On the Impossibility of Robust Combiners for Cryptographic Groups

Cong Zhang[†] Wenli Wang[†] Taiyu Wang[†] Hong-Sheng Zhou[‡]
Pengfei Chen[†] Zhihong Jia[†] Jian Liu[†] Jinfei Liu[†] Moti Yung [§]
Kui Ren[†]

August 20, 2026

Abstract

A (k, n)-robust combiner for a primitive $\mathcal{P}$ combines n candidate instantiations of $\mathcal{P}$ into a single scheme that remains secure as long as at least k of them remain secure. Robust combiners have been extensively studied for primitives such as hash functions, public-key encryption, and oblivious transfer, but much less is known in the setting of cryptographic groups. In this work, we initiate the study of robust combiners for cryptographic groups in Maurer's generic group model (GGM), where algorithms access group elements only through abstract algebraic operations.

We ask whether one can combine n candidate groups into a single group that remains secure provided that at least k of the underlying groups remain secure. A natural baseline is the direct-product construction, which preserves search hardness but fails for decisional assumptions and incurs substantial representation overhead. We show that these limitations are in fact inherent.

Our first result is a complete impossibility for the decisional Diffie–Hellman assumption: for every polynomially bounded n and k with $k < n$, there is no generic (k, n)-robust combiner for cryptographic groups that preserves DDH security. Our second result gives a tight threshold for search assumptions in the regime where n and k are fixed constants. For the discrete logarithm problem, robust generic combination is possible when the combined group order is large enough to encode the secrets of $n - k + 1$ components; concretely, if $\log N \geq (n - k + 1)\lambda$, where the component groups have distinct λ-bit prime order, then a robust combiner exists. Conversely, if $\log N \leq (n - k)\lambda$, then no generic (k, n)-robust DLog-secure combiner exists.

These results identify a fundamental limitation of robust hedging at the group level. Decisional assumptions such as DDH cannot be robustly combined in the GGM, while search assumptions admit robustness only at essentially optimal representation cost. Consequently, robustness for group-based cryptography must in general be achieved at higher layers, such as protocol design or key derivation.

[†]The State Key Laboratory of Blockchain and Data Security, Zhejiang University, congresearch@zju.edu.cn
[‡]Virginia Commonwealth University.
[§]Google LLC, Columbia University.

Contents

1 Introduction **3**
- 1.1 Our results . . . 4
- 1.2 Technical overview . . . 4
  - 1.2.1 Impossibility of robust generic combiner for DDH . . . 6
  - 1.2.2 Impossibility of robust combiner for DLog . . . 10
- 1.3 Interpretation . . . 12
- 1.4 Related work . . . 12

2 Preliminaries **13**
- 2.1 Robust combiners . . . 13
- 2.2 Cryptographic groups . . . 14
- 2.3 Generic groups and algorithms . . . 15

3 Generic combiners for groups **16**
- 3.1 Functionality . . . 16
- 3.2 Robust generic combiners and corrupted coordinates . . . 17

4 Impossibility of robust generic combiner for DDH **18**
- 4.1 Partial information . . . 19
- 4.2 Breaking DDH-security via partial information . . . 19
- 4.3 Extracting partial information . . . 21
- 4.4 Extension to general case . . . 29

5 Impossibility of robust generic combiner for DLog **31**
- 5.1 Extracting the remaining information efficiently . . . 33

1 Introduction

A basic principle of robust design is to avoid placing complete trust in any single component, and cryptography is no exception. Robust combiners provide a principled way to hedge such uncertainty: instead of committing to a single candidate instantiation of a primitive, one combines several candidates into a single scheme that remains secure as long as sufficiently many of the underlying candidates remain secure. This paradigm is especially appealing when the long-term reliability of any individual assumption or implementation is unclear. Over the past two decades, robust combiners have been extensively studied for a range of cryptographic primitives, including hash functions [BB06, Pie07], public-key encryption [Her05, DK05], and oblivious transfer [HKN+05, MPW07].

Many cryptographic systems, however, are built not directly from such primitives, but from *cryptographic groups*; see, for example, the extensive literature on group-based cryptography [DH76, ElG84, CS03]. In particular, foundational hardness assumptions such as the discrete logarithm problem (DLog), the computational Diffie–Hellman (CDH) assumption, and the decisional Diffie–Hellman (DDH) assumption underlie a broad range of protocols and systems. This raises a natural question: can robustness already be achieved at the level of the underlying group?

A natural starting point is the direct-product construction. Given two cyclic groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ of orders p_1 and p_2, respectively[1], one may form the product group $\mathbb{G} = \mathbb{G}_1 \times \mathbb{G}_2$ with componentwise operations and generator $g = (g_1, g_2)$. For search assumptions such as DLog and CDH, this construction immediately provides a form of robustness: a DLog instance in the product group has the form

$$(g, g^x) = \big((g_1, g_2), (g_1^x, g_2^x)\big),$$

and recovering x requires solving the corresponding search problem in each active coordinate. Thus, direct products give a simple and natural baseline for robust group combination.

At the same time, the direct-product construction already exposes two fundamental obstacles. The first concerns decisional security. In the product group, a Diffie–Hellman tuple takes the form

$$(g, g^x, g^y, g^{xy}) = \big((g_1, g_2), (g_1^x, g_2^x), (g_1^y, g_2^y), (g_1^{xy}, g_2^{xy})\big),$$

whereas a random tuple takes the form

$$(g, g^x, g^y, g^z) = \big((g_1, g_2), (g_1^x, g_2^x), (g_1^y, g_2^y), (g_1^z, g_2^z)\big).$$

The combined instance therefore exposes its full coordinatewise algebraic structure, rather than hiding it inside a single group. As a result, once one component fails to preserve DDH security, the direct product need not preserve DDH either. This leads to our first question:

> *Can DDH-secure groups be robustly combined while retaining DDH security?*

The second obstacle is efficiency. Even when direct products preserve search hardness, they do so at a substantial representational cost: if the component groups have orders p_1 and p_2, then the combined group has order $p_1 p_2$, and each group element must carry both coordinates. Thus, robustness is obtained only at the price of increased group size and representation overhead. This leads to our second question:

> *Can DLog-secure groups be robustly combined with substantially less overhead than the direct-product construction?*

[1] For simplicity, we assume that p_1 and p_2 are distinct primes.

1.1 Our results

In this work, we answer both questions in the negative. Specifically, we formalize them within Maurer's generic group model [Mau05] and show that DDH-secure groups cannot be robustly combined in a generic manner. By contrast, for DLog-secure groups, we show that robust combination is possible only above a tight threshold on the group order.

We emphasize that studying these questions in the generic group model is particularly well motivated. For many algebraic constructions, generic approaches are among the most natural ones to consider, and they are often attractive from an efficiency perspective as well. Moreover, identifying the exact boundary of what robust combiners can achieve in the generic setting is not merely of theoretical interest; it may also have practical implications when choosing concrete security parameters.

Impossibility of robust combiner for DDH (Section 4). As discussed above, the direct-product construction cannot achieve DDH security: once a single underlying group fails to preserve DDH security, this already suffices to reveal the distinguishing bit in the DDH game for the combined group. We show that this is not merely a weakness of the direct-product approach, but a fundamental barrier in the DDH setting. In particular, we prove that robust generic combiners for DDH-secure groups are impossible.

Theorem 1.1 (Informal)**.** *Let λ be the security parameter. Let k and n be any polynomials in λ with $k < n$. Then there is no (k, n)-robust generic combiner for DDH-secure groups.*

A tight order threshold for robust combiners for DLog (Section 5). In contrast to the DDH setting, the direct-product construction already gives a generic combiner for DLog-secure groups, and it extends naturally to the (k, n)-robust setting. Concretely, let $(\mathbb{G}_1, g_1, p_1), \ldots, (\mathbb{G}_n, g_n, p_n)$ be n groups of orders $p_1, \ldots, p_n$, where the p_i's are distinct λ-bit primes. The combiner C is defined as the direct product of $n - k + 1$ underlying groups, and hence has order $N = \prod_{i=1}^{n-k+1} p_i$. For $x \in \mathbb{Z}_N$, we define

$$g^x = \left(g_1^{x \bmod p_1}, \ldots, g_{n-k+1}^{x \bmod p_{n-k+1}}\right).$$

This construction already requires the combined group to have order N with

$$\log N = (n - k + 1)\lambda.$$

Our result shows that this loss is inherent. Specifically, we prove a *tight* lower bound for (k, n)-robust generic combiners for DLog-secure groups: any such combiner must have group order at least $(n - k)\lambda$ bits. This roughly matches the direct-product construction above.

Theorem 1.2 (Informal)**.** *Let λ be the security parameter. Let k and n be constants with $k < n$. Then there is no (k, n)-robust generic combiner for DLog-secure groups of order N such that $\log N \leq (n - k)\lambda$.*

Remark 1.3. *The same threshold also applies to CDH-secure groups. The upper bound follows from the same direct-product construction, which preserves CDH security as well. The lower bound follows immediately from the DLog lower bound.*

1.2 Technical overview

The framework. We prove our results in Maurer's generic group model, which, together with the incomparable model of Shoup [Sho97], constitutes one of the two main frameworks for formalizing generic-group computation. At a high level, in both models, algorithms are given oracle

access to the group operation and to an equality test for group elements. The key difference lies in how oracle queries are specified.

In Maurer's model, an algorithm specifies a query by referring to group elements that have already appeared during the computation, for example, by indicating the 3rd and the 5th previously generated elements. Moreover, group elements in Maurer's model can only be manipulated through the oracle interface, namely via group operations and equality tests. In Shoup's model, by contrast, group elements come with explicit random encodings, and queries are specified in terms of those encodings. We refer the reader to Section 2.3 for a formal description of Maurer's generic group model.

For ease of exposition, we focus in this section on the special case of a $(1, 2)$-robust *generic* combiner for cryptographic groups. The relevant definitions and proof techniques extend to the general (k, n) setting in a straightforward manner.

Generic combiner for groups. Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two generic group oracles. We write $\hat{g}_1^{x_1}$ and $\hat{g}_2^{x_2}$ for implicit group elements in $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively; these elements are accessible to any algorithm only through the oracle interfaces. Following [KL14], a $(1, 2)$-robust generic combiner C for groups of order N is specified by a tuple of efficient generic algorithms $(\mathsf{L}, \mathsf{Add}, \mathsf{VT})$, each given oracle access to $\mathcal{O}_1$ and $\mathcal{O}_2$, and satisfying the following properties:

- The labeling algorithm L outputs a unique representation of the combined-group element corresponding to x, of the form $\mathsf{L}(x) := (\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}_1}, \mathsf{L}_{x\text{-}\mathsf{G}_2})$, where $\mathsf{L}_{x\text{-}\mathsf{str}}$ is an explicit bit string, and
$$\mathsf{L}_{x\text{-}\mathsf{G}_1} = \left(\hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}\right), \ \mathsf{L}_{x\text{-}\mathsf{G}_2} = \left(\hat{g}_2^{x_{2,1}}, \ldots, \hat{g}_2^{x_{2,t_2}}\right).$$
- The group-operation algorithm Add takes as input $\mathsf{L}(x)$ and $\mathsf{L}(y)$, computes the group operation, and outputs $\mathsf{L}(x + y)$.
- The validity-testing algorithm VT decides whether a given representation is a valid encoding of a combined-group element.

Robustness of generic combiners. Following [HKN+05], a robust combiner should remain secure as long as sufficiently many of its candidate inputs remain secure. In the present setting, however, the candidate inputs are themselves *generic groups*. This creates a modeling issue. Each generic group is unconditionally hard against any adversary making only polynomially many generic queries, so the standard notion of an efficient generic adversary is too weak to represent the idea that one of the underlying candidates is already "broken."

To model corruption meaningfully, we therefore consider a stronger adversarial model. The adversary is allowed to make an unbounded number of queries to exactly one of $\mathcal{O}_1$ and $\mathcal{O}_2$, where the choice of the corrupted oracle is fixed in advance and is not made adaptively during the attack.[2] The remaining oracle is still accessed only polynomially many times, and so must continue to satisfy the usual generic hardness guarantees.

This is precisely the form needed in our lower bounds. If there exists an adversary that is allowed unbounded access to one fixed oracle and nevertheless breaks the combined construction, then no $(1, 2)$-robust generic combiner can exist. Indeed, robustness would require a generic reduction that turns such an attack into successful attacks on *both* underlying groups. But the resulting reduction inherits unbounded access to at most one of the two groups. Consequently, one of the two component groups would have to be broken using only polynomially many generic

[2] Intuitively, the fixed oracle represents the underlying candidate that has already failed.

queries, contradicting unconditional generic hardness. Thus, for impossibility, it suffices to fix the corrupted oracle in advance, construct such an adversary, and show that it breaks any candidate combiner.

1.2.1 Impossibility of robust generic combiner for DDH

In the following, we present the high-level idea behind constructing a generic adversary $\mathcal{A}$ that has unbounded access to only one of the underlying oracles, yet breaks the DDH security of any generic combiner. To motivate this adversary, we first revisit the attack on the direct-product construction and then explain how it inspires the design of our adversary.

Attack on the direct-product construction. Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two generic groups of orders p_1 and p_2, respectively. The corresponding direct-product construction is defined by $\mathsf{L}(x) := (\hat{g}_1^{x_1}, \hat{g}_2^{x_2})$, where $x_1 = x \bmod p_1$ and $x_2 = x \bmod p_2$. Without loss of generality, suppose that the adversary $\mathcal{A}$ has unbounded access to $\mathcal{O}_1$. Then $\mathcal{A}$ can recover x_1 from the first component $\hat{g}_1^{x_1}$. For convenience, for every $x \in \mathbb{Z}_N$, define $P(x) := x \bmod p_1$.

Now consider the DDH game, in which the adversary $\mathcal{A}$ is given $\mathsf{L}(x)$, $\mathsf{L}(y)$, and $\mathsf{L}(z)$, where

$$\mathsf{L}(x) := (\hat{g}_1^{x_1}, \hat{g}_2^{x_2}), \qquad \mathsf{L}(y) := (\hat{g}_1^{y_1}, \hat{g}_2^{y_2}), \qquad \mathsf{L}(z) := (\hat{g}_1^{z_1}, \hat{g}_2^{z_2}).$$

From these labels, the adversary can recover $P(x)$, $P(y)$, and $P(z)$. The key observation is that, under the direct-product construction, for any $x, y \in \mathbb{Z}_N$,

$$P(x) \cdot P(y) \bmod p_1 \;=\; P(xy).$$

Therefore, if $(\mathsf{L}(x), \mathsf{L}(y), \mathsf{L}(z))$ is a DDH tuple, then necessarily

$$P(z) = P(x) \cdot P(y) \bmod p_1 = P(xy).$$

By contrast, if $\mathsf{L}(z)$ is sampled independently at random, then this equality holds only with negligible probability.

We now move to the general setting, where the combiner C represents an element x by a label of the form $\mathsf{L}(x) := (\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}_1}, \mathsf{L}_{x\text{-}\mathsf{G}_2})$, with

$$\mathsf{L}_{x\text{-}\mathsf{G}_1} = \left(\hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}\right) \text{ and } \mathsf{L}_{x\text{-}\mathsf{G}_2} = \left(\hat{g}_2^{x_{2,1}}, \ldots, \hat{g}_2^{x_{2,t_2}}\right).$$

Our strategy consists of three steps, which we now describe in turn.

Step 1: Defining partial information. In analogy with the direct-product case, we define the *partial information* of $\mathsf{L}(x)$ to be

$$\mathsf{P}(x) := (\mathsf{L}_{x\text{-}\mathsf{str}}, x_{1,1}, \ldots, x_{1,t_1}).$$

In the direct-product construction, this notion coincides exactly with $P(x)$. Moreover, given $\mathsf{L}(x)$, an adversary with unbounded access to $\mathcal{O}_1$ can trivially compute $\mathsf{P}(x)$.

We then show that the partial information becomes sufficient for breaking DDH security, provided that there exists a generic algorithm $\mathcal{A}_{\mathsf{Ext}}$, with unbounded access only to $\mathcal{O}_1$, such that on input $\mathsf{L}(x)$ and $\mathsf{L}(y)$, it outputs $\mathsf{P}(xy)$ with high probability, say $(> 9/10)$.

Step 2: Breaking DDH security from partial information. The attack follows the same basic intuition as in the direct-product construction. Given $\mathsf{L}(x)$, $\mathsf{L}(y)$, and $\mathsf{L}(z)$, the adversary first computes

$\mathsf{P}(x)$, $\mathsf{P}(y)$, and $\mathsf{P}(z)$, and then runs $\bar{\mathsf{P}} \leftarrow \mathcal{A}_{\mathsf{Ext}}(\mathsf{L}(x), \mathsf{L}(y))$. By assumption, $\mathcal{A}_{\mathsf{Ext}}$ outputs the correct value $\mathsf{P}(xy)$ with probability at least $9/10$. Hence, conditioned on this event, if $(\mathsf{L}(x), \mathsf{L}(y), \mathsf{L}(z))$ is a DDH tuple, then necessarily $\mathsf{P}(z) = \bar{\mathsf{P}}$.

