Tight Time-Space Tradeoffs for the Decisional Diffie-Hellman Problem

Akshima[1], Tyler Besselman[1], Siyao Guo*[1], Zhiye Xie[1], and Yuping Ye[2,1]

[1]NYU Shanghai†
[1]{akshima, tyler.william.b, siyao.guo, zhiye.xie, yy3813}@nyu.edu
[2]East China Normal University

Abstract

In the (preprocessing) Decisional Diffie-Hellman (DDH) problem, we are given a cyclic group G with a generator g and a prime order N, and we want to prepare some advice of size S, such that we can efficiently distinguish (g^x, g^y, g^{xy}) from (g^x, g^y, g^z) in time T for uniformly and independently chosen x, y, z from $\mathbb{Z}_N$. This is a central cryptographic problem whose computational hardness underpins many widely deployed schemes, such as the Diffie–Hellman key exchange protocol.

We prove that any generic preprocessing DDH algorithm (operating in any cyclic group) achieves advantage at most $O(ST^2/N)$. This bound matches the best known attack up to poly-log factors, and confirms that DDH is as secure as the (seemingly harder) discrete logarithm problem against preprocessing attacks. Our result resolves an open question by Corrigan-Gibbs and Kogan (EUROCRYPT 2018), who proved optimal bounds for many variants of discrete logarithm problems except DDH (with an $\tilde{O}(\sqrt{ST^2/N})$ bound).

We obtain our results by adopting and refining the approach by Gravin, Guo, Kwok, Lu (SODA 2021) and by Yun (EUROCRYPT 2015). Along the way, we significantly simplified and extended the above techniques which may be of independent interest. The highlights of our techniques are as follows:

- We obtain a simpler reduction from decisional problems against S-bit advice to their S-wise XOR lemmas against *zero-advice*, recovering the reduction by Gravin, Guo, Kwok and Lu (SODA 2021).
- We show how to reduce generic hardness of decisional problems to their variants in the simpler hyperplane query model proposed by Yun (EUROCRYPT 2015). This is the first work analyzing a decisional problem in Yun's model, answering an open problem proposed by Auerbach, Hoffman, and Pascual-Perez (TCC 2023).
- We prove an S-wise XOR lemma of DDH in Yun's model. As a corollary, we obtain the generic hardness of the S-XOR DDH problem.

*Supported by National Natural Science Foundation of China Grant No. 62102260, NYTP Grant No. 20121201 and NYU Shanghai Boost Fund. Parts of the work are done while visiting CIFRA Institute, Bocconi University supported by Fondazione Cariplo (n. 2021-3641).

†Shanghai Frontiers Science Center of Artificial Intelligence and Deep Learning

1 Introduction and Overview

The Decisional Diffie-Hellman (DDH) problem is a classic problem in cryptography. The assumption that it is a hard problem forms the basis for the security claims of the Diffie-Hellman key exchange protocol, which was introduced in a 1976 paper by Diffie and Hellman [DH76], and is still used to this day to secure a large chunk of communication that takes place both over the public internet and between private parties. As such, strong theoretical bounds on algorithms attacking this problem are quintessential for guaranteeing the security of its users.

The DDH problem may be roughly summarized as follows: given a cyclic group G with a generator g and (usually prime) order N, sample three random integers x, y, z from $\mathbb{Z}_N$. Then, an attacker is tasked with determining, given the values (g^x, g^y), whether a third, unknown element $h \in G$ is equal to the value g^{xy} or g^z, chosen at random in each instance.

When this problem was introduced, there was only an assumption that it was difficult to solve. It is clear that an algorithm able to solve for the *discrete logarithm* of a given group element would be able to differentiate by computing g^{xy} and comparing it to the value h after finding the discrete logarithm value $x = \log_G(g^x)$ of the second input element. But is there a more efficient way? It was later shown by Shoup in [Sho97] that no algorithm solving the Diffie-Hellman problem for a generic group G can be more efficient than an algorithm solving for the discrete logarithm of a random element $h = g^x \in G$. Thus, in this simple case, solving the Diffie-Hellman problem is as difficult as solving the discrete logarithm problem (without relying on special techniques exploiting the structure of a specific group or class of groups G).

