf{rL}}(x_0, i_0), \mathsf{M}^{\mathsf{rL}}(x_1, i_1)) = \mathsf{M}^{\mathsf{rL}}(x_0 x_1, i_0 + i_1)$;
– $\tilde{\mathsf{M}}^{\mathsf{rA}}(\tilde{\mathsf{M}}^{\mathsf{rL}}(x_0, i), \tilde{\mathsf{M}}^{\mathsf{rL}}(x_1, i)) = \tilde{\mathsf{M}}^{\mathsf{rL}}(x_0 + x_1, i)$;

– $\tilde{\mathsf{M}}^{\mathsf{rM}}(\tilde{\mathsf{M}}^{\mathsf{rL}}(x_0, i_0), \tilde{\mathsf{M}}^{\mathsf{rL}}(x_1, i_1)) = \tilde{\mathsf{M}}^{\mathsf{rL}}(x_0 x_1, i_0 + i_1)$;
– There exist no $(x_0, i_0) \neq (x_1, i_1)$ such that $\tilde{\mathsf{M}}^{\mathsf{rL}}(x_0, i_0) = \tilde{\mathsf{M}}^{\mathsf{rL}}(x_1, i_1)$.

We immediately observe that, if $\mathsf{Z_0}$ is not a valid group element, then the view on Q_k are identical in both games, as $\tilde{\mathsf{M}}^{\mathsf{L}}_{\mathsf{real}}(\mathsf{Z_0}, \mathsf{Z_1}, i) = \perp$. Hence it suffice to prove that Z_0 is a non-valid group element except with negligible probability. In fact, the adversary only has knowledge of valid elements at level cl, where $c \in [d]$, and is independent of the encodings at other levels(except making a labeling query). And in case 5, we have $i = al + b(b > 0)$, which refers to that, the adversary is obligated to output at least one valid encoding at other levels(not divided by l). Hence,

$$|\Pr[\text{Game 1}] - \Pr[\text{Game 2}]| \leq \mathsf{negl}(\lambda).$$

Claim. Game 2 $\approx$ Game 3.

Proof. Due to definition, it's apparent that

$$|\Pr[\text{Game 3}] - \Pr[\text{Game 2}]| = |\Pr[\text{Game 2}] - \Pr[\text{Game 1}]|.$$

Claim. Game 3 $\approx$ Game 4.

Proof. Recalling that the only difference between Game 3 and Game 4 occurs in case 4, where both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are both upgradable, but there is no Z such that $\mathsf{up}(\mathsf{Z}) = \mathsf{M}^{\mathsf{rA}}(\mathsf{up}(\mathsf{Z_0}), \mathsf{up}(\mathsf{Z_1}), (\mathsf{a}+1))$. In game 3, the system responds with $\tilde{\mathsf{M}}^{\mathsf{rA}}_{\mathsf{real}}(\mathsf{Z_0}, \mathsf{Z_1}, i)$, while in game 4, the system replaces it with a random string str. And after Q_k, the system implicitly set $\tilde{\mathsf{M}}^{\mathsf{rA}}(\mathsf{Z_0}, \mathsf{Z_1}, i) = \mathsf{str}$.

Analogously, we know that, under the condition the bad events never happen, the adversary's view on the response of M and $\tilde{\mathsf{M}}$ are random strings. Hence, the response in Game 4 are well distributed. Next, we show that, except with negligible probability, the consistency conditions also hold.

Observe that, under the condition that both $\mathsf{Z_0}$ and $\mathsf{Z_1}$ are upgradable, the first five equations hold trivially. And for the last one, note that str is uniformly sampled and as long as it does not appear before Q_k and there exists no (x, i) such that $\tilde{\mathsf{M}}^{\mathsf{rA}}_{\mathsf{real}}(\mathsf{Z_0}, \mathsf{Z_1}, i) = \mathsf{str}$, then the equation holds. Thus, which refers to

$$|\Pr[\text{Game 3}] - \Pr[\text{Game 4}]| \leq \mathsf{negl}(\lambda).$$

Claim. Game 4 $\approx$ Game 5. Applying the same analysis above(Game 3 $\approx$ Game 2), it is trivial that

$$|\Pr[\text{Game 5}] - \Pr[\text{Game 4}]| = |\Pr[\text{Game 4}] - \Pr[\text{Game 3}]|.$$

Claim. Game 5 $\approx$ Game 6. Due to definition, it's apparent that

$$|\Pr[\text{Game 6}] - \Pr[\text{Game 5}]| = |\Pr[\text{Game 4}] - \Pr[\text{Game 3}]|.$$

Claim. Game 6 $\approx$ Game 7.

Proof. Let MM be an ideal d-linear MMap, and we note that the difference between Game 6 and Game 7 is: in Game 6 the system responds to all the queries by the tables and access to M, while in Game 7, it would make queries to MM. Hence, it suffices to show that the distribution of M and MM are close. In fact, by definition, $[\![\alpha]\!]_k = \widetilde{[\![\alpha]\!]}_{kl}$ and $\tilde{\mathsf{M}}$ is an ideal dl-linear MMaps, thus the distribution of M and MM are identical, referring to

$$\Pr[\text{Game 4}] = \Pr[\text{Game 5}].$$

Combing together, we establish the entire proof.