To turn this into a distinguisher, it remains to argue that when z is replaced by an independently random value $r \in \mathbb{Z}_N$, the probability that $\mathsf{P}(r) = \mathsf{P}(xy)$ is small; say, at most $1/2$. Intuitively, if this were not the case, then there would exist some partial information

$$\mathsf{P}^* = (\mathsf{L}_{x\text{-}\mathsf{str}}{}^*, x^*_{1,1}, \ldots, x^*_{1,t_1})$$

such that $\Pr_{r \leftarrow \mathbb{Z}_N}[\mathsf{P}(r) = \mathsf{P}^*] > \frac{1}{2}$. Let $S^* \subseteq \mathbb{Z}_N$ denote the set of all values r satisfying $\mathsf{P}(r) = \mathsf{P}^*$. Then $|S^*| > N/2$, meaning that more than half of the elements of the combined group share the same explicit string part and the same $\mathcal{O}_1$-component.

We show that such a concentration phenomenon would already contradict the DLog security of the combiner. Indeed, consider an adversary $\mathcal{A}_2$ with unbounded access to $\mathcal{O}_2$. Given a challenge label

$$\mathsf{L}(x) = \big(\mathsf{L}_{x\text{-}\mathsf{str}}, (\hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}), (\hat{g}_2^{x_{2,1}}, \ldots, \hat{g}_2^{x_{2,t_2}})\big),$$

the adversary can recover the entire $\mathcal{O}_2$-component $(x_{2,1}, \ldots, x_{2,t_2})$ using its unbounded access to $\mathcal{O}_2$. It then forms the tuple

$$T = \big(\mathsf{L}_{x\text{-}\mathsf{str}}{}^*, (x^*_{1,1}, \ldots, x^*_{1,t_1}), (x_{2,1}, \ldots, x_{2,t_2})\big).$$

Since the combiner is generic, given any candidate $x \in \mathbb{Z}_N$, one can compute the $\mathcal{O}_1$- and $\mathcal{O}_2$-component of $\mathsf{L}(x)$ without making any queries to the oracles. Thus $\mathcal{A}_2$ can exhaustively search over all $x \in \mathbb{Z}_N$ and output the one whose $\mathcal{O}_1$- and $\mathcal{O}_2$-component equals T.

Whenever the challenge x lies in S^*, this procedure recovers the correct discrete logarithm. Since $|S^*| > N/2$, it follows that $\mathcal{A}_2$ succeeds with probability greater than $1/2$.

Therefore, for a uniformly random $r \in \mathbb{Z}_N$, the probability that $\mathsf{P}(r) = \mathsf{P}(xy)$ must be bounded away from 1. Combined with the extractor $\mathcal{A}_{\mathsf{Ext}}$, this suffices to distinguish DDH tuples from random ones.

Step 3: Extracting the partial information. We now explain how to extract the partial information $\mathsf{P}(xy)$ from $\mathsf{L}(x)$ and $\mathsf{L}(y)$. At first sight, this seems difficult, since the relationship among $\mathsf{P}(x)$, $\mathsf{P}(y)$, and $\mathsf{P}(xy)$ is no longer explicit. In particular, unlike in the direct-product case, there is in general no fixed function that, given only $\mathsf{P}(x)$ and $\mathsf{P}(y)$, directly computes $\mathsf{P}(xy)$.

To circumvent this obstacle, we consider extracting additional information from $\mathsf{P}(x)$ and $\mathsf{P}(y)$. Specifically, using the unbounded computational power, the adversary can compute the sets S_x and S_y, which consist of all values whose corresponding partial information equals $\mathsf{P}(x)$ and $\mathsf{P}(y)$, respectively[3]. That is,

$$S_x := \{\tilde{x} \in \mathbb{Z}_N \mid \mathsf{P}(\tilde{x}) = \mathsf{P}(x)\} \text{ and } S_y := \{\tilde{y} \in \mathbb{Z}_N \mid \mathsf{P}(\tilde{y}) = \mathsf{P}(y)\}.$$

Note that, in the direct-product case, the following two properties hold:

- For every $\tilde{x} \in S_x$, we have $\underline{\mathsf{P}(\tilde{x}y)} = \mathsf{P}(\tilde{x}) \cdot \mathsf{P}(y) = \mathsf{P}(x) \cdot \mathsf{P}(y) = \underline{\mathsf{P}(xy)}$.
- Fix any $\tilde{x} \in S_x$. Then, for every $\tilde{y} \in S_y$, we have $\underline{\mathsf{P}(\tilde{x}\tilde{y})} = \mathsf{P}(\tilde{x}) \cdot \mathsf{P}(\tilde{y}) = \mathsf{P}(\tilde{x}) \cdot \mathsf{P}(y) = \underline{\mathsf{P}(\tilde{x}y)}$.

[3] In Shoup's GGM, $\mathsf{P}(x)$ may also depend on the random encoding of underlying generic groups, and hence S_x cannot in general be recovered by brute force over x alone, without making additional queries to $\mathcal{O}_1$ and $\mathcal{O}_2$. This is exactly why we establish our results in Maurer's GGM rather than in Shoup's model.

Combining these two properties, we conclude that for every $\tilde{x} \in S_x$ and every $\tilde{y} \in S_y$, the partial information of $\mathsf{L}(\tilde{x}\tilde{y})$ is equal to that of $\mathsf{L}(xy)$. This suggests that, even in the general case, one may hope that a sufficiently large fraction of pairs $(\tilde{x}, \tilde{y}) \in S_x \times S_y$ satisfy $\mathsf{P}(\tilde{x}\tilde{y}) = \mathsf{P}(xy)$. If so, the extractor $\mathcal{A}_{\mathsf{Ext}}$ can compute $\mathsf{P}(\tilde{x}\tilde{y})$ for all pairs $(\tilde{x}, \tilde{y}) \in S_x \times S_y$, and output the majority value as its guess for $\mathsf{P}(xy)$.

To understand why these two properties hold in the direct-product case, it is helpful to examine how $\mathsf{L}(xy)$ is computed. Observe that $\mathsf{L}(xy)$ can be obtained by adding $\mathsf{L}(x)$ to itself y times, which can be carried out in $O(\log y)$ operations via repeated doubling. We write

$$\mathsf{L}(xy) \leftarrow \mathsf{Scale}\big(\mathsf{L}(x), y\big) := \mathsf{Add}\big(\underbrace{\mathsf{L}(x), \ldots, \mathsf{L}(x)}_{y}\big).$$

In the direct-product case, where $\mathsf{L}(x) = (\hat{g}_1^{x_1}, \hat{g}_2^{x_2})$, the procedure $\mathsf{Scale}\big(\mathsf{L}(x), y\big)$ simply adds $\hat{g}_1^{x_1}$ and $\hat{g}_2^{x_2}$ to themselves $(y \bmod p_1)$ and $(y \bmod p_2)$ times, respectively. Consequently,

$$\mathsf{L}(xy) = \left(\hat{g}_1^{xy \bmod p_1}, \hat{g}_2^{xy \bmod p_2}\right) = \left(\mathcal{O}_1^{\mathsf{Add}}\big(\underbrace{\hat{g}_1^{x_1}, \ldots, \hat{g}_1^{x_1}}_{y \bmod p_1}\big), \mathcal{O}_2^{\mathsf{Add}}\big(\underbrace{\hat{g}_2^{x_2}, \ldots, \hat{g}_2^{x_2}}_{y \bmod p_2}\big)\right).$$

It follows that, in the direct-product case, the partial information of $\mathsf{L}(xy)$ is completely determined by $\mathsf{P}(x)$ and y. Therefore, for every $\tilde{x}$ such that $\mathsf{P}(\tilde{x}) = \mathsf{P}(x)$ and the same multiplier y, we have $\mathsf{P}(\tilde{x}y) = \mathsf{P}(xy)$. (This is exactly the first property above; the second property follows from the same reasoning.)

Difficulty. We now extend the above view of the execution of Scale to the general case, where

$$\mathsf{L}(xy) \leftarrow \mathsf{Scale}\big(\mathsf{L}(x), y\big) = \mathsf{Scale}\big((\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}_1}, \mathsf{L}_{x\text{-}\mathsf{G}_2}), y\big),$$

with $\mathsf{L}(xy) = (\mathsf{L}_{xy\text{-}\mathsf{str}}, \mathsf{L}_{xy\text{-}\mathsf{G}_1}, \mathsf{L}_{xy\text{-}\mathsf{G}_2})$. Since Scale is generic, the bit-string part of the output, namely $\mathsf{L}_{xy\text{-}\mathsf{str}}$, can depend only on the input bit string $\mathsf{L}_{x\text{-}\mathsf{str}}$, the multiplier y, and the responses res_1 and res_2 to the equality-test queries made to $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. Similarly, each group element in $\mathsf{L}_{xy\text{-}\mathsf{G}_1}$ is obtained as a linear combination of the elements in $\mathsf{L}_{x\text{-}\mathsf{G}_1}$, where the corresponding coefficients are determined by $(\mathsf{L}_{x\text{-}\mathsf{str}}, y, \mathsf{res}_1, \mathsf{res}_2)$.

In other words, the partial information of $\mathsf{L}(xy)$ is completely determined by

$$(\mathsf{P}(x), y, \mathsf{res}_1, \mathsf{res}_2).$$

Thus, exactly as in the direct-product case, if the executions of $\mathsf{Scale}\big(\mathsf{L}(x), y\big)$ and $\mathsf{Scale}\big(\mathsf{L}(\tilde{x}), y\big)$ give rise to the same tuples

$$(\mathsf{P}(x), y, \mathsf{res}_1, \mathsf{res}_2) = (\mathsf{P}(\tilde{x}), y, \widetilde{\mathsf{res}}_1, \widetilde{\mathsf{res}}_2),$$

then the resulting partial information must also be identical. In particular, this implies $\mathsf{P}(\tilde{x}y) = \mathsf{P}(xy)$.

At this point, since $\mathsf{P}(x) = \mathsf{P}(\tilde{x})$, the only possible source of discrepancy is the $\mathcal{O}_2$-part, namely $(x_{2,1}, \ldots, x_{2,t_2})$ and $(\tilde{x}_{2,1}, \ldots, \tilde{x}_{2,t_2})$. Indeed, although the two executions start from the same partial information, they may behave differently on $\mathcal{O}_2$, which in turn may lead to different responses res_2 and $\widetilde{\mathsf{res}}_2$. In particular, different res_2 and $\widetilde{\mathsf{res}}_2$ may change both the output bit string and the coefficients of the linear combinations that define the elements in the $\mathcal{O}_1$-part of the output. As a result, even though $\mathsf{P}(x) = \mathsf{P}(\tilde{x})$, one may still have $\mathsf{P}(\tilde{x}y) \neq \mathsf{P}(xy)$.

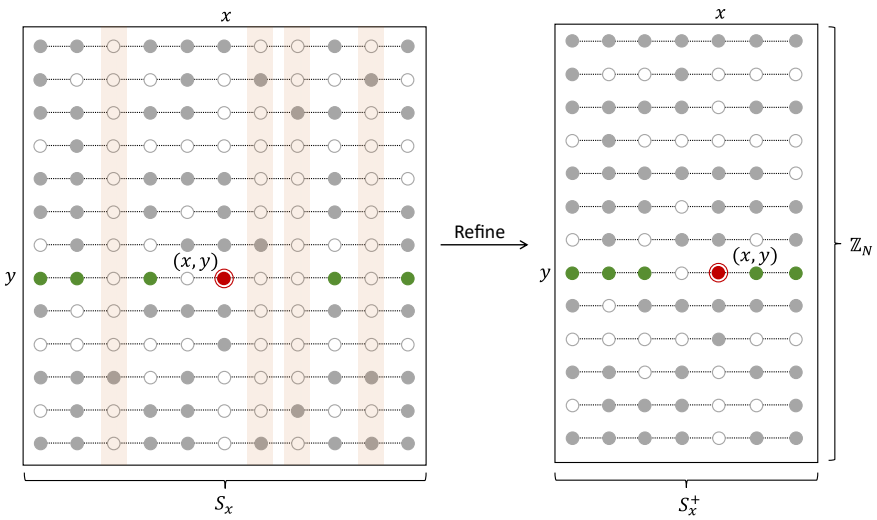

Figure 1: Refining S_x into S_x^+ over $\mathbb{Z}_N$.

Our core techniques. Intuitively, we would like the sets S_x and S_y to satisfy the following:

- There exist sufficiently many $\tilde{x} \in S_x$ such that, when running $\mathsf{Scale}\big(\mathsf{L}(\tilde{x}), y\big)$, the responses to the equality-test queries to $\mathcal{O}_2$ coincide with those arising in the execution of $\mathsf{Scale}\big(\mathsf{L}(x), y\big)$. This would imply $\mathsf{P}(\tilde{x}y) = \mathsf{P}(xy)$.
- Fix such a $\tilde{x} \in S_x$. Then there exist sufficiently many $\tilde{y} \in S_y$ such that, when running $\mathsf{Scale}\big(\mathsf{L}(\tilde{y}), \tilde{x}\big)$, the responses to the equality-test queries to $\mathcal{O}_2$ coincide with those arising in the execution of $\mathsf{Scale}\big(\mathsf{L}(y), \tilde{x}\big)$. This would imply $\mathsf{P}(\tilde{x}\tilde{y}) = \mathsf{P}(\tilde{x}y)$.

Therefore, our core idea is to iteratively filter out from S_x and S_y those candidates $\tilde{x}$ and $\tilde{y}$ that do not satisfy the above consistency conditions, while ensuring that the remaining "good" candidates still constitute a sufficiently large fraction of the sets.

Filtering out bad $\tilde{x}$ from S_x. We say that $\tilde{x}$ is *good with respect to* y if the executions of Scale on inputs $\big(\mathsf{L}(\tilde{x}), y\big)$ and $\big(\mathsf{L}(x), y\big)$ produce identical responses to the equality-test queries made to $\mathcal{O}_2$. The purpose of this step is to refine S_x into a subset S_x^+ such that, for a uniformly random choice of y, with high probability a sufficiently large fraction of candidates $\tilde{x} \in S_x^+$ are good with respect to y. Accordingly, the refinement procedure proceeds as follows:

- **Step 1: Testing phase.** Randomly sample a collection of multipliers $(r_1, \ldots, r_t)$ from $\mathbb{Z}_N$. If, for every r_i, a sufficiently large fraction of candidates $\tilde{x} \in S_x$ are good with respect to r_i, then output S_x as S_x^+. Otherwise, proceed to Step 2.
- **Step 2: Filtering phase.** For each $\tilde{x} \in S_x$, if there exists some r_i sampled in Step 1 such that $\tilde{x}$ is bad with respect to r_i, then remove $\tilde{x}$ from S_x. After that, return to Step 1.

Intuitively, suppose that for a uniformly random choice of y, the original set S_x contains a large fraction of bad candidates $\tilde{x}$, as illustrated on the left-hand side of Figure 1. Then, provided that the testing phase samples sufficiently many random multipliers r—while still keeping the number of samples polynomial—such a set S_x is unlikely to pass Step 1. On the other hand, these bad candidates are also more likely to be detected and removed in Step 2. As the refinement procedure is repeated, the fraction of good candidates in the remaining set gradually increases,

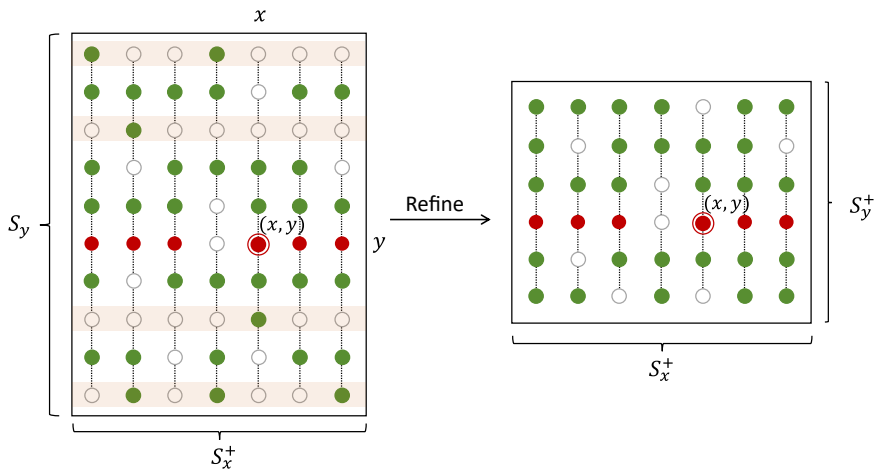

Figure 2: Refining S_y into S_y^+ over S_x^+.

until eventually the set becomes stable enough to pass the testing phase and is output as S_x^+, as illustrated on the right-hand side of Figure 1.