In the modern age of computation, however, a security bound for this simple setting is not enough. It is not unreasonable for strong adversaries, with lots of computation power, to invest their resources in a one-time large pre-computation, which can significantly reduce the time required to solve an instance of a problem later. In some cases, there can be time limits to solve an instance of a problem, and using the pre-computation may allow the adversary to solve the problem within the given time frame. For instance, while it may be hard to invert a cryptographic hash function, an attacker with enough computation power and time may pre-compute rainbow tables in advance so as to more efficiently invert a single hash value given at a later point in time. In this case, we assume the time for pre-computations is effectively unlimited, and we concern ourselves with the size of the pre-computed input required to find inversions efficiently. Specifically, it becomes interesting to analyze the trade-off between the space used in pre-computation and the amount of time taken when presented with an actual instance.

While auxiliary input was originally defined for the random oracle model in [Unr07, DGK17], it was later extended to the Generic Group Model (GGM) by Corrigan-Gibbs and Kogan in [CK18] and by Coretti et al. in [CDG18]. An adversary $\mathcal{A}$ in the auxiliary input(AI) idealized oracle (random oracle/random permutation/ generic group) model can be thought of as a two-stage algorithm $(\mathcal{A}_0, \mathcal{A}_1)$:

- In the first stage, which we will refer to as the "offline" stage or the "pre-computation" stage, $\mathcal{A}_0$ gets unbounded access to the oracle but has to output bounded advice γ about the oracle.
- In the second stage, which we will refer to as the "online" stage, $\mathcal{A}_1$ gets the advice γ from $\mathcal{A}_0$ and a challenge about the oracle as input. $\mathcal{A}_1$ makes a bounded number of queries to the oracle.

In the Generic Group Model, a group of order N is described in terms of a random injective labeling function σ mapping $\mathbb{Z}_N$ to a set of labels $\mathcal{L}$ such that $\sigma(1), \ldots, \sigma(N-1)$ correspond to the elements of the group with discrete logarithms $1, \ldots, N-1$ respectively. Group operations are performed by querying a generic group oracle $\mathcal{O}_\sigma(\cdot, \cdot)$ corresponding to the labeling function σ such that for any $\mathsf{X}, \mathsf{Y} \in \mathcal{L}$, $\mathcal{O}_\sigma(\mathsf{X}, \mathsf{Y})$ outputs

- $\sigma(x + y)$ if there exists $x, y \in \mathbb{Z}_N$ such that $\sigma(x) = \mathsf{X}$ and $\sigma(y) = \mathsf{Y}$
- $\perp$ otherwise.

For any adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ in AI-GGM, $\mathcal{A}_0$ gets input $\sigma(1)$ and unbounded access to the oracle $\mathcal{O}_\sigma$. In effect, then, $\mathcal{A}_0$ gets access to the entire function table of σ. However, $\mathcal{A}_0$ can output only bounded length

advice, say some γ where $|\gamma| = S$ bits. Then $\mathcal{A}_1$ takes γ and a challenge as input, makes a bounded number of queries, say T, to $\mathcal{O}_\sigma$ and computes an output.

Note that the challenge and the output of $\mathcal{A}_1$ depend on the problem $\mathcal{A}$ is solving. For an algorithm $\mathcal{A}$ solving the discrete logarithm problem, $\mathcal{A}_1$ gets $(\sigma(1), \sigma(x))$ as challenge for a random $x \in \mathbb{Z}_N$. To "win", $\mathcal{A}_1$ needs to output x. For an algorithm $\mathcal{A}$ solving the DDH problem, $\mathcal{A}_1$ gets a tuple $(\sigma(1), \sigma(x), \sigma(y), \sigma(u))$ as challenge where for random and independent $x, y, z \in \mathbb{Z}_N$ and random $b \in \{0, 1\}$, $u := (1-b)xy + bz$, and $\mathcal{A}_1$ outputs a guess, say $\hat{b}$, for b and "wins" if $\hat{b} = b$.

1.1 Our Results

In this work, we analyze the Decisional Diffie-Hellman (DDH) problem in the AI-GGM setting. We try to understand the following aspects of the DDH problem: for any algorithm with bounded space given access to the cyclic group G, what is the lower bound on the number of group operations required to solve a given instance of the problem? Is the DDH problem in this setting still asymptotically as difficult as the discrete logarithm problem? We successfully obtained answers to both questions, positively affirming the latter.

Tight time-space lower bounds for the DDH Problem. The main result of this work is the tight lower bound $ST^2 = \Omega(\varepsilon N)$ for any generic algorithm with S-bit advice/auxiliary input, making T oracle queries on a group of order N and solving the DDH problem in AI-GGM with probability at least $1/2 + \varepsilon$. Formally,

Theorem 1. *Let N be a prime and $S, T \geq 1$ be integers. For any generic algorithm with preprocessing $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{A}_0$ outputs an S-bit advice string, and $\mathcal{A}_1$ makes at most T oracle queries, it holds that*

$$\mathbb{P}\left[\mathcal{A}_1^{\mathcal{O}_\sigma}\left(\mathcal{A}_0^{\mathcal{O}_\sigma}\left(\sigma(1)\right), \sigma(1), \sigma(x), \sigma(y), \sigma(u)\right) = b\right] \leq \frac{1}{2} + 2^{11}\frac{ST^2}{N},$$

where $x, y, z \leftarrow \mathbb{Z}_N$, $b \leftarrow \{0, 1\}$, $u = bz + (1-b)xy$, and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ is a random injective function.

There is a corresponding algorithm for DDH with advantage $\tilde{\Omega}(ST^2/N)$ based on known algorithms for the discrete logarithm problem with an asymptotically equivalent advantage in the preprocessing setting (see Appendix A). Thus, the above security bound is tight for all values S, T, N up to poly-log factors.

Prior to this work, the best-known security bound in this setting was $\widetilde{O}(\sqrt{ST^2/N})$ in [CK19], leaving a quadratic gap between security upper and lower bounds. Our result closes this gap and confirms that DDH is as secure against preprocessing attacks as the (seemingly more difficult) discrete logarithm problem.

Generic hardness of the k-XOR DDH Problem. To prove this theorem, we consider a fundamental variant of DDH, called the *k-XOR DDH problem*. Let's define the DDH bit of a given DDH instance as a bit indicating whether the third coordinate is xy or z. The k-XOR DDH problem can be described as follows: given k independently sampled DDH instances, compute the XORed DDH bits of the k DDH instances.

We prove the lower bound $kT^2 = \Omega(\varepsilon^{1/k} N)$ for any generic algorithm (without preprocessing) with kT oracle queries on a group of order N that solves the k-XOR DDH problem in GGM with probability at least $1/2 + \varepsilon$.

Theorem 2. *Let N be a prime and $k, T \geq 1$ be integers. For any generic algorithm $\mathcal{B}$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{B}$ makes at most kT oracle queries, it holds that*

$$\mathbb{P}\left[\mathcal{B}^{\mathcal{O}_\sigma}\left(\sigma(1), \sigma(\mathbf{w})\right) = \bigoplus_{i=1}^{k} b_i\right] \leq \frac{1}{2} + \frac{1}{2}\left(2^{11}\frac{kT^2}{N}\right)^k,$$

where $\sigma(\mathbf{w}) := \left\{\sigma(x_i), \sigma(y_i), \sigma(u_i)\right\}_{i=1}^k$ for $u_i = b_i z_i + (1-b_i)x_i y_i$, $\left\{x_i, y_i, z_i\right\}_{i=1}^k \leftarrow \mathbb{Z}_N^{3k}$, $\{b_1, \ldots, b_k\} \leftarrow \{0, 1\}^k$, and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ a uniformly random injective function.

Notice that when $k = 1$, the security bound from the theorem above is exactly the lower bound for the standard DDH problem, which was proven tight in [Sho97]. For general k, our bound confirms that XOR amplifies the hardness of DDH.

Yun [Yun15] considered the generic hardness of solving multiple discrete logarithm problems simultaneously and showed a similar lower bound $kT^2 = \Omega(\varepsilon^{1/k}N)$ for solving k-discrete logarithm problems using kT queries with success probability at least ε. Our result suggests that the (seemingly easier) k-XOR DDH admits similar hardness to the k-discrete logarithm problem.

1.2 Our Proofs

To obtain this result, we adopt and further improve the techniques from [GGKL21] and [Yun15]. Our simplification and refinement of these techniques not only allow us to obtain the desired time-space lower bounds for the DDH problem but also characterize the potential of these techniques, which could be of independent interest. The proof of our main results will be split into three stages, each corresponding to one of the three lemmas in Section 3.

From preprocessing DDH to k-XOR DDH. Once a single bit of advice about σ is allowed, standard analyzing techniques for the generic group model (such as the commonly used lazy sampling technique) may become inapplicable. One natural idea is reducing preprocessing problems to their variants without preprocessing so we can apply standard techniques.