Filtering out bad $\tilde{y}$ from S_y. Suppose that we have already refined S_x into a subset S_x^+. Then, as illustrated on the left-hand side of Figure 2, the row corresponding to the fixed value y contains many red points $(\tilde{x}, y)$ such that $\mathsf{P}(\tilde{x}y) = \mathsf{P}(xy)$.

However, this row alone accounts for only a small fraction of the entire product set $S_x^+ \times S_y$. To propagate this consistency to a large portion of the whole grid, we further refine S_y into S_y^+. More concretely, for each such red point $(\tilde{x}, y)$, we would like the corresponding column to contain a sufficiently large fraction of values $\tilde{y} \in S_y^+$ such that $\mathsf{P}(\tilde{x}\tilde{y}) = \mathsf{P}(\tilde{x}y)$. The refinement procedure for S_y is analogous to that for S_x:

- **Step 1: Testing phase.** Randomly sample a collection of values $(\tilde{x}_1, \ldots, \tilde{x}_t)$ from S_x^+. If, for every sampled $\tilde{x}_i$, a sufficiently large fraction of candidates $\tilde{y} \in S_y$ are consistent with y under $\tilde{x}_i$, then output S_y as S_y^+. Otherwise, proceed to Step 2.
- **Step 2: Filtering phase.** For each $\tilde{y} \in S_y$, if there exists some sampled $\tilde{x}_i$ from Step 1 such that $\tilde{y}$ is inconsistent with y under $\tilde{x}_i$, then remove $\tilde{y}$ from S_y. After that, return to Step 1.

Intuitively, suppose that for a candidate $\tilde{y}$, there exist many values $\tilde{x}$ such that $(\tilde{x}, \tilde{y})$ is inconsistent with $(\tilde{x}, y)$. If many such $\tilde{y}$ remain in the set, then the current set is unlikely to pass Step 1. At the same time, these $\tilde{y}$ are also more likely to be detected and removed in Step 2. Consequently, among the surviving candidates, each column corresponding to a fixed $\tilde{x}$ becomes more likely to remain consistent with $\mathsf{P}(\tilde{x}y)$, as illustrated on the right-hand side of Figure 2.

To summarize, if the refinement procedure eventually produces subsets S_x^+ and S_y^+ satisfying the above properties, then a sufficiently large fraction of pairs $(\tilde{x}, \tilde{y}) \in S_x^+ \times S_y^+$ will satisfy $\mathsf{P}(\tilde{x}\tilde{y}) = \mathsf{P}(xy)$. This means that the adversary can effectively recover the partial information of $\mathsf{L}(xy)$.

The above outline is not comprehensive; for detailed technical explanations, please refer to Section 4.

1.2.2 Impossibility of robust combiner for DLog

In contrast to DDH, the direct-product construction already gives a $(1, 2)$-robust generic combiner for DLog-secure groups, where the combined group has order $N = p_1 \cdot p_2$ and hence bit-length

2λ. We then show that this is in fact optimal: specifically, there is no $(1,2)$-robust combiner for DLog-secure groups if the bit-length of the combined group order is at most λ

Consider a generic combiner C of the labeling form

$$\mathsf{L}(x) := \left(\mathsf{L}_{x\text{-}\mathsf{str}}, \hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}, \hat{g}_2^{x_{2,1}}, \ldots, \hat{g}_2^{x_{2,t_2}}\right).$$

The adversary is given a challenge combined-group element $\mathsf{L}(x)$, where x is sampled uniformly at random from $\mathbb{Z}_N$, and may make an unbounded number of queries to either $\mathcal{O}_1$ or $\mathcal{O}_2$. Clearly, the adversary can directly recover partial information from $\mathsf{L}(x)$, namely either $(\mathsf{L}_{x\text{-}\mathsf{str}}, x_{1,1}, \ldots, x_{1,t_1})$ or $(\mathsf{L}_{x\text{-}\mathsf{str}}, x_{2,1}, \ldots, x_{2,t_2})$.

However, while such partial information may already suffice to distinguish a DDH tuple from a random one, breaking DLog appears to require substantially more: namely, recovering the full information

$$\mathsf{Inf}(x) := (\mathsf{L}_{x\text{-}\mathsf{str}}, x_{1,1}, \ldots, x_{1,t_1}, x_{2,1}, \ldots, x_{2,t_2}).$$

Indeed, since the combiner is generic, this information is determined solely by x. Therefore, once the adversary recovers the full information above, it can use its unbounded computational power to exhaustively search for a value x that matches it, without making further queries to $\mathcal{O}_1$ or $\mathcal{O}_2$.

To bridge the gap, we present a new observation based on the validity-testing algorithm VT, and show that when $\log N \leq \lambda$, the missing information can in fact be recovered *efficiently*. Specifically, the validity-testing algorithm VT takes candidate labeling as input, namely

$$1/0 \leftarrow \mathsf{VT}\left(\mathsf{L}_{x\text{-}\mathsf{str}}, \hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}, \hat{g}_2^{x_{2,1}}, \ldots, \hat{g}_2^{x_{2,t_2}}\right),$$

and decides whether this input represents a valid combined-group element. Since VT is generic, its output bit is determined solely by $\mathsf{L}_{x\text{-}\mathsf{str}}$ together with the responses res_1 and res_2 returned to its equality-testing queries to $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. In other words, a tuple $(\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{res}_1, \mathsf{res}_2)$ determines the output of VT to be 1 or 0.

Consider the execution of VT on input $\mathsf{L}(x)$. Each equality-testing query made to $\mathcal{O}_1$ corresponds to a linear combination of $\left(\hat{g}_1, \hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}\right)$ with coefficient vector $(\alpha_0, \ldots, \alpha_{t_1})$. A positive answer then yields a linear equation

$$\alpha_0 + \sum_{i=1}^{t_1} \alpha_i \cdot x_{1,i} = 0.$$

Let $\mathcal{L}_1$ denote the set of all coefficient vectors corresponding to positive answers from $\mathcal{O}_1$. Similarly, let $\mathcal{L}_2$ denote the analogous set for $\mathcal{O}_2$. Observe that if either $\mathcal{L}_1$ or $\mathcal{L}_2$ has full rank, then the corresponding discrete logarithms $(x_{1,1}, \ldots, x_{1,t_1})$ or $(x_{2,1}, \ldots, x_{2,t_2})$ can be fully recovered by solving the corresponding linear system. The remaining information can be obtained directly from the corrupted group, which it has unbounded access to. Therefore, in either case, the adversary recovers the entire $\mathsf{Inf}(x)$, and can thus determine x.

It remains to consider the case where neither $\mathcal{L}_1$ nor $\mathcal{L}_2$ has full rank. In this case, each corresponding linear system has at least one degree of freedom. Consequently, there exist many alternative assignments $(\tilde{x}_{1,1}, \ldots, \tilde{x}_{1,t_1})$ and $(\tilde{x}_{2,1}, \ldots, \tilde{x}_{2,t_2})$ that satisfy exactly the same linear constraints. In particular, the number of such assignments is at least p_1 on the $\mathcal{O}_1$-side and at least p_2 on the $\mathcal{O}_2$-side. It follows that there exist at least $p_1 p_2$ assignments such that the corresponding

representation is accepted by the validity-testing algorithm, namely,

$$\mathsf{VT}\left(\mathsf{L}_{x\text{-}\mathsf{str}}, \hat{g}_1^{\tilde{x}_{1,1}}, \ldots, \hat{g}_1^{\tilde{x}_{1,t_1}}, \hat{g}_2^{\tilde{x}_{2,1}}, \ldots, \hat{g}_2^{\tilde{x}_{2,t_2}}\right) = 1.$$

This is because each such assignment induces exactly the same tuple $(\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{res}_1, \mathsf{res}_2)$ as $\mathsf{L}(x)$ in the execution of VT. However, if $\log N \leq \lambda$, then $N < p_1 p_2$. Since the combined group contains exactly N valid elements, this would imply that VT accepts more representations than there are valid combined-group elements, contradicting the correctness of the validity-testing functionality.

The above outline is not comprehensive; for detailed technical explanations, please refer to Section 5.

1.3 Interpretation

Our results admit two complementary interpretations.

Practical perspective. Our results identify a fundamental limitation of what may be viewed as *algebraic hedging*: one cannot, in general, obtain robustness for group-based cryptography by combining multiple candidate groups within the group abstraction itself. Thus, if the goal is to hedge against uncertainty in the underlying algebraic assumption, then robustness should instead be sought at higher layers of the cryptographic stack—for example, through protocol-level design, compiler transformations, or key-derivation mechanisms—rather than through the construction of a universally robust combined group.

Theoretical perspective. Theoretically, our results reveal a sharp boundary for robust combination in algebraic settings. First, our work turns a long-standing folklore intuition into a formal theorem: robust combiners for DDH-secure groups do not exist in the generic setting. Second, we show that this impossibility is not merely a pathology of the model, but reflects a deeper divide between decisional and search assumptions: DDH is generically non-combinable, whereas DLog admits robust combination only above a tight order threshold. In this sense, our work uncovers a qualitative asymmetry between decisional and search hardness in the robust-combiner regime for groups.

1.4 Related work

Robust Combiners. The formal notion of robust combiners was introduced by [HKN+05] in the context of oblivious transfer and related primitives. Since then, combiners have been studied for a broad range of cryptographic tasks. A major line of work concerns hash-function combiners: Boneh and Boyen [BB06] showed that collision-resistant hash combiners cannot asymptotically beat concatenation in output length, a barrier that motivated much of the subsequent literature [FL08, FLP08, Pie07, Pie08, Mit12]; later, [DFG+23] showed that this barrier can be overcome in the random-oracle setting. Robust combiners have also been studied for several other primitives, including oblivious transfer [HKN+05, MPW07, CDFR17, FRG23], public-key encryption [DK05, HKN+05, Her09], key encapsulation mechanisms [GHP18, HMS19], commitment schemes [MP06, Her09], private information retrieval [MP06], password-authenticated key exchange [LL25, HR25], oblivious pseudorandom functions [FFHH25], non-interactive zero knowledge [CMVX25], indistinguishability obfuscation [AJN+16], functional encryption [AJS17, JMS20] and several quantum cryptographic primitives [HKNY24], including quantum bit commitments, unclonable encryption, public-key quantum money.

Impossibility results in the generic group model. A related line of work studies impossibility results in the generic group model. In Shoup's generic group model, a number of impossibility results have been established, including for virtual-black-box obfuscation [Ps16], order-revealing encryption [ZZ18], and identity-based encryption [Zha22]; related limitations were also extended to ideal non-commutative ring models [MMN16]. In Maurer's GGM, Rotem, Segev, and Shahaf [RSS20] showed that generic-group delay functions in groups of known order are impossible, giving evidence that hidden-order groups may be inherently necessary for realizing delay in this setting. Schul-Ganz and Segev [SGS21] obtained a tight impossibility result for generic-group identity-based encryption, proving that any IBE scheme whose public parameters contain n group elements can support at most n identity secret keys. [DHH+21] established impossibility results for purely algebraic signatures, showing that algebraic verification alone imposes strong structural limitations on efficient signature constructions in Maurer's model.

2 Preliminaries

Notations. Let $\lambda \in \mathbb{N}$ denote the security parameter. For a positive integer n, we write $[n] = \{1, 2, \ldots, n\}$. For a finite set S, the notation $s \xleftarrow{\$} S$ means that s is sampled uniformly at random from S, and $|S|$ denotes the cardinality of S. A function $\mathsf{negl}(\cdot)$ is negligible if for every positive polynomial $\mathsf{poly}(\cdot)$ there exists $\lambda^* \in \mathbb{N}$ such that for all $\lambda > \lambda^*$, $\mathsf{negl}(\lambda) < 1/\mathsf{poly}(\lambda)$. A function $\epsilon(\cdot)$ is noticeable if there exists a positive polynomial $\mathsf{poly}(\cdot)$ and $\lambda^* \in \mathbb{N}$ such that for all $\lambda > \lambda^*$, $\epsilon(\lambda) \geq 1/\mathsf{poly}(\lambda)$. We write PPTM for a probabilistic polynomial-time Turing machine, and oracle PPTM for a PPTM with access to one or more oracles.

2.1 Robust combiners

We follow the formulation of robust combiners for cryptographic primitives from [HKN+05]. We begin by recalling the notion of a cryptographic primitive.

Definition 2.1 (Cryptographic Primitive [RTV04])**.** *A primitive $\mathcal{P}$ is a pair $\langle \mathcal{F}, \mathcal{R} \rangle$, where $\mathcal{F}$ is a set of functions $f : \{0,1\}^* \to \{0,1\}^*$ specifying correctness, and $\mathcal{R}$ is a relation on pairs $\langle f, \mathcal{A} \rangle$, where $f \in \mathcal{F}$ and $\mathcal{A}$ is an adversarial machine, specifying security.*

- ***Efficient implementation.*** *A function f is an* implementation *of $\mathcal{P}$ if $f \in \mathcal{F}$. An* efficient implementation *of $\mathcal{P}$ is an implementation that is computable in polynomial time.*
- ***Secure implementation.*** *An adversarial machine $\mathcal{A}$ is said to* $\mathcal{P}$-break *f if $\langle f, \mathcal{A} \rangle \in \mathcal{R}$. An implementation f is* secure *if no PPT adversary $\mathcal{A}$ $\mathcal{P}$-breaks f.*

An efficient implementation of $\mathcal{P}$ will also be called a *candidate* for $\mathcal{P}$. A robust combiner takes several candidates for the same primitive and outputs a single implementation that remains secure as long as sufficiently many of the candidates are secure.

Definition 2.2 ((k,n)-Robust Combiner [HKN+05])**.** *Let $\mathcal{P}$ be a cryptographic primitive. A (k,n)-robust combiner for $\mathcal{P}$ is a PPTM C that takes n candidate implementations as input and implements $\mathcal{P}$ while satisfying the following properties:*

- ***Robustness.*** *If at least k of the input candidates securely implement $\mathcal{P}$, then C also securely implements $\mathcal{P}$.*

- ***Efficiency.*** *The running time of* C *is polynomial in the security parameter* λ*, in* n*, and in the input lengths of the primitive.*

The definition above allows *trivial* combiners that ignore the candidate implementations and realize the primitive independently. Such constructions are usually uninformative, since their security does not derive from the candidates themselves. To exclude this behavior, one often requires the construction and its proof to be *black-box*.

Definition 2.3 (Black-Box Combiner [HKN+05])**.** *A* (k, n)*-robust combiner is said to be* black-box *if the following conditions hold:*

- ***Black-box implementation.*** *The combiner is an oracle PPTM that accesses the candidate implementations only via oracle calls to their implementation functions.*
- ***Black-box proof.*** *For every candidate implementation, there exists an oracle PPTM* $\mathcal{R}$ *with oracle access to* $\mathcal{A}$ *such that if* $\mathcal{A}$ *breaks the combiner, then* $\mathcal{R}^{\mathcal{A}}$ *breaks at least* $n - k + 1$ *of the candidate implementations.*

2.2 Cryptographic groups

We focus on robust combiners for cryptographic groups. A group-generation algorithm outputs the description of a finite cyclic group together with a generator. Following [KL14], we assume that the group description includes efficient procedures for the group operation and for testing whether a given bit string encodes a valid group element. For simplicity, we further assume that group elements are represented uniquely.

Definition 2.4 (DLog- and DDH-Secure Groups)**.** *Let* $\mathcal{F}_{\mathrm{grp}}$ *be the set of group-generation functions* f *such that, on input* 1^λ*, the function* f *outputs a description*

$$(\mathbb{G}, g, N) \leftarrow f(1^\lambda),$$

where $\mathbb{G}$ *is a finite cyclic group of order* N*,* g *is a generator of* $\mathbb{G}$*, and the largest prime divisor of* N *is a* λ*-bit prime.*

We define two induced primitives, both with functionality family $\mathcal{F}_{\mathrm{grp}}$*:*

- ***DLog-secure group.*** *Let* $\mathcal{R}^{\mathrm{DLog}}$ *be the relation such that* $\langle f, \mathcal{A}\rangle \in \mathcal{R}^{\mathrm{DLog}}$ *if there exists a polynomial* $\mathsf{poly}_{\mathcal{A}}(\cdot)$ *such that*

$$\Pr\big[\mathcal{A}(\mathbb{G}, g, N, g^x) = x\big] > \frac{1}{\mathsf{poly}_{\mathcal{A}}(\lambda)}$$

for infinitely many values of λ*, where* $(\mathbb{G}, g, N) \leftarrow f(1^\lambda)$ *and* $x \xleftarrow{\$} \mathbb{Z}_N$*. The corresponding primitive is*

$$\mathcal{P}^{\mathrm{DLog}} := \langle \mathcal{F}_{\mathrm{grp}}, \mathcal{R}^{\mathrm{DLog}}\rangle.$$

- ***DDH-secure group.*** *Let* $\mathcal{R}^{\mathrm{DDH}}$ *be the relation such that* $\langle f, \mathcal{A}\rangle \in \mathcal{R}^{\mathrm{DDH}}$ *if there exists a polynomial* $\mathsf{poly}_{\mathcal{A}}(\cdot)$ *such that*

$$\Big|\Pr\big[\mathcal{A}(\mathbb{G}, g, N, g^x, g^y, g^{xy \bmod N}) = 1\big] - \Pr\big[\mathcal{A}(\mathbb{G}, g, N, g^x, g^y, g^z) = 1\big]\Big| > \frac{1}{\mathsf{poly}_{\mathcal{A}}(\lambda)}$$

for infinitely many values of λ*, where* $(\mathbb{G}, g, N) \leftarrow f(1^\lambda)$ *and* $x, y, z \xleftarrow{\$} \mathbb{Z}_N$*. The corresponding primitive is*

$$\mathcal{P}^{\mathrm{DDH}} := \langle \mathcal{F}_{\mathrm{grp}}, \mathcal{R}^{\mathrm{DDH}}\rangle.$$

Remark 2.5 (About CDH)**.** *The analogous search assumption for CDH can be formulated in exactly the same style. We do not foreground CDH in Sections 2.3 and 3, because the main conceptual contrast in this paper is between the search-type assumption DLog and the decisional assumption DDH. When needed later, the CDH formulation can be treated as the obvious intermediate search variant.*

2.3 Generic groups and algorithms

In the setting of group-based cryptography, generic constructions are often the most natural place to start: they capture what can be done using only the algebraic interface of the candidate groups, without exploiting any accidental feature of a concrete encoding. Since our goal is to understand whether robustness can already be achieved *at the group layer*, we focus on *generic* robustness, namely the setting in which both the combiner and the reduction are required to be generic. We formalize this using Maurer's generic group model (GGM) [Mau05].

Definition 2.6 (Generic Group Model [Mau05])**.** *Let N be a positive integer whose largest prime divisor is a λ-bit prime. The generic group oracle $\mathcal{O}$ maintains a counter* $\mathsf{ctr} \leftarrow 1$ *and an initially empty table* $\mathbf{B}$*, where each entry stores a value in $\mathbb{Z}_N$. Let V_i denote the value stored in the i-th entry. The oracle $\mathcal{O}$ provides the following interfaces:*

- ***Labeling.*** *On input $x \in \mathbb{Z}_N$, store x as V_{ctr}, return* ctr*, and increment* ctr*.*
- ***Group operation.*** *On input two integers $i, j <$* ctr*, store $(V_i + V_j) \bmod N$ as V_{ctr}, return* ctr*, and increment* ctr*.*
- ***Equality test.*** *On input two integers $i, j <$* ctr*, return 1 if $V_i = V_j$ and 0 otherwise.*

A generic algorithm receives handles to group elements rather than their concrete representations. Formally, suppose $\mathcal{A}$ is given t group elements as input, say $(g^{x_1}, \ldots, g^{x_t})$, together with an optional auxiliary bit string. Instead of receiving explicit encodings of these group elements, $\mathcal{A}$ is given their corresponding indices $(X_1, \ldots, X_t)$ in the table $\mathbf{B}$ maintained by the relevant generic-group oracle, where the first t entries of $\mathbf{B}$ store $x_1, \ldots, x_t$, respectively. The algorithm may then access and manipulate these group elements only through queries to the oracle interface.

We also allow different generic algorithms to exchange group elements during an interactive computation. If an algorithm outputs k group elements, it outputs the indices of k entries in its local table. When these values are passed to another generic algorithm, the recipient starts with a fresh table whose first k entries contain the corresponding group elements.

Finally, in our setting a generic algorithm may interact with multiple generic groups simultaneously. For generic-group oracles $\mathcal{O}_1, \ldots, \mathcal{O}_m$, each oracle maintains its own table and counter. Indices produced by $\mathcal{O}_i$ refer only to entries in the table of $\mathcal{O}_i$ and cannot be used as inputs to another oracle $\mathcal{O}_j$ for $j \neq i$.

Remark 2.7 (DLog and DDH within the GGM)**.** *Within Maurer's model, the DLog and DDH problems are interpreted in the usual way, except that the adversary sees only handles. For DLog, the adversary is given handles for g and g^x and must recover $x \in \mathbb{Z}_N$. For DDH, the adversary is given handles for g, g^x, g^y, and either g^{xy} or g^z, and must distinguish the two cases. Here "efficient generic adversary" means an adversary making only polynomially many oracle queries. Thus, the generic model isolates the algebraic information that can be extracted from the oracle interface alone.*

The standard notion of efficient generic computation recalled above is enough to talk about ordinary DLog and DDH hardness. For robust combiners, however, it is still too weak: robustness

must continue to hold even when up to $n - k$ component groups are regarded as completely broken. In Section 3, we therefore strengthen the adversarial model by allowing unrestricted generic access on a designated subset of component groups.

3 Generic combiners for groups

In this section, we specialize the robust-combiner framework to the setting of cryptographic groups. Our focus is on *generic* combiners, namely constructions that interact with the candidate groups only through their generic-group interfaces. The point of this section is to formalize two issues that drive the later impossibility proofs: first, what a combined group element can look like in the generic setting; and second, how to model the fact that up to $n - k$ component groups may be completely corrupted.

3.1 Functionality

Let $\mathcal{O}_1, \ldots, \mathcal{O}_n$ be n independent generic group oracles, where each $\mathcal{O}_i$ has prime order p_i. Let C be a combiner that, relative to $\mathcal{O}_1, \ldots, \mathcal{O}_n$, is intended to realize a cyclic group of order N. A valid implementation of the combined group must provide efficient algorithms for labeling, group operation, equality testing, and validity testing.

The key structural point is that a combined element is no longer a single group element. Instead, it is a label assembled from an explicit bit-string part together with tuples of handles in the underlying generic groups. Since the combiner is generic, this label can depend only on the abstract input exponent $x \in \mathbb{Z}_N$ and on the algebraic interfaces of the component groups; it cannot depend on any concrete encoding of the underlying group elements.

Definition 3.1 (Generic Group Combiner Functionality)**.** *Let $\mathcal{O}_1, \ldots, \mathcal{O}_n$ be independent generic groups, where $\mathcal{O}_i$ has prime order p_i. A generic group combiner for order N consists of efficient generic algorithms $(\mathsf{L}, \mathsf{Add}, \mathsf{Eq}, \mathsf{VT})$ satisfying the following properties.*

- ***Labeling.*** *On input $x \in \mathbb{Z}_N$, the algorithm outputs a representation*

$$\mathsf{L}(x) = (\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}}),$$

  where $\mathsf{L}_{x\text{-}\mathsf{str}} \in \{0,1\}^*$ is an explicit bit string and

$$\mathsf{L}_{x\text{-}\mathsf{G}} = \Big((\hat{g}_1^{f_{1,1}(x)}, \ldots, \hat{g}_1^{f_{1,t_1}(x)}), \ldots, (\hat{g}_n^{f_{n,1}(x)}, \ldots, \hat{g}_n^{f_{n,t_n}(x)}) \Big)$$

  is a sequence of tuples of elements in the component groups. Equivalently, for each $i \in [n]$ there exists a function

$$F_i : \mathbb{Z}_N \to \mathbb{Z}_{p_i}^{t_i}, \qquad F_i(x) = (f_{i,1}(x), \ldots, f_{i,t_i}(x)),$$

  and there exists a function $f : \mathbb{Z}_N \to \{0,1\}^*$ such that $f(x) = \mathsf{L}_{x\text{-}\mathsf{str}}$.

- ***Group operation.*** *For all $x, y \in \mathbb{Z}_N$, the algorithm Add takes $\mathsf{L}(x)$ and $\mathsf{L}(y)$ as input and outputs*

$$\mathsf{Add}(\mathsf{L}(x), \mathsf{L}(y)) = \mathsf{L}(x + y \bmod N).$$

- ***Equality test.*** *For all $x, y \in \mathbb{Z}_N$, the algorithm Eq outputs 1 if and only if $\mathsf{L}(x) = \mathsf{L}(y)$. Equivalently, $\mathsf{Eq}(\mathsf{L}(x), \mathsf{L}(y)) = 1$ iff both of the following hold:*

(i) $\mathsf{L}_{x\text{-str}} = \mathsf{L}_{y\text{-str}}$*; and*

(ii) for every $i \in [n]$ *and every* $j \in [t_i]$*, the equality test of* $\mathcal{O}_i$ *accepts the corresponding pair of handles.*

- ***Validity test.*** *For every candidate label* L^**, the algorithm* VT *outputs* 1 *if and only if there exists some* $x \in \mathbb{Z}_N$ *such that* $\mathsf{L}^* = \mathsf{L}(x)$*.*

We assume throughout that the combiner satisfies perfect correctness. Concretely, for all valid inputs the labeling and group-operation algorithms behave exactly as specified above, the validity test accepts exactly the valid labels, and in particular the labeling map is injective, so for distinct $x, y \in \mathbb{Z}_N$ *we have* $\mathsf{L}(x) \neq \mathsf{L}(y)$*.*

The definition above makes explicit the structural restriction that will matter later. Because the combiner is generic, the representation $\mathsf{L}(x)$ is entirely determined by x. Thus the combined group is encoded by a visible string part $f(x)$ together with hidden algebraic data $F_1(x), \ldots, F_n(x)$ distributed across the component groups. The lower bounds later will exploit exactly this split between what is explicitly visible and what can be recovered from corrupted coordinates.

3.2 Robust generic combiners and corrupted coordinates

We now formalize robustness for generic group combiners. Following the usual black-box paradigm, we require both the construction and the proof of robustness to stay within the generic model. In the main text we focus on DLog and DDH, since these are the two assumptions that cleanly capture the search-versus-decision contrast underlying our results.

Definition 3.2 ((k,n)-Robust Generic Combiner for DLog or DDH)**.** *Let* $\mathcal{O}_1, \ldots, \mathcal{O}_n$ *be* n *independent generic group oracles. A* (k,n)*-robust generic combiner* C *for DLog-secure groups (resp. DDH-secure groups) is a generic algorithm implementing Definition 3.1 and satisfying the following two conditions:*

- ***Generic implementation.*** *The combiner* C *is an efficient generic algorithm with oracle access to* $\mathcal{O}_1, \ldots, \mathcal{O}_n$*.*
- ***Generic proof of robustness.*** *There exists an efficient generic reduction* $\mathcal{R}$ *such that, whenever a generic adversary* $\mathcal{A}$ *breaks the DLog security (resp. DDH security) of the combined construction, the reduction* $\mathcal{R}^{\mathcal{A}}$ *succeeds in breaking the same underlying hardness assumption with respect to at least* $n - k + 1$ *of the component groups.*

At this point, however, one subtlety becomes essential. In an ordinary generic-group hardness definition, an efficient adversary is restricted to polynomially many queries. That is the right notion when one studies a single uncorrupted generic group. For robustness, it is too weak. Indeed, the statement that "the combiner remains secure as long as at least k components are secure" means that the remaining $n-k$ components should be viewed as potentially *fully broken*. To model this inside the generic world, we must allow the adversary essentially unrestricted generic access on the corrupted coordinates.

Definition 3.3 (Efficient Generic Adversary with respect to I)**.** *Let* $\mathcal{O}_1, \ldots, \mathcal{O}_n$ *be generic group oracles, and let* $I \subseteq [n]$*. A generic algorithm* $\mathcal{A}_I$ *is called an* efficient generic adversary with respect to I *if the following conditions hold:*

- *for every* $i \in I$*, the adversary* $\mathcal{A}_I$ *may issue an arbitrary (possibly superpolynomial) number of queries to* $\mathcal{O}_i$*; and*

- *for every $j \notin I$, the adversary $\mathcal{A}_I$ issues at most polynomially many queries to $\mathcal{O}_j$.*

This is the right generic analogue of corruption. The set I indexes the component groups that are treated as broken: on those coordinates the adversary can extract essentially all algebraic information that is visible through the generic interface, while on the remaining coordinates it still faces the usual efficient-query restriction.

Suppose there exists an efficient generic adversary $\mathcal{A}_I$ with respect to some set $I \subseteq [n]$ with $|I| \leq n - k$ that breaks the DLog (resp. DDH) security of a candidate generic combiner $\mathsf{C}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$. Then C cannot be (k, n)-robust.

To see this, recall that robustness requires an efficient generic reduction $\mathcal{R}$ that transforms any successful adversary against the combiner into successful attacks on at least $n - k + 1$ of the component groups. Suppose such a reduction existed. Then, by combining $\mathcal{R}$ with $\mathcal{A}_I$, we would obtain a generic algorithm $\mathcal{B}^{\mathcal{A}_I,\mathcal{R}^{\mathcal{A}_I}}$ that breaks the corresponding hardness assumption in at least $n - k + 1$ of $\mathcal{O}_1, \ldots, \mathcal{O}_n$. However, $\mathcal{B}$ inherits unbounded access only to the oracles indexed by I, and thus has unbounded access to at most $n - k$ component groups. Consequently, among the $n - k + 1$ groups that $\mathcal{B}$ is required to break, at least one must be queried only polynomially many times, contradicting the standard generic hardness of that component group. Hence no such reduction $\mathcal{R}$ can exist.

Accordingly, in order to prove impossibility, it suffices to construct an efficient generic adversary $\mathcal{A}_I$ with $|I| \leq n - k$ that breaks the relevant security notion of the combiner. This is the form that we use in the sequel. It is tailored to the robust-combiner setting: the adversary is polynomially bounded on the still-secure coordinates and unrestricted only on the corrupted ones. Importantly, we do *not* claim here that this condition is equivalent to Definition 3.2; for the lower bounds, only this forward implication is needed.

For intuition, one may view this argument as describing the corruption game that underlies our lower bounds: the challenger gives the adversary a combined-group DLog or DDH challenge and then allows unrestricted generic interaction with the corrupted coordinates indexed by I. We prefer the "with respect to I" formulation because it cleanly captures the sufficient condition for impossibility that will be used later, rather than a separate equivalent definition of robustness.

4 Impossibility of robust generic combiner for DDH

In this section, we prove the impossibility of robust generic combiners for DDH-secure groups. Concretely, we construct an efficient generic adversary $\mathcal{A}_I$ with respect to a set $I \subseteq [n]$ satisfying $|I| = n - k$ such that, for any generic combiner $\mathcal{C}$, the adversary $\mathcal{A}_I$ breaks the DDH security of $\mathcal{C}$.

Moreover, it suffices to rule out the case of $(n - 1, n)$-robustness. Indeed, for any $k \leq n - 1$, a (k, n)-robust combiner is in particular also an $(n - 1, n)$-robust combiner. Hence, if no $(n - 1, n)$-robust generic combiner exists, then no (k, n)-robust generic combiner can exist either. For clarity of exposition, we begin with the special case $n = 2$, and then extend the argument to general n.

In a nutshell, our adversarial strategy consists of three main steps:

1. We first identify a suitable notion of *partial information* associated with a combined-group element. In particular, for a Diffie–Hellman label $\mathsf{L}(xy)$, we define a quantity $\mathsf{P}(xy)$ that captures a certain partial view of its representation.

2. Next, we prove that this partial information $\mathsf{P}(xy)$ already suffices to distinguish $\mathsf{L}(xy)$ from a random label $\mathsf{L}(z)$ with high probability.

3. We then design a generic adversary $\mathcal{A}_I$ that, given $\mathsf{L}(x)$ and $\mathsf{L}(y)$, efficiently recovers $\mathsf{P}(xy)$. Combining this extraction procedure with the distinguishing property from the second step yields the desired attack on DDH security.

4.1 Partial information

To implement the first step of the above strategy, we now formalize the notion of partial information associated with a combined-group element. Throughout this part, we assume without loss of generality that the combiner C makes at most q queries to the oracles $\mathcal{O}_1$ and $\mathcal{O}_2$, where q is polynomial in the security parameter. Let $\mathsf{L}(x) = \big(\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}_1}, \mathsf{L}_{x\text{-}\mathsf{G}_2}\big)$ denote the representation of a combined-group element. Concretely,

$$\mathsf{L}_{x\text{-}\mathsf{G}_1} = \big(\hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}\big) \qquad \text{and} \qquad \mathsf{L}_{x\text{-}\mathsf{G}_2} = \big(\hat{g}_2^{x_{2,1}}, \ldots, \hat{g}_2^{x_{2,t_2}}\big),$$

where the two tuples consist of group elements in $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively.