Thus, in Lemma 1 (see Section 3.1 for details), we reduce the DDH problem with S-bit advice to the S-XOR DDH problem without advice. In particular, we show that, for any even S, a preprocessing adversary $\mathcal{A}$ for DDH with advantage ϵ, by simply guessing the advice uniformly and randomly, and then running its online algorithm S times, we obtain an adversary $\mathcal{B}$ for the S-XOR DDH problem with advantage at least $(\epsilon/2)^S$. The restriction on S being even is minor because we can reduce an odd S to an even S before using Lemma 1. Intuitively, if σ is fixed, guessing the S-bit advice correctly happens with probability $1/2^S$, and conditioning on that, the advantage of solving S-XOR DDH is ϵ^S if $\mathcal{A}$ has advantage ϵ (for this fixed σ). Thus, the advantage of $\mathcal{B}$ is at least $\epsilon^S/2^S$. To obtain the formal argument, we use the fact that S is an even number to argue that the advantage is at least zero when conditioning on any wrong guess, and we apply Jensen's inequality to take care of the distributional case of σ.

Reducing decisional problems with S-bit advice to their S-wise XOR variants without advice was first proposed by Gravin et al. [GGKL21] in the context of proving tight time-space tradeoffs for the hardcore bit problem. The reduction in [GGKL21] requires proving a concentration bound for distributions satisfying a related k-wise XOR condition. Our reduction, which provides an algorithm for the k-XOR variant by guessing the advice based on the preprocessing algorithm, is considerably simpler and more constructive. The idea of guessing the advice is inspired by the reduction for reducing search problems with S-bit advice to their S-wise multi-instance variants without advice in [CGLQ20] (and its refinement [AGL22]). Our reduction can be viewed as an analog to their reductions for decisional problems.

From Generic Group Model to Hyperplane Query Model. By Lemma 1, we can shift our attention to the k-XOR DDH problem in the generic group model. The encoding σ and the oracle $\mathcal{O}_\sigma$ are slightly complex to work with for problems involving multiple instances. Inspired by Yun [Yun15]'s approach for the multiple discrete logarithm problem, we aim to remove the encoding σ and work with a simpler oracle.

Thus, in Lemma 2 (see Section 3.2 for details), we reduce the k-XOR DDH problem in the generic group model to a corresponding k-XOR DDH problem in the Hyperplane Query Model (HQM) introduced by Yun [Yun15]. In particular, we show that any algorithm with input $\big(\sigma(1), \sigma(\mathbf{w})\big)$, where $\mathbf{w} := \{x_i, y_i, u_i\}_{i=1}^k$ and with oracle access to $\mathcal{O}_\sigma$, can be simulated by an algorithm with only a few hyperplane queries to $H_{\mathbf{w}}$, and incurs a quadratic blow-up in the query complexity. The hyperplane query simply submits a linear equation of the form $L(\cdot) = c$, and the hyperplane query oracle $\mathcal{H}_{\mathbf{w}}$ outputs 1 if $L(\mathbf{w}) = c$, otherwise it outputs 0.

Our proof is largely based on Yun's proof, which utilizes the lazy sampling property of σ — i.e., new input-output pairs can be generated on the fly with limited information from previously generated pairs. To record the generated input-output pairs, the simulation maintains a list $\{(L_i, s_i)\}_{i\geq 0}$, where L_i is a multi-linear polynomial in variables $\{X_1, Y_1, U_1, \ldots, X_k, Y_k, U_k\}$ such that $\sigma(L_i(\mathbf{w})) = s_i$. To simulate $\sigma(\cdot)$ and $\sigma^{-1}(\cdot)$ (and thus $\mathcal{O}_\sigma$), we demonstrate how to search an existing pair and how to sample a new pair in the maintained list using a few hyperplane queries to $\mathbf{w}$.

One of the main differences between our proof and Yun's proof is that Yun's reduction assumes that the adversary in the generic group model never queries the oracle on labels that have not appeared previously. Our proof makes no such assumptions and shows how to sample new preimages using only a few hyperplane queries in total (see the subroutine **FindPreimage** in Section 3.2).

k-XOR DDH in the Hyperplane Query Model. In Lemma 3, we show that any algorithm with q queries for the k-XOR DDH problem has advantage $\approx (q/NK)^k$. Plugging in $q \approx (kT)^2$, we obtain that the generic hardness of k-XOR DDH against kT-query adversaries is $(kT^2/N)^k$, and the generic hardness of preprocessing DDH against S-bit advice and T-query algorithm is $\approx ST^2/N$. See Section 3.4 for details.