We next define the *partial information* of $\mathsf{L}(x)$ with respect to a set $I \in \big\{\{1\},\{2\}\big\}$, denoted $\mathsf{P}_I(x)$, to consist of the bit-string part $\mathsf{L}_{x\text{-}\mathsf{str}}$ together with the exponent tuple contributed by the oracle indexed by I. Thus, for $I = \{1\}$,

$$\mathsf{P}_I(x) = \big(\mathsf{L}_{x\text{-}\mathsf{str}}, (x_{1,1}, \ldots, x_{1,t_1})\big),$$

and for $I = \{2\}$,

$$\mathsf{P}_I(x) = \big(\mathsf{L}_{x\text{-}\mathsf{str}}, (x_{2,1}, \ldots, x_{2,t_2})\big).$$

4.2 Breaking DDH-security via partial information

In this section, we show how the partial information defined above can be exploited to break the DDH security of any generic combiner. This yields the following theorem.

Theorem 4.1. *Let λ be the security parameter, and let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two generic group oracles of orders p_1 and p_2, respectively. For any generic combiner $\mathsf{C}^{\mathcal{O}_1,\mathcal{O}_2}$, there exists an efficient generic adversary $\mathcal{A}_I$ with respect to some $I \in \{\{1\},\{2\}\}$ such that $\mathcal{A}_I$ breaks the DDH security of C with noticeable probability in λ.*

We now explain why the partial information $\mathsf{P}_I(xy)$ is useful for distinguishing $\mathsf{L}(xy)$ from $\mathsf{L}(z)$. We begin with the simple observation that, given the representation of a combined-group element, say $\mathsf{L}(x)$, the corresponding partial information $\mathsf{P}_I(x)$ can be efficiently recovered by a generic adversary with respect to I.

To make this precise, let $\mathcal{A}_{I,\mathsf{Dec}}(\cdot)$ denote the procedure that, on input $\mathsf{L}(x)$, outputs $\mathsf{P}_I(x)$. By construction, $\mathcal{A}_{I,\mathsf{Dec}}$ is an efficient generic algorithm with respect to I. Indeed, without loss of generality, consider the case $I = \{1\}$. Since $\mathcal{A}_{I,\mathsf{Dec}}$ is allowed to make an unbounded number of queries to the oracle $\mathcal{O}_1$, and since the representation $\mathsf{L}(x)$ explicitly contains the tuple $\mathsf{L}_{x\text{-}\mathsf{G}_1} = (\hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}})$, the algorithm can recover the exponents $(x_{1,1}, \ldots, x_{1,t_1})$ by solving the corresponding discrete logarithms in $\mathcal{O}_1$ using its unlimited oracle access. Together with the bit-string component $\mathsf{L}_{x\text{-}\mathsf{str}}$, this determines $\mathsf{P}_I(x)$. The case $I = \{2\}$ is analogous.

We next bound the probability that the partial information of a uniformly random element collides with that of a fixed element. The following lemma shows that, for at least one choice of $I \in \{\{1\},\{2\}\}$, $\mathsf{P}_I(x)$ retains enough information to rule out large collision probability.

Lemma 4.2. *There exists $I \in \big\{\{1\},\{2\}\big\}$ such that for every $x \in \mathbb{Z}_N$, $\Pr_{z \xleftarrow{\$} \mathbb{Z}_N}\big[\mathsf{P}_I(z) = \mathsf{P}_I(x)\big] \leq \frac{3}{4}$.*

Adversary $\mathcal{A}_{I,\mathsf{DDH}}^{\mathcal{O}_1,\mathcal{O}_2}(\mathsf{L}(1),\mathsf{L}(x),\mathsf{L}(y),\mathsf{L}_T)$

$\mathcal{A}_{I,\mathsf{DDH}}^{\mathcal{O}_1,\mathcal{O}_2}(\mathsf{L}(1),\mathsf{L}(x),\mathsf{L}(y),\mathsf{L}_T)$:

The challenge representation L_T is either $\mathsf{L}(xy)$ or $\mathsf{L}(z)$, where $z \xleftarrow{\$} \mathbb{Z}_N$.

Compute $\mathsf{P}_T \leftarrow \mathcal{A}_{I,\mathsf{Dec}}(\mathsf{L}_T)$.

Compute $\mathsf{P}_I(xy) \leftarrow \mathcal{A}_{I,\mathsf{Ext}}(\mathsf{L}(x),\mathsf{L}(y))$.

If $\mathsf{P}_T = \mathsf{P}_I(xy)$: **Return** 1.

Otherwise: **Return** 0.

Figure 3: The description of $\mathcal{A}_{I,\mathsf{DDH}}$ that breaks the DDH security of the combiner.

Proof. Suppose toward contradiction that the claim is false. Then, for each $I \in \{\{1\},\{2\}\}$, there exists some $x_I \in \mathbb{Z}_N$ such that $\Pr_{z \xleftarrow{\$} \mathbb{Z}_N}\big[\mathsf{P}_I(z) = \mathsf{P}_I(x_I)\big] > \frac{3}{4}$.

For $i \in \{1,2\}$, define

$$S_i := \big\{z \in \mathbb{Z}_N : \mathsf{P}_{\{i\}}(z) = \mathsf{P}_{\{i\}}(x_i)\big\}.$$

Then $|S_1| > \frac{3N}{4}$ and $|S_2| > \frac{3N}{4}$. Hence, $|S_1 \cap S_2| \geq |S_1| + |S_2| - N > \frac{N}{2}$.

In particular, there exist two distinct values $z, z' \in S_1 \cap S_2$. By the definition of S_1 and S_2, $\mathsf{P}_{\{1\}}(z) = \mathsf{P}_{\{1\}}(z')$ and $\mathsf{P}_{\{2\}}(z) = \mathsf{P}_{\{2\}}(z')$. Together, these two equalities determine both the bit-string part and all exponent tuples of the representation. Therefore, $\mathsf{L}(z) = \mathsf{L}(z')$, contradicting the injectivity of the labeling map.

□

We now show that an algorithm capable of extracting $\mathsf{P}_I(xy)$ from $\mathsf{L}(x)$ and $\mathsf{L}(y)$ immediately yields an attack on the DDH security of the combiner. Specifically, suppose there exists an algorithm $\mathcal{A}_{I,\mathsf{Ext}}$ such that, for every choice of I,

$$\Pr\big[\mathcal{A}_{I,\mathsf{Ext}}(\mathsf{L}(x),\mathsf{L}(y)) = \mathsf{P}_I(xy)\big] \geq 1 - \frac{1}{Q},$$

where Q is a sufficiently large polynomial. Then one can construct a generic adversary $\mathcal{A}_{I,\mathsf{DDH}}$, described in Figure 3, that breaks the DDH security of the combiner.

If the challenge term L_T is equal to $\mathsf{L}(xy)$, then $\mathcal{A}_{I,\mathsf{DDH}}$ outputs 1 with probability at least $1 - \frac{1}{Q}$. By contrast, if $\mathsf{L}_T = \mathsf{L}(z)$ for uniformly random $z \xleftarrow{\$} \mathbb{Z}_N$, then by the preceding lemma the probability that $\mathcal{A}_{I,\mathsf{DDH}}$ outputs 1 is at most $\frac{3}{4}$. It follows that the distinguishing advantage of $\mathcal{A}_{I,\mathsf{DDH}}$ is at least

$$\left(1 - \frac{1}{Q}\right) - \frac{3}{4} = \frac{1}{4} - \frac{1}{Q}.$$

It therefore remains to construct such an extractor $\mathcal{A}_{I,\mathsf{Ext}}$. This is the core technical step in our analysis.

4.3 Extracting partial information

In this section, we show how to extract the partial information $\mathsf{P}_I(xy)$ from $\mathsf{L}(x)$ and $\mathsf{L}(y)$. Without loss of generality, we set $I = \{1\}$; that is, the extractor $\mathcal{A}_{I,\mathsf{Ext}}$ is allowed to make an unbounded number of queries to $\mathcal{O}_1$, while making only polynomially many queries to $\mathcal{O}_2$. As shown above, given $\mathsf{L}(x)$, $\mathcal{A}_{I,\mathsf{Dec}}$ can compute $\mathsf{P}_I(x)$.

Moreover, since C is generic, $\mathsf{P}_I(x)$ is determined solely by the exponent $x \in \mathbb{Z}_N$. In particular, neither the bit-string component $\mathsf{L}_{x\text{-}\mathsf{str}}$ nor the exponent tuple defining the partial information depends on any concrete encoding of the oracles. Consequently, once the combiner C is fixed, the value $\mathsf{P}_I(x)$ can be computed from x alone, without making any queries to $\mathcal{O}_1$ or $\mathcal{O}_2$. It follows that, given $\mathsf{L}(x)$ and $\mathsf{L}(y)$, adversary can compute the sets

$$S_x := \big\{\tilde{x} \in \mathbb{Z}_N \mid \mathsf{P}_I(\tilde{x}) = \mathsf{P}_I(x)\big\} \qquad \text{and} \qquad S_y := \big\{\tilde{y} \in \mathbb{Z}_N \mid \mathsf{P}_I(\tilde{y}) = \mathsf{P}_I(y)\big\}.$$

Motivated by the DDH distinguisher for the direct-product construction, consider the following heuristic: if, for every pair $(\tilde{x}, \tilde{y}) \in S_x \times S_y$, one has $\mathsf{P}_I(\tilde{x}\tilde{y}) = \mathsf{P}_I(xy)$—or even if this equality holds for a large majority of such pairs—then extracting $\mathsf{P}_I(xy)$ becomes straightforward. Indeed, given $\mathsf{L}(x)$ and $\mathsf{L}(y)$, the adversary first computes the corresponding sets S_x and S_y. It then evaluates $\mathsf{P}_I(\tilde{x}\tilde{y})$ for each pair $(\tilde{x}, \tilde{y}) \in S_x \times S_y$, and outputs the majority value.

Unfortunately, this heuristic need not hold for a general generic construction. We next describe the obstacle and then show how to resolve it. To this end, given a representation

$$\mathsf{L}(x) = \big(\mathsf{L}_{x\text{-}\mathsf{str}}, (\hat{g}_1^{x_{1,1}}, \ldots, \hat{g}_1^{x_{1,t_1}}), (\hat{g}_2^{x_{2,1}}, \ldots, \hat{g}_2^{x_{2,t_2}})\big),$$

and an integer $y \in \mathbb{Z}_N$, we define $\mathsf{Scale}(\mathsf{L}(x), y)$ to be the procedure that computes the y-fold sum of $\mathsf{L}(x)$ in the combined group[4]. By correctness, its output is precisely $\mathsf{L}(xy)$.

During the execution of $\mathsf{Scale}(\mathsf{L}(x), y)$, each equality-test query issued to $\mathcal{O}_1$ induces a linear constraint on the exponents appearing in $(\hat{g}_1, \mathsf{L}_{x\text{-}\mathsf{G}_1})$. Concretely, such a query can be represented by a coefficient vector $(\alpha_0, \ldots, \alpha_{t_1}) \in \mathbb{Z}^{t_1+1}$. The oracle $\mathcal{O}_1$ returns a positive answer precisely when

$$\alpha_0 + \sum_{i=1}^{t_1} \alpha_i x_{1,i} = 0,$$

and returns a negative answer otherwise, namely when

$$\alpha_0 + \sum_{i=1}^{t_1} \alpha_i x_{1,i} \neq 0.$$

Accordingly, we write

$$(\mathcal{L}_1, \mathcal{T}_1) \xleftarrow{\mathcal{O}_1} \mathsf{Scale}\big(\mathsf{L}(x), y\big)$$

for the collection of all such queries arising during the execution, where $\mathcal{L}_1$ consists of those coefficient vectors that receive a positive answer, and $\mathcal{T}_1$ consists of those that receive a negative answer. Analogously, we define

$$(\mathcal{L}_2, \mathcal{T}_2) \xleftarrow{\mathcal{O}_2} \mathsf{Scale}\big(\mathsf{L}(x), y\big).$$

We now explain why the heuristic described above may fail. Let $\tilde{x} \in S_x$ and $r \in S_y$. Consider the two executions $\mathsf{Scale}(\mathsf{L}(x), r)$ and $\mathsf{Scale}(\mathsf{L}(\tilde{x}), r)$, which are intended to output $\mathsf{L}(xr)$ and $\mathsf{L}(\tilde{x}r)$,

[4]This can be implemented using only $O(\log y)$ invocations of Add via repeated doubling.

respectively. Write $\big((\mathcal{L}_1,\mathcal{T}_1),(\mathcal{L}_2,\mathcal{T}_2)\big)$ and $\big((\widetilde{\mathcal{L}}_1,\widetilde{\mathcal{T}}_1),(\widetilde{\mathcal{L}}_2,\widetilde{\mathcal{T}}_2)\big)$ for the corresponding collections of equality-test queries generated during these two executions, and let $(\mathsf{res}_1,\mathsf{res}_2)$ and $(\widetilde{\mathsf{res}}_1,\widetilde{\mathsf{res}}_2)$ denote the corresponding oracle responses.

Since Scale is generic, it does not exploit any structure of the concrete encodings of the oracles. Its behavior is therefore completely determined by the input representation, the scalar r, and the responses to the equality-test queries issued during the execution. In particular, the bit-string component of the output representation $\mathsf{L}_{xr\text{-}\mathsf{str}}$ (resp. $\mathsf{L}_{\tilde{x}r\text{-}\mathsf{str}}$) depends only on $\mathsf{L}_{x\text{-}\mathsf{str}}$ (resp. $\mathsf{L}_{\tilde{x}\text{-}\mathsf{str}}$), r, and the response vectors $\mathsf{res}_1,\mathsf{res}_2$ (resp. $\widetilde{\mathsf{res}}_1,\widetilde{\mathsf{res}}_2$) returned by $\mathcal{O}_1$ and $\mathcal{O}_2$. Likewise, each group element in $\mathsf{L}_{xr\text{-}\mathsf{G}_1}$ (resp. $\mathsf{L}_{\tilde{x}r\text{-}\mathsf{G}_1}$) is obtained as a linear combination of the elements in $\mathsf{L}_{x\text{-}\mathsf{G}_1}$ (resp. $\mathsf{L}_{\tilde{x}\text{-}\mathsf{G}_1}$), with coefficients determined by $(\mathsf{L}_{x\text{-}\mathsf{str}},r,\mathsf{res}_1,\mathsf{res}_2)$ (resp. $(\mathsf{L}_{\tilde{x}\text{-}\mathsf{str}},r,\widetilde{\mathsf{res}}_1,\widetilde{\mathsf{res}}_2)$).

Observe that the responses to equality-test queries issued to $\mathcal{O}_1$ are identical across the two executions, since $\mathsf{P}_I(x)=\mathsf{P}_I(\tilde{x})$. The situation is different for queries to $\mathcal{O}_2$: their responses may vary between the two executions. This discrepancy may in turn alter either the output bit string or the coefficients of the resulting linear combinations. Consequently, the two executions need not produce outputs with the same partial information; in particular, it may happen that $\mathsf{P}_I(xr)\neq\mathsf{P}_I(\tilde{x}r)$

To overcome this obstacle, we rely on the following simple but crucial observation: if the two executions issue exactly the same equality-test queries and receive exactly the same responses, namely if

$$\big((\mathcal{L}_1,\mathcal{T}_1),(\mathcal{L}_2,\mathcal{T}_2)\big)=\big((\widetilde{\mathcal{L}}_1,\widetilde{\mathcal{T}}_1),(\widetilde{\mathcal{L}}_2,\widetilde{\mathcal{T}}_2)\big)\qquad\text{and}\qquad(\mathsf{res}_1,\mathsf{res}_2)=(\widetilde{\mathsf{res}}_1,\widetilde{\mathsf{res}}_2),$$

then the two executions must produce outputs with the same partial information; that is, $\mathsf{P}_I(xr)=\mathsf{P}_I(\tilde{x}r)$.

This motivates the following strategy. We design a carefully chosen filtering procedure Refine that, roughly speaking, takes $\big((S_x,\mathsf{L}(x)),(S_y,\mathsf{L}(y))\big)$ as input and outputs the corresponding refined sets (S_x^+,S_y^+) satisfying:

- $S_x^+\subseteq S_x$ and $S_y^+\subseteq S_y$;
- $x\in S_x^+$ and $y\in S_y^+$; and
- when $(\tilde{x},\tilde{y})$ is sampled uniformly from $S_x^+\times S_y^+$, both

  $$\mathsf{P}_I(xy)=\mathsf{P}_I(\tilde{x}y)\qquad\text{and}\qquad\mathsf{P}_I(y\tilde{x})=\mathsf{P}_I(\tilde{y}\tilde{x})$$

  hold with high probability.

Consequently, with high probability over $(\tilde{x},\tilde{y})\xleftarrow{\$}S_x^+\times S_y^+$, we have $\mathsf{P}_I(\tilde{x}\tilde{y})=\mathsf{P}_I(xy)$. In particular, the majority value of $\mathsf{P}_I(\tilde{x}\tilde{y})$ over pairs $(\tilde{x},\tilde{y})\in S_x^+\times S_y^+$ is equal to $\mathsf{P}_I(xy)$ with high probability.