Our proof proceeds by induction on (k, q). In particular, in Claim 2 and Claim 3, we bound the best advantage of q-query algorithm for k-XOR DDH problem by related best advantages achieved by $(q-1)$-query algorithms for k-XOR problem or q-query algorithm for $(k-1)$-XOR DDH problem. Then by carefully choosing an inductive hypothesis $\binom{q+k}{k}(4/N)^k$ and an inductive step, we obtain the desired bound.

Given the best q-query algorithm for the k-XOR DDH problem, it is helpful to view it as a decision tree of depth q such that each internal node is a hyperplane query, and its subtrees represent the sub-algorithms executed according to the answer from the hyperplane query oracle. Let H denote the first hyperplane query. We will bound the advantage of the algorithm based on whether H is satisfied or not.

Intuitively, if H is not satisfied, this query should be somewhat useless because it provides very little information about the variables. So the contribution of this case should be roughly the same as that of its $(q-1)$ sub-algorithm for the k-XOR DDH problem. We capture this intuition in Claim 2. In particular, we show that, at the cost of an additional item of the best advantage assuming that H holds (which we will handle later anyway), we can bound the advantage when H is not satisfied by the advantage of a $(q-1)$-query algorithm for the k-XOR DDH problem.

If H is satisfied, the intuition is that this happens with a probability of only $O(1/N)$. Conditioning on H happens, the sub-algorithm executed can be converted into either a $(q-1)$ algorithm for the $(k-1)$-XOR problem or a q algorithm for the $(k-1)$-XOR problem where the first equation is satisfied. See Claim 3 for the details.

1.3 Prior and Other Related Works

Several works [BL13, LCH11, Mih10] have unveiled the ability of generic algorithms with preprocessing to tackle the discrete logarithm problem. These works have shown that generic algorithms for every group of order N use $N^{1/3}$ bits of group-specific advice and approximately $N^{1/3}$ online time to solve the discrete logarithm problem. This beats Shoup's bound without preprocessing of $\Omega(\sqrt{N})$ group operations for a group of order N in [Sho97]. Inspired by these preprocessing attacks, Corrigan-Gibbs and Kogan [CK18] were the first to define and analyze generic algorithms with precomputation for the discrete logarithm, DDH, Computational Diffie Hellman and multiple discrete logarithm problems. Corrigan-Gibbs and Kogan showed the optimal generic hardness $ST^2 = \widetilde{\Omega}(\varepsilon N)$ for the discrete logarithm problem in the auxiliary input model, where the advice is S-bits, the algorithm makes T queries in the online stage, and the success probability is ε for a group of (prime) size N. Later, Bartusek, Ma, and Zhandry [BMZ19] found that the fixed-generator and the random-generator variants of the discrete logarithm problem are not equally hard and proved the security bound of the random-generator variant of the discrete logarithm problem is $ST^2 = \widetilde{\Theta}(\sqrt{\varepsilon}N)$ using the presampling technique from [CDG18].

Corrigan-Gibbs and Kogan also studied the multiple discrete logarithm problem (where the algorithm is tasked with solving multiple instances of the discrete logarithm problem simultaneously) for preprocessing

generic algorithms (with S-bit advice and T online queries) and proved a security bound for solving M instances of the discrete logarithm with a probability of at least ε of $ST^2 = \Omega(\varepsilon^{1/M}NM)$. They were the first to study the multiple discrete logarithm problem with auxiliary input. Several works [FJM14, KS01, Yun15] prior to [CK18] studied both upper and lower bounds for the problem without preprocessing. One work of particular importance is [Yun15]. To show the generic hardness of solving the multiple discrete logarithm problem, Yun introduced a related computational model called the search-by-hyperplane-queries model and reduced the security of the multiple discrete logarithm problems in this model.

For the DDH problem in the auxiliary input model, Corrigan-Gibbs and Kogan showed the bound $ST^2 = \widetilde{\Omega}(\varepsilon^2 N)$ for any generic algorithm solving the problem with a probability of at least $1/2 + \varepsilon$. There is a gap between this Corrigan-Gibbs and Kogan bound and the best-known preprocessing attack for the DDH problem. At the same time, they propose a new attack for square DDH (distinguishing g, g^x, g^{x^2} from g, g^x, g^y for random $x, y \sim \mathbb{Z}_N$) showing that, unlike in the non-preprocessing setting, square DDH is less secure than the discrete logarithm problem. Their result motivates the question of whether DDH is as hard as the discrete logarithm problem in this setting. Corrigan-Gibbs and Kogan's results are based on the incompressibility argument and a non-trivial batching technique. In the same year, Coretti et al. in [CDG18], using their refined presampling techniques (first developed by Unruh [Unr07] for the Random Oracle model) proved the same bound. In the same work, Coretti et al. showed hardness bounds for several other security applications in the Auxiliary Input Generic Group model, including the discrete logarithm problem and the DDH problem, using their presampling technique.