Next, we present the formal description of the extracting algorithm $\mathcal{A}_{I,\mathsf{Ext}}$ and the algorithm Refine, depicted in Figure 4. For clarity, we first clarify several notions used in the algorithm. Let $x\in\mathbb{Z}_N$, let $S\subseteq\mathbb{Z}_N$ be a set containing x, and let $S_{\mathsf{base}}\subseteq\mathbb{Z}_N$. For any $r\in S_{\mathsf{base}}$, write

$$(\mathcal{L}_2,\mathcal{T}_2)\xleftarrow{\mathcal{O}_2}\mathsf{Scale}\big(\mathsf{L}(x),r\big)$$

for the collection of queries issued to $\mathcal{O}_2$ during the execution of $\mathsf{Scale}(\mathsf{L}(x),r)$, where $\mathcal{L}_2$ consists of those queries that receive a positive answer and $\mathcal{T}_2$ consists of those that receive a negative

answer. For any $\tilde{x} \in S$, write

$$\mathsf{L}(\tilde{x}) = \big(\mathsf{L}_{\tilde{x}\text{-}\mathsf{str}}, (\hat{g}_1^{\tilde{x}_{1,1}}, \ldots, \hat{g}_1^{\tilde{x}_{1,t_1}}), (\hat{g}_2^{\tilde{x}_{2,1}}, \ldots, \hat{g}_2^{\tilde{x}_{2,t_2}})\big),$$

and let

$$F_1(\tilde{x}) = (\tilde{x}_{1,1}, \ldots, \tilde{x}_{1,t_1}) \qquad \text{and} \qquad F_2(\tilde{x}) = (\tilde{x}_{2,1}, \ldots, \tilde{x}_{2,t_2}).$$

We say that $F_2(\tilde{x})$ is *inconsistent* with $(\mathcal{L}_2, \mathcal{T}_2)$ if either of the following holds:

- $\mathcal{L}_2 \cdot F_2(\tilde{x}) \neq 0$: there exists $(\alpha_0, \ldots, \alpha_{t_2}) \in \mathcal{L}_2$ such that

$$\alpha_0 + \sum_{i=1}^{t_2} \alpha_i \tilde{x}_{2,i} \neq 0;$$

- $\mathcal{T}_2 \cdot F_2(\tilde{x}) = 0$: there exists $(\alpha_0, \ldots, \alpha_{t_2}) \in \mathcal{T}_2$ such that

$$\alpha_0 + \sum_{i=1}^{t_2} \alpha_i \tilde{x}_{2,i} = 0.$$

Equivalently, $F_2(\tilde{x})$ is inconsistent with $(\mathcal{L}_2, \mathcal{T}_2)$ if it violates at least one positive constraint in $\mathcal{L}_2$ or satisfies at least one negative constraint in $\mathcal{T}_2$. Accordingly, we say that $\tilde{x}$ is a *bad candidate with respect to* r if $F_2(\tilde{x})$ is inconsistent with $(\mathcal{L}_2, \mathcal{T}_2)$.

Before showing that $\mathcal{A}_{I,\mathsf{Ext}}$ yields an attack on DDH security, we first verify that it is indeed an efficient generic algorithm with respect to I. This is immediate for Phases 1 and 3, whose computations require no access to $\mathcal{O}_2$ and can be carried out entirely using the unrestricted access to $\mathcal{O}_1$. It therefore remains to analyze Phase 2, namely the procedure Refine. In each iteration, Refine invokes Scale at most q^3 times. Consequently, to prove that $\mathcal{A}_{I,\mathsf{Ext}}$ is efficient with respect to I, it suffices to show that Refine terminates after only polynomially many iterations. The following lemma gives the required bound.

Lemma 4.3. *Let λ be the security parameter, let $x \in \mathbb{Z}_N$, and let q be a polynomial such that $q(\lambda) > \log N$. Let $S_{\mathsf{refine}} \subseteq \mathbb{Z}_N$ be any set with $x \in S_{\mathsf{refine}}$, and let $S_{\mathsf{base}} \subseteq \mathbb{Z}_N$ be arbitrary. Then* $\mathsf{Refine}(\mathsf{L}(x), S_{\mathsf{refine}}, S_{\mathsf{base}})$ *terminates after at most $q^2 + 1$ iterations.*

Proof. By the description of Refine, we have the following immediate observation: if Refine terminates after exactly $t \geq 1$ iterations and outputs the set S^+_{refine}, then the Testing phase must have failed at the end of each of the first $t - 1$ iterations.

Moreover, recall that failure of the Testing Phase means that, among the q^3 sampled values r, there exists at least one value r_i that identifies at least a $1/q$ fraction of the bad candidates $\tilde{x}$ in the current set. Accordingly, in the subsequent Filtering Phase, all such candidates are removed, and hence each unsuccessful iteration eliminates at least a $1/q$ fraction of the current set.

Therefore, if the Testing Phase fails in each of the first $t - 1$ iterations, then every such iteration shrinks the current set by a factor of at least $1 - 1/q$. It follows that the final output set S^+_{refine} satisfies

$$\left|S^+_{\mathsf{refine}}\right| \leq \left(1 - \frac{1}{q}\right)^{t-1} \left|S_{\mathsf{refine}}\right|.$$

Adversary $\mathcal{A}_{I,\mathsf{Ext}}^{\mathcal{O}_1,\mathcal{O}_2}\big(\mathsf{L}(x),\mathsf{L}(y)\big)$

$\underline{\mathcal{A}_{I,\mathsf{Ext}}^{\mathcal{O}_1,\mathcal{O}_2}\big(\mathsf{L}(x),\mathsf{L}(y)\big)}$:

Let $\mathsf{L}(x) = \big(\mathsf{L}_{x\text{-str}}, \mathsf{L}_{x\text{-G}_1}, \mathsf{L}_{x\text{-G}_2}\big)$ and $\mathsf{L}(y) = \big(\mathsf{L}_{y\text{-str}}, \mathsf{L}_{y\text{-G}_1}, \mathsf{L}_{y\text{-G}_2}\big)$;

$S_{\mathsf{ext}} \leftarrow \emptyset$;

Phase 1: Initialization

$\mathsf{P}_I(x) \leftarrow \mathcal{A}_{I,\mathsf{Dec}}^{\mathcal{O}_1}(\mathsf{L}(x))$; $\mathsf{P}_I(y) \leftarrow \mathcal{A}_{I,\mathsf{Dec}}^{\mathcal{O}_1}(\mathsf{L}(y))$;

Compute $S_x := \big\{\tilde{x} \in \mathbb{Z}_N \mid \mathsf{P}_I(\tilde{x}) = \mathsf{P}_I(x)\big\}$;

Compute $S_y := \big\{\tilde{y} \in \mathbb{Z}_N \mid \mathsf{P}_I(\tilde{y}) = \mathsf{P}_I(y)\big\}$;

Phase 2: Refining S_x and S_y

Compute $S_x^+ \leftarrow \mathsf{Refine}(\mathsf{L}(x), S_x, \mathbb{Z}_N)$ and $S_y^+ \leftarrow \mathsf{Refine}(\mathsf{L}(y), S_y, S_x^+)$;

Phase 3: Extracting $\mathsf{P}(xy)$

For each $\tilde{x} \in S_x^+$ and $\tilde{y} \in S_y^+$:

Compute $\mathsf{P}_I(\tilde{x}\tilde{y})$;

Update $S_{\mathsf{ext}} \leftarrow S_{\mathsf{ext}} \cup \mathsf{P}_I(\tilde{x}\tilde{y})$;

Return the majority value of S_{ext}.

$\underline{\mathsf{Refine}\big(\mathsf{Elem}, S_{\mathsf{refine}}, S_{\mathsf{base}}\big)}$:

While **True** do:

Sample $r_1, \ldots, r_{q^3} \xleftarrow{\$} S_{\mathsf{base}}$;

For $i = 1$ to q^3: `identifying bad candidates in` S_{refine} `via each` r_i

Compute $(\mathcal{L}_{2,i}, \mathcal{T}_{2,i}) \xleftarrow{\mathcal{O}_2} \mathsf{Scale}\big(\mathsf{Elem}, r_i\big)$; `collecting queries and responses`

Compute $S_{r_i} := \{\tilde{x} \in S_{\mathsf{refine}} \mid (\mathcal{L}_{2,i} \cdot F_2(\tilde{x}) \neq 0) \vee (\mathcal{T}_{2,i} \cdot F_2(\tilde{x}) = 0)\}$;
$\tilde{x}$ `is a bad candidate if` $F_2(\tilde{x})$ `is inconsistent with query-response behavior`

Testing Phase:

If $|S_{r_i}| < |S_{\mathsf{refine}}|/q$ for all $i \in [q^3]$: **Return** S_{refine};

Filtering Phase: `removing all known bad candidates`

Update $S_{\mathsf{refine}} := S_{\mathsf{refine}} \setminus \cup_{i\in[q^3]} S_{r_i}$.

Figure 4: The extractor $\mathcal{A}_{I,\mathsf{Ext}}$ recovering the partial information $\mathsf{P}_I(xy)$ through refinement procedure Refine.

Now suppose that $t \geq q^2 + 1$. Then

$$\left|S^+_{\mathsf{refine}}\right| \leq \left(1 - \frac{1}{q}\right)^{q^2} |S_{\mathsf{refine}}| \leq \frac{N}{e^q} < 1,$$

which is impossible. Indeed, we must have $|S^+_{\mathsf{refine}}| \geq 1$, since x can never been removed. □

We next justify why the refined sets (S^+_x, S^+_y) suffice for our purposes. Specifically, we show that when a pair $(\tilde{x}, \tilde{y})$ is sampled uniformly from $S^+_x \times S^+_y$, the probability that $\mathsf{P}_I(\tilde{x}\tilde{y}) = \mathsf{P}_I(xy)$ is high. To establish this claim, we introduce several auxiliary notions.

Let $x \in \mathbb{Z}_N$, let $S \subseteq \mathbb{Z}_N$ be a set containing x, and let $S_{\mathsf{base}} \subseteq \mathbb{Z}_N$. For any $\tilde{x} \in S$ and $r \in S_{\mathsf{base}}$, we say that r is *bad for* S if

$$\Pr_{\tilde{x} \xleftarrow{\$} S}\left[\tilde{x} \text{ is a bad candidate with respect to } r\right] \geq \frac{1}{q}.$$

Furthermore, We say that S is *bad over* S_{base} if

$$\Pr_{r \xleftarrow{\$} S_{\mathsf{base}}}\left[r \text{ is bad for } S\right] > \frac{1}{q^2},$$

and *good over* S_{base} otherwise; that is, if

$$\Pr_{r \xleftarrow{\$} S_{\mathsf{base}}}\left[r \text{ is bad for } S\right] \leq \frac{1}{q^2}.$$

We next demostrate the power of Refine. The following lemma shows that , with overwhelming probability, the refined set $S^+ \leftarrow \mathsf{Refine}(\mathsf{L}(x), S, S_{\mathsf{base}})$ is good over S_{base}.

Lemma 4.4. *Let λ be the security parameter, let $x \in \mathbb{Z}_N$, and let q be a polynomial satisfying $q(\lambda) > \log N$. Let $S \subseteq \mathbb{Z}_N$ be any set such that $x \in S$, and let $S_{\mathsf{base}} \subseteq \mathbb{Z}_N$ be arbitrary. If $S^+ \leftarrow \mathsf{Refine}(\mathsf{L}(x), S, S_{\mathsf{base}})$, then*

$$\Pr\left[S^+ \textit{ is bad over } S_{\mathsf{base}}\right] \leq \frac{q^2 + 1}{2^q},$$

where the probability is taken over the internal randomness of Refine.

Proof. For clarity, let $\mathcal{D}_{\mathsf{Refine}}$ denote the output distribution of the procedure $\mathsf{Refine}(\mathsf{L}(x), S, S_{\mathsf{base}})$. That is,

$$\mathcal{D}_{\mathsf{Refine}} = \{(S^+_1, p_1), \ldots, (S^+_T, p_T)\},$$

where $\mathsf{Refine}(\mathsf{L}(x), S, S_{\mathsf{base}})$ outputs S^+_i with probability p_i.

Without loss of generality, assume that $S^+_1, \ldots, S^+_t$ are precisely those outputs that are bad over S_{base}. It therefore suffices to show that

$$p_1 + \cdots + p_t \leq \frac{q^2 + 1}{2^q}.$$

By Lemma 4.3, the procedure Refine terminates after at most $q^2 + 1$ iterations. For each output S^+_i and each iteration index $j \in [q^2 + 1]$, let $p_{i,j}$ denote the probability that Refine outputs S^+_i

exactly at the end of the j-th iteration. Then

$$p_i = \sum_{j=1}^{q^2+1} p_{i,j}.$$

Hence, it is enough to prove that for every $j \in [q^2+1]$,

$$p_{1,j} + \cdots + p_{t,j} \leq \frac{1}{2^q}.$$

Indeed, the left-hand side is exactly the probability that Refine outputs a bad set at the end of the j-th iteration.

We begin with the first iteration, where the current set is S. In this iteration, Refine either outputs S immediately or proceeds to the next iteration. If S is good over S_{base}, then the probability of outputting a bad set in the first iteration is clearly 0. Thus, only the case where S is bad over S_{base} needs to be considered.

Let w_0^* denote the probability that Refine outputs S in the first iteration. By definition of Refine, the algorithm samples q^3 values $r_1, \ldots, r_{q^3}$ independently and uniformly from S_{base}. It outputs S precisely when all sampled values are good for S. Since S is bad over S_{base}, we have

$$\Pr_{r \xleftarrow{\$} S_{\mathsf{base}}} [r \text{ is good for } S] \leq 1 - \frac{1}{q^2}.$$

Therefore,

$$p_{1,1} + \cdots + p_{t,1} = w_0^* \leq \left(1 - \frac{1}{q^2}\right)^{q^3} \leq e^{-q} < \frac{1}{2^q}.$$

We now turn to the second iteration. After the first iteration, the current set becomes a random variable; let

$$\mathcal{D}_1 = \{(S, w_0^*), (S_{1,1}, w_{1,1}), \ldots, (S_{1,z_1}, w_{1,z_1})\}$$

denote its distribution, where the probabilities are taken over the randomness used in the first iteration. Without loss of generality, assume that among these sets, $S_{1,1}, \ldots, S_{1,t_1}$ are bad over S_{base}.

For each bad set $S_{1,j}$, let $w_{1,j}^*$ denote the probability that Refine outputs $S_{1,j}$ in the second iteration. Applying the same argument as above, conditioned on reaching $S_{1,j}$, the probability that Refine outputs it in that iteration is at most 2^{-q}. Hence

$$w_{1,j}^* \leq \frac{1}{2^q}\, w_{1,j}.$$

Summing over all bad sets in $\mathcal{D}_1$, we obtain

$$p_{1,2} + \cdots + p_{t,2} = w_{1,1}^* + \cdots + w_{1,t_1}^* \leq \frac{1}{2^q}(w_{1,1} + \cdots + w_{1,t_1}) \leq \frac{1}{2^q}.$$

The same argument applies inductively to every subsequent iteration, as illustrated in Figure 5. Namely, if

$$\mathcal{D}_i = \{(S_{i,1}, w_{i,1}), \ldots, (S_{i,z_i}, w_{i,z_i})\}$$

denotes the distribution of the current set at the beginning of iteration $i+1$, and if $S_{i,1}, \ldots, S_{i,t_i}$

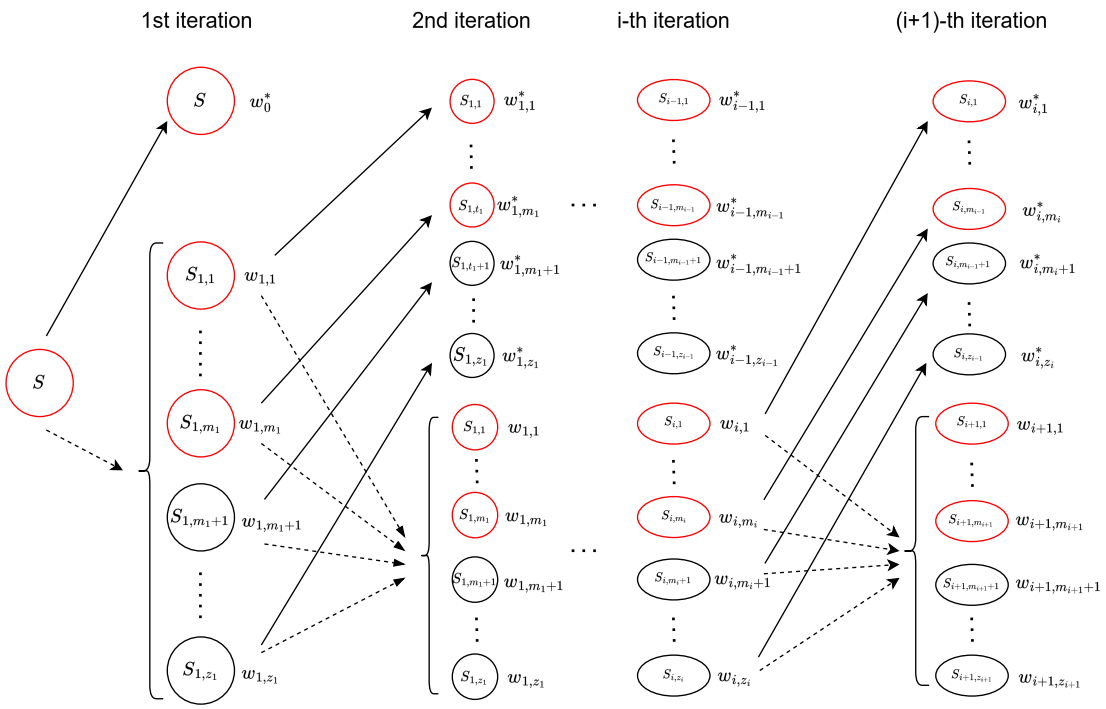

Figure 5: A schematic view of the refinement process across successive iterations and the probability mass of bad outputs.

are the bad sets in this distribution, then

$$p_{1,i+1} + \cdots + p_{t,i+1} \leq \frac{1}{2^q}(w_{i,1} + \cdots + w_{i,t_i}) \leq \frac{1}{2^q}.$$

Thus, for every iteration index $j \in [q^2 + 1]$,

$$p_{1,j} + \cdots + p_{t,j} \leq \frac{1}{2^q}.$$

Summing over all at most $q^2 + 1$ iterations yields

$$p_1 + \cdots + p_t \leq \frac{q^2 + 1}{2^q},$$

which proves the lemma.