Gravin et al.[GGKL21], inspired by the elegant and short application of union-bounds in Impagliazzo's proof in [Imp11], reduced the security for algorithms with S-bit advice to showing a roughly 2^{-S} concentration bound on the advantage of algorithms with no advice. Thus, they derived concentration bounds for the k-wise XOR version of the problem. For a decision problem G, if no algorithm with zero advice and kT queries can solve the k-wise XOR version of problem G with a probability of more than $1/2 + \varepsilon^k$, then for any $k \leq S + \log N$, no algorithm with S-bit advice making T online queries can solve G with a probability of more than ε' where $\varepsilon' = 2\varepsilon + O(\sqrt{S/N})$.

We note several previous works [Hel80, FN91, DGK17, CDG18, CDGS18, CK19, CHM20, ACDW20, CGLQ20, Liu23, AGL22, GK22, FGK22, FGK23, GGPS23, ADGL23, GLLZ21] that have explored the time-space lower bounds for various cryptographic primitives and applications in idealized models.

2 Preliminaries

In this paper, we write $\mathbb{Z}_N$ for $N \geq 1$ to denote the ring of integers modulo N. Let $[n] = \{1, \ldots, n\}$ for any integer $n \geq 1$. We use the notation $\widetilde{O}$, $\widetilde{\Omega}$, and $\widetilde{\Theta}$ to denote asymptotic bounds up to a poly-log factor. An algorithm $\mathcal{A}$ taking an input the value x is denoted $\mathcal{A}(x)$. Given a set S, let $x \leftarrow S$ denote that x is sampled randomly from the uniform distribution over S.

Next, we formally define the idealized models used to prove our results.

Definition (Generic Group Model (GGM) [Sho97, CK18])**.** *For any cyclic group G of order N with generator g, let $\sigma : \mathbb{Z}_N \to \mathcal{L}$ be a random injective "labeling" function, where $\mathcal{L}$ is an arbitrary set of "labels" for encoding the set of group elements. We identify group elements with their images $\{g^i\}_{i=0}^{N-1} \cong \{\sigma(i)\}_{i=0}^{N-1}$. As such, we will say that an element $i \in \mathbb{Z}_N$ is the **discrete logarithm** of the label $\sigma(i)$. The **generic group oracle** $\mathcal{O}_\sigma : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ is a function that takes two elements $x, y \in \mathcal{L}$ as input and response as follows:*

- *If there exists $i, j \in \mathbb{Z}_N$ such that $\sigma(i) = x$ and $\sigma(j) = y$ respectively, then the oracle responds with $\sigma(i + j)$, where addition is modulo the group order N.*
- *If there exists no such i or j, then the oracle responds with $\perp$.*

Definition (Generic Algorithm)**.** *A **generic algorithm** over a label set $\mathcal{L}$ for $\mathbb{Z}_N$ is a probabilistic algorithm $\mathcal{A}$ which has access to the generic group oracle $\mathcal{O}_\sigma$ and takes as input a list of labels $\{\sigma(x_i)\}_{i=1}^{\ell}$. Furthermore, for such an algorithm $\mathcal{A}$, we define the **query complexity** of $\mathcal{A}$ to be the number of calls the algorithm makes to the oracle $\mathcal{O}_\sigma$.*

Definition (Generic Algorithm with Preprocessing)**.** *A **generic algorithm with preprocessing** over a label set $\mathcal{L}$ for $\mathbb{Z}_N$ consists of a pair of generic group algorithms $(\mathcal{A}_0, \mathcal{A}_1)$:*

- *Offline stage: The algorithm $\mathcal{A}_0$ takes $\sigma(1)$ as input, can make an unlimited number of queries to the oracle $\mathcal{O}_\sigma$, and computes an advice string $s_\sigma \in \{0,1\}^*$.*
- *Online stage: The algorithm $\mathcal{A}_1$ takes an advice string s_σ output in the offline stage and a list of labels $\{\sigma(x_i)\}_{i=1}^{\ell}$ as input, makes some number of queries to oracle $\mathcal{O}_\sigma$, and outputs some result. The query complexity of the online algorithm $\mathcal{A}_1$ is called the **online query complexity**.*

In particular, we focus on generic algorithms with preprocessing where the advice string s_σ is limited to S-bits, *i.e.* $s_\sigma \in \{0,1\}^S$ and with online query complexity T. Here S, T may be functions of the group order N.