□

We are now ready to analyze the success probability of $\mathcal{A}_{I,\mathsf{DDH}}$. Let (S_x^+, S_y^+) denote the refined sets produced by the procedure, and let P^* be the output of $\mathcal{A}_{I,\mathsf{DDH}}(\mathsf{L}(x), \mathsf{L}(y))$. Consider the following two events:

- E_1: y is good for S_x^+;
- E_2: S_y^+ is good over S_x^+.

We claim that, whenever both E_1 and E_2 occur, the output satisfies

$$\mathsf{P}^* = \mathsf{P}(xy).$$

Indeed, if E_1 occurs, then by definition at least a $(1 - 1/q)$ fraction of elements $\tilde{x} \in S_x^+$ satisfy

$$\mathsf{P}(\tilde{x}y) = \mathsf{P}(xy).$$

Now suppose in addition that E_2 occurs. Since S_y^+ is good over S_x^+, at most a $1/q^2$ fraction of elements $\tilde{x} \in S_x^+$ are bad for S_y^+. It follows that at least a

$$1 - \frac{1}{q} - \frac{1}{q^2} \geq 1 - \frac{2}{q}$$

fraction of elements $\tilde{x} \in S_x^+$ simultaneously satisfy both

$$\mathsf{P}(\tilde{x}y) = \mathsf{P}(xy) \qquad \text{and} \qquad \tilde{x} \text{ is good for } S_y^+.$$

Fix any such $\tilde{x}$. Since $\tilde{x}$ is good for S_y^+, again by definition at least a $(1 - 1/q)$ fraction of elements $\tilde{y} \in S_y^+$ satisfy

$$\mathsf{P}(\tilde{x}\tilde{y}) = \mathsf{P}(\tilde{x}y).$$

Combining the two bounds, we conclude that at least a

$$\left(1 - \frac{2}{q}\right)\left(1 - \frac{1}{q}\right) > \frac{1}{2}$$

fraction of pairs $(\tilde{x}, \tilde{y}) \in S_x^+ \times S_y^+$ satisfy

$$\mathsf{P}(\tilde{x}\tilde{y}) = \mathsf{P}(xy).$$

Therefore, the majority value of $\mathsf{P}(\tilde{x}\tilde{y})$ over pairs in $S_x^+ \times S_y^+$ is exactly $\mathsf{P}(xy)$, and hence $\mathsf{P}^* = \mathsf{P}(xy)$. It remains to bound the probability of the event $\mathsf{E}_1 \wedge \mathsf{E}_2$, where the probability is taken over the random choice of x, y and the internal randomness of $\mathcal{A}_{I,\mathsf{Ext}}$. To this end, we decompose the probability as

$$\Pr[\mathsf{E}_1 \wedge \mathsf{E}_2] = \Pr[\mathsf{E}_1] \cdot \Pr[\mathsf{E}_2 \mid \mathsf{E}_1].$$

We begin with the second factor. By definition,

$$\Pr[\mathsf{E}_2 \mid \mathsf{E}_1] = \Pr\big[S_y^+ \text{ is good over } S_x^+ \mid \mathsf{E}_1\big].$$

Now fix arbitrary values of x and y, and also fix an arbitrary realization of the internal randomness used to generate S_x^+. Under this conditioning, the sets S_x^+ and S_y are completely determined, and the set S_y^+ is produced solely by the randomness of the procedure

$$S_y^+ \leftarrow \mathsf{Refine}(\mathsf{L}(y), S_y, S_x^+).$$

Lemma 4.4 applies uniformly to every fixed choice of input set and base set. Therefore, for every such fixed choice of x, y, and randomness leading to S_x^+, the probability that S_y^+ is bad over S_x^+ is

at most $(q^2+1)/2^q$. Equivalently,

$$\Pr\left[S_y^+ \text{ is good over } S_x^+ \mid x, y, S_x^+\right] \geq 1 - \frac{q^2+1}{2^q}.$$

Since this lower bound holds for every conditioning of this form, it also follows that

$$\Pr[\mathsf{E}_2 \mid \mathsf{E}_1] \geq 1 - \frac{q^2+1}{2^q}.$$

It remains to bound $\Pr[\mathsf{E}_1]$, where the probability is taken over the random choice of x, y and the internal randomness used to generate S_x^+. In fact, we prove the following stronger statement: for every fixed $x \in \mathbb{Z}_N$,

$$\begin{aligned}\Pr[y \text{ is bad for } S_x^+] &\leq \Pr[S_x^+ \text{ is bad over } \mathbb{Z}_N] + \Pr[y \text{ is bad for } S_x^+ \mid S_x^+ \text{ is good over } \mathbb{Z}_N] \\ &\leq \frac{q^2+1}{2^q} + \frac{1}{q^2} \leq \frac{2}{q^2}.\end{aligned}$$

Equivalently,

$$\Pr[\mathsf{E}_1] = \Pr[y \text{ is good for } S_x^+] \geq 1 - \frac{2}{q^2}.$$

Combining this bound with the estimate for $\Pr[\mathsf{E}_2 \mid \mathsf{E}_1]$, we obtain

$$\begin{aligned}\Pr[\mathcal{A}_{I,\mathsf{DDH}}(\mathsf{L}(x), \mathsf{L}(y)) = \mathsf{P}(xy)] &\geq \Pr[\mathsf{E}_1 \wedge \mathsf{E}_2] \\ &\geq \left(1 - \frac{2}{q^2}\right)\left(1 - \frac{q^2+1}{2^q}\right) \\ &\geq 1 - \frac{2}{q^2} - \frac{q^2+1}{2^q} \geq 1 - \frac{3}{q^2}.\end{aligned}$$

This completes the proof.

4.4 Extension to general case

We now extend the above impossibility result from the special case $(1,2)$ to general parameters (k,n). As noted earlier, it suffices to rule out the case of $(n-1,n)$-robustness. Indeed, for every $k \leq n-1$, the notion of (k,n)-robustness is stronger than that of $(n-1,n)$-robustness: any combiner that remains secure whenever at least k candidates are secure must, a fortiori, remain secure whenever at least $n-1$ candidates are secure.

The impossibility for $(n-1,n)$-robustness follows the same overall strategy as in the $(1,2)$-case. We begin by defining the corresponding notion of partial information. Let

$$\mathsf{L}(x) = \left(\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}1}, \ldots, \mathsf{L}_{x\text{-}\mathsf{G}n}\right)$$

denote the representation of a combined-group element. For each $i \in [n]$, write

$$\mathsf{L}_{x\text{-}\mathsf{G}i} = \left(\hat{g}_i^{x_{i,1}}, \ldots, \hat{g}_i^{x_{i,t_i}}\right),$$

where the tuple $\mathsf{L}_{x\text{-}\mathsf{G}i}$ consists of group elements in the oracle $\mathcal{O}_i$. Thus, the n tuples $\mathsf{L}_{x\text{-}\mathsf{G}1}, \ldots, \mathsf{L}_{x\text{-}\mathsf{G}n}$ correspond to the n underlying generic groups $\mathcal{O}_1, \ldots, \mathcal{O}_n$, respectively.

$\mathsf{Refine}\big(\mathsf{Elem}, S_{\mathsf{refine}}, S_{\mathsf{base}}\big)$

$\underline{\mathsf{Refine}\big(\mathsf{Elem}, S_{\mathsf{refine}}, S_{\mathsf{base}}\big)}$:

While **True** do:

Sample $r_1, \ldots, r_{q^3} \xleftarrow{\$} S_{\mathsf{base}}$;

For $i = 1$ to q^3:

For $j \in \{2, \ldots, n\}$: Compute $(\mathcal{L}_{j,i}, \mathcal{T}_{j,i}) \xleftarrow{\mathcal{O}_j} \mathsf{Scale}\big(\mathsf{Elem}, r_i\big)$;

Compute $S_{r_i} := \{\tilde{x} \in S_{\mathsf{refine}} \mid \exists j \in \{2, \ldots, n\} \text{ s.t. } (\mathcal{L}_{j,i} \cdot F_j(\tilde{x}) \neq 0) \vee (\mathcal{T}_{j,i} \cdot F_j(\tilde{x}) = 0)\}$;

Testing Phase:

If $|S_{r_i}| < |S_{\mathsf{refine}}|/q$ for all $i \in [q^3]$: **Return** S_{refine};

Filtering Phase:

Update $S_{\mathsf{refine}} := S_{\mathsf{refine}} \setminus \cup_{i \in [q^3]} S_{r_i}$.

Figure 6: The procedure Refine for the general $(n-1, n)$ setting.

For each singleton set $I \in \big\{\{1\}, \ldots, \{n\}\big\}$, we define the *partial information* of $\mathsf{L}(x)$ with respect to I, denoted $\mathsf{P}_I(x)$, to consist of the bit-string component $\mathsf{L}_{x\text{-}\mathsf{str}}$ together with the exponent tuple contributed by the oracle indexed by I. Concretely, for $I = \{i\}$,

$$\mathsf{P}_I(x) = \big(\mathsf{L}_{x\text{-}\mathsf{str}}, (x_{i,1}, \ldots, x_{i,t_i})\big).$$

The extraction algorithm $\mathcal{A}_{I,\mathsf{Ext}}$, which takes $\mathsf{L}(x)$ and $\mathsf{L}(y)$ as input and aims to output $\mathsf{P}_I(xy)$, is defined in essentially the same way as before, with only minor modifications; see Figure 6. The only substantive difference is that, in the procedure Refine, one must now collect the query-response sets $(\mathcal{L}_{j,i}, \mathcal{T}_{j,i})$ for every oracle $\mathcal{O}_j$ with $j \in \{2, \ldots, n\}$.

The analysis of $\mathcal{A}_{I,\mathsf{Ext}}$ then proceeds exactly as in the $(1,2)$-case. In particular, for the $(n-1, n)$-setting, we obtain

$$\Pr\big[\mathcal{A}_{I,\mathsf{Ext}}(\mathsf{L}(x), \mathsf{L}(y)) = \mathsf{P}_I(xy)\big] \geq 1 - \frac{3}{q^2}.$$

We next show that this notion of partial information can again be used to break DDH security. Specifically, the following lemma shows that for at least one choice of $I \in \big\{\{1\}, \ldots, \{n\}\big\}$, the probability that the partial information of a uniformly random element collides with that of any fixed element is suitably bounded.

Lemma 4.5. *There exists $I \in \big\{\{1\}, \ldots, \{n\}\big\}$ such that for every $x \in \mathbb{Z}_N$,*

$$\Pr_{z \xleftarrow{\$} \mathbb{Z}_N}\big[\mathsf{P}_I(z) = \mathsf{P}_I(x)\big] \leq 1 - \frac{1}{2n}.$$

Proof. Suppose toward contradiction that the claim is false. Then, for every $i \in [n]$, there exists some $x_i \in \mathbb{Z}_N$ such that

$$\Pr_{z \xleftarrow{\$} \mathbb{Z}_N}\big[\mathsf{P}_{\{i\}}(z) = \mathsf{P}_{\{i\}}(x_i)\big] > 1 - \frac{1}{2n}.$$

For each $i \in [n]$, define

$$S_i := \left\{ z \in \mathbb{Z}_N : \mathsf{P}_{\{i\}}(z) = \mathsf{P}_{\{i\}}(x_i) \right\}.$$

Then, for every $i \in [n]$,

$$|S_i| > \left(1 - \frac{1}{2n}\right) N.$$

Equivalently,

$$|\mathbb{Z}_N \setminus S_i| < \frac{N}{2n}.$$

Therefore,

$$\left| \bigcap_{i=1}^{n} S_i \right| = N - \left| \bigcup_{i=1}^{n} (\mathbb{Z}_N \setminus S_i) \right| > N - \sum_{i=1}^{n} \frac{N}{2n} = \frac{N}{2}.$$

In particular, there exist two distinct values

$$z, z' \in \bigcap_{i=1}^{n} S_i.$$

For every $i \in [n]$, by the definition of S_i,

$$\mathsf{P}_{\{i\}}(z) = \mathsf{P}_{\{i\}}(z').$$

Together, these equalities determine the bit-string part and all exponent tuples of the representation. Hence,

$$\mathsf{L}(z) = \mathsf{L}(z'),$$

contradicting the injectivity of the labeling map. □

Putting the above bounds together, we conclude that $\mathcal{A}_{I,\mathsf{DDH}}$ breaks the DDH security of the $(n-1, n)$ robust combiner with advantage at least $\frac{1}{2n} - \frac{3}{q^2}$, which is noticeable for a suitable choice of the polynomial q.

5 Impossibility of robust generic combiner for DLog

In this section, we prove the DLog impossibility result for robust generic combiners in the small-order regime. At first sight this may seem counterintuitive, since the direct-product construction already gives a (k, n)-robust generic combiner for DLog-secure groups (and, similarly, for CDH-secure groups).

Let $(\mathbb{G}_1, g_1, p_1), \ldots, (\mathbb{G}_n, g_n, p_n)$ be n groups, where $p_1, \ldots, p_n$ are distinct λ-bit primes. Without loss of generality, define the combiner C as the direct product of the first $n - k + 1$ underlying groups, namely

$$g^x = \left(g_1^{x \bmod p_1}, \ldots, g_{n-k+1}^{x \bmod p_{n-k+1}}\right).$$

The (k, n)-robustness of this construction is immediate: any adversary that breaks the DLog (resp. CDH) security of C trivially breaks the DLog (resp. CDH) security of each of these $n - k + 1$ underlying groups.

At the same time, in this construction the order of the combined group is

$$N = \prod_{i=1}^{n-k+1} p_i.$$

Since each p_i is a λ-bit prime, it follows that $\log N = (n-k+1)\lambda$. Thus, the direct-product construction incurs a linear blow-up in the bit-length of the group order.

In what follows, we show that this overhead is inherent by proving a matching lower bound on the order of any such combiner. Roughly speaking, we show that for any constants k and n, no (k,n)-robust generic combiner for DLog-secure groups can have order N satisfying $\log N \leq (n-k)\lambda$.

Theorem 5.1. *Let λ be the security parameter, let n and k be constants, and let $\mathcal{O}_1, \ldots, \mathcal{O}_n$ be generic groups of distinct prime orders $p_1, \ldots, p_n$, where each p_i is a λ-bit prime. Let $\mathsf{C}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}$ be any generic combiner whose combined group has order N. If*

$$\log N \leq (n-k)\lambda,$$

then there exists a set $I \subseteq [n]$ with $|I| = n-k$ and a generic adversary $\mathcal{A}_I$ that is efficient with respect to I, such that $\mathcal{A}_I$ breaks the DLog security of C with noticeable probability in λ.

As in Section 3.2, to prove impossibility it suffices to exhibit a set $I \subseteq [n]$ with $|I| \leq n-k$ and an efficient generic adversary with respect to I that solves the DLog problem in the combined group with noticeable probability. The main difficulty, compared with the DDH setting where partial information may already suffice to distinguish a DDH tuple from a random one, is that breaking DLog security appears to require recovery of the entire hidden exponent information.