Definition (Hyperplane Query Model (HQM) [Yun15])**.** *Let N be a prime number and $\mathbb{Z}_N^\ell$ be the ℓ-dimensional affine space over finite field $\mathbb{Z}_N$. Let $X_1, \ldots, X_\ell$ be the canonical coordinates of $\mathbb{Z}_N^\ell$. Let Λ be the set of all affine hyperplanes in $\mathbb{Z}_N^\ell$. Then an affine hyperplane $H \in \Lambda$ can be described as the set of points in $\mathbb{Z}_N^\ell$ satisfying linear equation $\sum_{i=1}^{\ell} \alpha_i X_i = \alpha_0$ for some $\{\alpha_i\}_{i=0}^{\ell} \subseteq \mathbb{Z}_N$. Let $\mathbf{w} \in \mathbb{Z}_N^\ell$ be a point in the affine space. A **hyperplane query oracle** $\mathcal{H}_{\mathbf{w}} : \Lambda \to \{0,1\}$ is a function that takes as input a hyperplane H and response as follows:*

$$\mathcal{H}_{\mathbf{w}}(H) = \begin{cases} 1, & \text{if } \mathbf{w} \in H \\ 0, & \text{if } \mathbf{w} \notin H \end{cases}$$

Definition (Hyperplane Query Algorithm)**.** *A **hyperplane query algorithm** for $\mathbb{Z}_N^\ell$ is a probabilistic algorithm $\mathcal{B}$ that has access to the oracle $\mathcal{H}_{\mathbf{w}}$. The **hyperplane query complexity** of such an algorithm is the number of calls made by the algorithm to the hyperplane query oracle $\mathcal{H}_{\mathbf{w}}$.*

3 Tight Time-Space Tradeoffs for DDH

3.1 Reducing Preprocessing DDH to k-XOR DDH Problem

In this section, we show that to prove a security bound on the preprocessing DDH problem, it suffices to prove a security bound on the k-XOR DDH problem.

Lemma 1. *Let N be a prime and $S \geq 2$ be an even integer. For any generic algorithm with preprocessing $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{A}_0$ outputs an S-bit advice string, $\mathcal{A}_1$ makes at most T oracle queries, and*

$$\boldsymbol{Adv}_{\mathcal{A}} := 2\mathbb{P}\left[\mathcal{A}_1^{\mathcal{O}_\sigma}\left(\mathcal{A}_0^{\mathcal{O}_\sigma}(\sigma(1)), \sigma(1), \sigma(x), \sigma(y), \sigma(u)\right) = b\right] - 1,$$

where $x, y, z \leftarrow \mathbb{Z}_N$, $b \leftarrow \{0,1\}$, $u = bz + (1-b)xy$, and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ is a random injective function, there exists a generic algorithm $\mathcal{B}$ which makes at most ST queries, such that for

$$\boldsymbol{Adv}_{\mathcal{B}} := 2\mathbb{P}\left[\mathcal{B}^{\mathcal{O}_\sigma}\left(\sigma(1), \left\{\sigma(x_i), \sigma(y_i), \sigma(u_i)\right\}_{i=1}^{S}\right) = \bigoplus_{i=1}^{S} b_i\right] - 1,$$

where $\{x_i, y_i, z_i\}_{i=1}^{S} \leftarrow \mathbb{Z}_N^{3S}$, $\{b_1, \ldots, b_S\} \leftarrow \{0,1\}^S$, $u_i = b_i z_i + (1-b_i)x_i y_i$ and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ is a random injective function, it holds that

$$\boldsymbol{Adv}_{\mathcal{B}} \geq \frac{\boldsymbol{Adv}_{\mathcal{A}}^S}{2^S}.$$