More concretely, let the challenge representation be

$$\mathsf{L}(x) = \big(\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}_1}, \ldots, \mathsf{L}_{x\text{-}\mathsf{G}_n}\big),$$

where for each $i \in [n]$, the component associated with $\mathcal{O}_i$ is of the form

$$\mathsf{L}_{x\text{-}\mathsf{G}_i} = \big(\hat{g}_i^{x_{i,1}}, \ldots, \hat{g}_i^{x_{i,t_i}}\big).$$

Thus, breaking DLog requires recovering the full hidden information

$$\mathsf{Inf}(x) := \big(\mathsf{L}_{x\text{-}\mathsf{str}}, F_1(x), \ldots, F_n(x)\big), \qquad F_i(x) := (x_{i,1}, \ldots, x_{i,t_i}).$$

By definition, the adversary $\mathcal{A}_I$ may be inefficient only with respect to the oracles indexed by I, where $|I| \leq n-k$. In other words, $\mathcal{A}_I$ may make unboundedly many queries to the oracles in I, but must remain efficient with respect to the remaining oracles in $\bar{I} = [n] \setminus I$. For any subset $I \subseteq [n]$, we write

$$\mathsf{P}_I(x) := \big(\mathsf{L}_{x\text{-}\mathsf{str}}, (F_i(x))_{i\in I}\big) \qquad \text{and} \qquad \mathsf{P}_{\bar{I}}(x) := \big(\mathsf{L}_{x\text{-}\mathsf{str}}, (F_i(x))_{i\notin I}\big).$$

Thus $\mathsf{P}_I(x)$ records the hidden information carried by the corrupted coordinates, while $\mathsf{P}_{\bar{I}}(x)$ records the remaining coordinates.

Our key observation is that, when the combined group has order N satisfying $\log N \leq (n-k)\lambda$, one can still efficiently extract nontrivial information about the coordinates indexed by $\bar{I}$. More precisely, we construct an efficient extractor $\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}$ that, given $\mathsf{L}(x)$, outputs partial information $\mathsf{P}_{\bar{I}}(x)$ with noticeable probability.

Adversary $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}_{I,\mathsf{DLog}}(\mathsf{L}(x))$

$\underline{\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_n}_{I,\mathsf{DLog}}(\mathsf{L}(x))}$:

Compute $\mathsf{P}_I(x) \leftarrow \mathcal{A}_{I,\mathsf{Dec}}(\mathsf{L}(x))$.

Compute $\mathsf{P}_{\bar{I}}(x) \leftarrow \mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}(\mathsf{L}(x))$.

If there exists $\tilde{x} \in \mathbb{Z}_N$ such that $\mathsf{P}_I(\tilde{x}) = \mathsf{P}_I(x) \wedge \mathsf{P}_{\bar{I}}(\tilde{x}) = \mathsf{P}_{\bar{I}}(x)$: **Return** $\tilde{x}$.

Return $\perp$.

Figure 7: The description of $\mathcal{A}_{I,\mathsf{DLog}}$ that breaks the DLog security of the combiner.

Adversary $\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}(\mathsf{L}(x))$

$\underline{\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}(\mathsf{L}(x))}$:

Initialize $S \leftarrow \emptyset$.

For $i \in [n]$: compute $(\mathcal{L}_i, \mathcal{T}_i) \xleftarrow{\mathcal{O}_i} \mathsf{VT}(\mathsf{L}(x))$.

For $i \in [n]$ with $i \notin I$:

If there exist t_i linear independent constraints in $\mathcal{L}_i$:

Solve $(x_{i,1}, \ldots, x_{i,t_i})$ from $\mathcal{L}_i \cdot F_i(x) = 0$;

$S \leftarrow S \cup \{(x_{i,1}, \ldots, x_{i,t_i})\}$.

Else **Return** $\perp$.

Return S.

Figure 8: The efficient algorithm $\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}$ that recovers the partial information $\mathsf{P}_{\bar{I}}(x)$.

This already suffices for the impossibility proof. Indeed, once such an extractor is available, we can combine it with the unbounded power of $\mathcal{A}_I$ on the coordinates in I to recover the full exponent information, thereby breaking the DLog security of the combiner, as illustrated in Figure 7.

5.1 Extracting the remaining information efficiently

We now prove Theorem 5.1 by showing how to construct an efficient generic algorithm $\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}$ that takes $\mathsf{L}(x)$ as input and outputs $\mathsf{P}_{\bar{I}}(x)$. Our construction relies on the validity-test algorithm VT as a subroutine, as illustrated in Figure 8. More concretely, for each $i \in [n]$, let

$$(\mathcal{L}_i, \mathcal{T}_i) \xleftarrow{\mathcal{O}_i} \mathsf{VT}(\mathsf{L}(x))$$

denote the collection of queries issued to $\mathcal{O}_i$ during the execution of $\mathsf{VT}(\mathsf{L}(x))$, where $\mathcal{L}_i$ consists of those queries that receive a positive answer and $\mathcal{T}_i$ consists of those that receive a negative answer.

We observe that for each $i \in \bar{I}$, if $\mathcal{L}_i$ contains t_i linearly independent constraints, then one can directly recover the tuple $(x_{i,1}, \ldots, x_{i,t_i})$ by solving the corresponding linear system $\mathcal{L}_i \cdot F_i(x) = 0$.

It therefore remains to show that this good event occurs with noticeable probability whenever $\log N \leq (n-k)\lambda$.

At a high level, our analysis relies critically on the perfect correctness of VT. Specifically, for every candidate label L^*, the algorithm VT outputs 1 if and only if there exists some $x \in \mathbb{Z}_N$ such that $\mathsf{L}^* = \mathsf{L}(x)$. Consequently, the set of accepting labels is of size at most N, since each accepting label must correspond to some element $x \in \mathbb{Z}_N$. Our proof shows that, conditioned on $\log N \leq (n-k)\lambda$, if the above good event fails to occur for any I where $|I| = n-k$, then the number of accepting labels must in fact be significantly larger than N, yielding a contradiction.

More concretely, let $\mathsf{L}(x)$ be a representation of x, where

$$\mathsf{L}(x) = \left(\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}_{x\text{-}\mathsf{G}_1}, \ldots, \mathsf{L}_{x\text{-}\mathsf{G}_n}\right).$$

For each $i \in [n]$, the component associated with $\mathcal{O}_i$ is of the form $\mathsf{L}_{x\text{-}\mathsf{G}_i} = \left(\hat{g}_i^{x_{i,1}}, \ldots, \hat{g}_i^{x_{i,t_i}}\right)$. We then consider a candidate label L^* of the form

$$\mathsf{L}^* = \left(\mathsf{L}_{x\text{-}\mathsf{str}}, \mathsf{L}^*_{x\text{-}\mathsf{G}_1}, \ldots, \mathsf{L}^*_{x\text{-}\mathsf{G}_n}\right),$$

where for each $i \in [n]$, the component associated with $\mathcal{O}_i$ is $\mathsf{L}^*_{x\text{-}\mathsf{G}_i} = \left(\hat{g}_i^{x^*_{i,1}}, \ldots, \hat{g}_i^{x^*_{i,t_i}}\right)$. We say that L^* is *consistent* with $(\mathcal{L}_i, \mathcal{T}_i)$ if both of the following conditions hold:

- for every $(\alpha_0, \ldots, \alpha_{t_i}) \in \mathcal{L}_i$, there is $\alpha_0 + \sum_{j=1}^{t_i} \alpha_j x^*_{i,j} = 0$;
- for every $(\alpha_0, \ldots, \alpha_{t_i}) \in \mathcal{T}_i$, there is $\alpha_0 + \sum_{j=1}^{t_i} \alpha_j x^*_{i,j} \neq 0$.

Next, let S^* denote the set of all candidate labels that are consistent with $(\mathcal{L}_i, \mathcal{T}_i)$ for every $i \in [n]$. The key observation is that every candidate label $\mathsf{L}^* \in S^*$ must be accepted by the validity test, namely,

$$\mathsf{VT}(\mathsf{L}^*) = 1.$$

Therefore, to derive the desired contradiction, it suffices to show that the size of S^* is in fact much larger than N.

We begin with a simple counting observation. For any pair $(\mathcal{L}_i, \mathcal{T}_i)$, if $\mathcal{L}_i$ does not contain t_i linearly independent constraints, then we claim that the number of tuples $(x^*_{i,1}, \ldots, x^*_{i,t_i})$ consistent with $(\mathcal{L}_i, \mathcal{T}_i)$ is at least $p_i/2$.

Indeed, suppose that $\mathcal{L}_i$ contains d_i linearly independent constraints, where $d_i < t_i$. Then the linear system determined by $\mathcal{L}_i$ alone admits exactly $p_i^{t_i - d_i}$ solutions. It remains to impose consistency with $\mathcal{T}_i$, which amounts to ruling out those tuples that also satisfy at least one linear equation corresponding to a vector in $\mathcal{T}_i$.

More precisely, for each $(\beta_0, \ldots, \beta_{t_i}) \in \mathcal{T}_i$, the number of tuples $(x^*_{i,1}, \ldots, x^*_{i,t_i})$ that are consistent with $\mathcal{L}_i$ and also satisfy

$$\beta_0 + \sum_{j=1}^{t_i} \beta_j \cdot x^*_{i,j} = 0$$

is at most $p_i^{t_i - d_i - 1}$. Therefore, the number of tuples consistent with $(\mathcal{L}_i, \mathcal{T}_i)$ is at least

$$p_i^{t_i - d_i} - |\mathcal{T}_i| \cdot p_i^{t_i - d_i - 1} = p_i^{t_i - d_i - 1} \cdot (p_i - |\mathcal{T}_i|) \geq (p_i - |\mathcal{T}_i|) \geq \frac{p_i}{2},$$

where the last inequality holds for sufficiently large p_i, since the adversary makes only polynomially many queries to $\mathcal{O}_i$, and hence $|\mathcal{T}_i|$ is polynomially bounded.

We say that $i \in \mathsf{Bad}$ if $\mathcal{L}_i$ does not contain t_i linearly independent constraints. Suppose that there are at least $n - k + 1$ bad indices. Then

$$|S^*| \;\geq\; \prod_{i \in \mathsf{Bad}} \frac{p_i}{2} \;\geq\; \left(2^{\lambda-2}\right)^{n-k+1}.$$

Since $\log N \leq (n-k)\lambda$, it follows that $|S^*| \gg N$, contradicting the fact that the number of accepting labels is at most N.

Therefore, for every $x \in \mathbb{Z}_N$, the extractor $\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}$ can find a set $I' \subseteq [n]$ such that $|\bar{I}'| \geq k$ and

$$\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}(\mathsf{L}(x)) = \mathsf{P}_{\bar{I}'}(x).$$

Finally, since n and k are constants, the number of possible choices of I is at most $\binom{n}{k}$, which is also a constant. By an averaging argument, there exists a set $I \subseteq [n]$ with $|I| = n - k$ such that

$$\Pr\left[\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}(\mathsf{L}(x)) = \mathsf{P}_{\bar{I}}(x)\right] \geq \frac{1}{\binom{n}{k}}.$$

Hence $\mathcal{A}^{\mathsf{eff}}_{I,\mathsf{Ext}}$ outputs $\mathsf{P}_{\bar{I}}(x)$ with noticeable probability, as required.

References

[AJN+16] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 491–520. Springer, Berlin, Heidelberg, August 2016.

[AJS17] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Robust transforming combiners from indistinguishability obfuscation to functional encryption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 91–121. Springer, Cham, April / May 2017.

[BB06] Dan Boneh and Xavier Boyen. On the impossibility of efficiently combining collision resistant hash functions. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 570–583. Springer, Berlin, Heidelberg, August 2006.

[CDFR17] Ignacio Cascudo, Ivan Damgård, Oriol Farràs, and Samuel Ranellucci. Resource-efficient ot combiners with active security. In *Theory of Cryptography Conference*, pages 461–486. Springer, 2017.

[CMVX25] Michele Ciampi, Lorenzo Magliocco, Daniele Venturi, and Yu Xia. Robust non-interactive zero-knowledge combiners. *Cryptology ePrint Archive*, 2025.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003.

[DFG+23] Yevgeniy Dodis, Niels Ferguson, Eli Goldin, Peter Hall, and Krzysztof Pietrzak. Random oracle combiners: breaking the concatenation barrier for collision-resistance. In *Annual International Cryptology Conference*, pages 514–546. Springer, 2023.

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

[DHH+21] Nico Döttling, Dominik Hartmann, Dennis Hofheinz, Eike Kiltz, Sven Schäge, and Bogdan Ursu. On the impossibility of purely algebraic signatures. In *Theory of Cryptography Conference*, pages 317–349. Springer, 2021.

[DK05] Yevgeniy Dodis and Jonathan Katz. Chosen-ciphertext security of multiple encryption. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 188–209. Springer, Berlin, Heidelberg, February 2005.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Berlin, Heidelberg, August 1984.

[FFHH25] Sebastian Faller, Marc Fischlin, Julius Hardt, and Julia Hesse. Combining oblivious pseudorandom functions. Cryptology ePrint Archive, Report 2025/1084, 2025.

[FL08] Marc Fischlin and Anja Lehmann. Multi-property preserving combiners for hash functions. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 375–392. Springer, Berlin, Heidelberg, March 2008.

[FLP08] Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust multi-property combiners for hash functions revisited. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 655–666. Springer, Berlin, Heidelberg, July 2008.

[FRG23] Oriol Farràs and Jordi Ribes-González. One-out-of-q ot combiners. *IEEE Transactions on Information Theory*, 70(4):2984–2998, 2023.

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. Kem combiners. In *IACR International Workshop on Public Key Cryptography*, pages 190–218. Springer, 2018.

[Her05] Amir Herzberg. On tolerant cryptographic constructions. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 172–190. Springer, Berlin, Heidelberg, February 2005.

[Her09] Amir Herzberg. Folklore, practice and theory of robust combiners. *J. Comput. Secur.*, 17(2):159–189, April 2009.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 96–113. Springer, Berlin, Heidelberg, May 2005.

[HKNY24] Taiga Hiroka, Fuyuki Kitagawa, Ryo Nishimaki, and Takashi Yamakawa. Robust combiners and universal constructions for quantum cryptography. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part II*, volume 15365 of *LNCS*, pages 126–158. Springer, Cham, December 2024.

[HMS19] Goichiro Hanaoka, Takahiro Matsuda, and Jacob CN Schuldt. A new combiner for key encapsulation mechanisms. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 102(12):1668–1675, 2019.

[HR25] Julia Hesse and Michael Rosenberg. PAKE combiners and efficient post-quantum instantiations. In *EUROCRYPT 2025, Part II*, LNCS, pages 395–420. Springer, Cham, June 2025.

[JMS20] Aayush Jain, Nathan Manohar, and Amit Sahai. Combiners for functional encryption, unconditionally. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 141–168, Cham, 2020. Springer International Publishing.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall, CRC Press, third edition, 2014.

[LL25] You Lyu and Shengli Liu. Hybrid password authentication key exchange in the uc framework. In *Advances in Cryptology – EUROCRYPT 2025: 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4–8, 2025, Proceedings, Part II*, page 421–450, Berlin, Heidelberg, 2025. Springer-Verlag.

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

[Mit12] Arno Mittelbach. Hash combiners for second pre-image resistance, target collision resistance and pre-image resistance have long output. In *International Conference on Security and Cryptography for Networks*, pages 522–539. Springer, 2012.

[MMN16] Mohammad Mahmoody, Ameer Mohammed, and Soheil Nematihaji. On the impossibility of virtual black-box obfuscation in idealized models. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 18–48. Springer, Berlin, Heidelberg, January 2016.

[MP06] Remo Meier and Bartosz Przydatek. On robust combiners for private information retrieval and other primitives. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 555–569. Springer, Berlin, Heidelberg, August 2006.

[MPW07] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. Robuster combiners for oblivious transfer. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 404–418. Springer, Berlin, Heidelberg, February 2007.

[Pie07] Krzysztof Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 23–33. Springer, 2007.

[Pie08] Krzysztof Pietrzak. Compression from collisions, or why crhf combiners have a long output. In *Annual International Cryptology Conference*, pages 413–432. Springer, 2008.

[Ps16] Rafael Pass and abhi shelat. Impossibility of VBB obfuscation with ideal constant-degree graded encodings. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 3–17. Springer, Berlin, Heidelberg, January 2016.

[RSS20] Lior Rotem, Gil Segev, and Ido Shahaf. Generic-group delay functions require hidden-order groups. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 155–180. Springer, Cham, May 2020.

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 1–20. Springer, Berlin, Heidelberg, February 2004.

[SGS21] Gili Schul-Ganz and Gil Segev. Generic-group identity-based encryption: A tight impossibility result. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPIcs*, pages 26:1–26:23. Schloss Dagstuhl, July 2021.

[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, Berlin, Heidelberg, May 1997.

[Zha22] Mark Zhandry. To label, or not to label (in generic groups). In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 66–96. Springer, Cham, August 2022.

[ZZ18] Mark Zhandry and Cong Zhang. Impossibility of order-revealing encryption in idealized models. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 129–158. Springer, Cham, November 2018.