Proof. Without loss of generality, we assume that $\mathcal{A}$ is deterministic otherwise we fix the best randomness of $\mathcal{A}$. Consider the following algorithm $\mathcal{B}$ based on $\mathcal{A}$: sample a random advice string $s \leftarrow \{0,1\}^S$, then for all $i \in [S]$, compute

$$\hat{b}_i^{(\sigma,s)} := \mathcal{A}_1^{\mathcal{O}_\sigma}\big(s, \sigma(1), \sigma(x_i), \sigma(y_i), \sigma(u_i)\big),$$

and output a single bit $\hat{b} := \bigoplus_{i=1}^S \hat{b}_i^{(\sigma,s)}$. It is clear that $\mathcal{B}$ makes at most ST queries. Now, we show that such an algorithm has the desired advantage. Fix a labeling function σ and some advice string $a \in \{0,1\}^S$. The advantage of $\mathcal{A}$ and $\mathcal{B}$ on a are

$$\varepsilon_{\mathcal{A}}(\sigma, a) := 2\mathbb{P}\Big[\hat{b}^{(\sigma,a)} = b\Big] - 1 = \mathbb{E}\Big[(-1)^{\hat{b}^{(\sigma,a)} \oplus b}\Big],$$

$$\varepsilon_{\mathcal{B}}(\sigma, a) := 2\mathbb{P}\left[\bigoplus_{i=1}^S \hat{b}_i^{(\sigma,a)} = \bigoplus_{i=1}^S b_i\right] - 1 = \mathbb{E}\Big[(-1)^{\bigoplus_{i=1}^S b_i \oplus \hat{b}_i^{(\sigma,a)}}\Big].$$

Because σ, a are fixed and $\{x_i, y_i, u_i\}$ are independently sampled, we can derive that

$$\varepsilon_{\mathcal{B}}(\sigma, a) = \mathbb{E}\Big[\Pi_{i=1}^S (-1)^{\hat{b}_i^{(\sigma,a)} \oplus b_i}\Big] = \Pi_{i=1}^S \mathbb{E}\Big[(-1)^{\hat{b}_i^{(\sigma,a)} \oplus b_i}\Big] = \varepsilon_{\mathcal{A}}(\sigma, a)^S \ .$$

For a fixed σ, let $s_\sigma := \mathcal{A}_0^{\mathcal{O}_\sigma}\big(\sigma(1)\big)$. Note that

$$\boldsymbol{Adv}_{\mathcal{B}} = \mathbb{E}_{\sigma,a}\big[\varepsilon_{\mathcal{B}}(\sigma, a)\big] = \mathbb{E}_{\sigma,a}\big[\varepsilon_{\mathcal{A}}(\sigma, a)^S\big] \geq \frac{1}{2^S}\mathbb{E}_\sigma\big[\varepsilon_{\mathcal{A}}(\sigma, s_\sigma)^S\big] \geq \frac{1}{2^S}\mathbb{E}_\sigma\big[\varepsilon_{\mathcal{A}}(\sigma, s_\sigma)\big]^S = \frac{1}{2^S}\boldsymbol{Adv}_{\mathcal{A}}^S \ ,$$

where the first inequality follows from a being uniformly sampled (so $\Pr[a = s_\sigma] = 1/2^S$) and S being even (so $\varepsilon_{\mathcal{A}}(\sigma, a)^S \geq 0$, $\forall a \in \{0,1\}^S$), and the second inequality follows from taking the expectation over σ and using Jensen's inequality. □

3.2 Reducing k-XOR DDH from GGM to HQM

In this section, we prove a reduction of the k-XOR DDH problem from GGM to HQM which increases in query complexity while maintaining the advantage. Here, we effectively create a hyperplane simulation of a k-XOR algorithm in GGM which exploits the lazy sampling property of the labeling function to keep the number of hyperplane queries low.

Lemma 2. *Let N be a prime, and $k \geq 1$ be an integer. For any generic algorithm $\mathcal{B}$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{B}$ makes at most Q oracle queries, and*

$$\boldsymbol{Adv}_{\mathcal{B}} := 2\mathbb{P}\left[\mathcal{B}^{\mathcal{O}_\sigma}\Big(\sigma(1), \sigma(\mathbf{w})\Big) = \bigoplus_{i=1}^k b_i\right] - 1,$$

where $\sigma(\mathbf{w}) := \big\{\sigma(x_i), \sigma(y_i), \sigma(u_i)\big\}_{i=1}^k$, $u_i = b_i z_i + (1 - b_i)x_i y_i$, $\{x_i, y_i, z_i\}_{i=1}^k \leftarrow \mathbb{Z}_N^{3k}$, $\{b_1, \ldots, b_k\} \leftarrow \{0,1\}^k$, and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ is a random injective function, there exists a hyperplane query algorithm $\mathcal{C}$ which makes at most $2(3k + 2Q + 1)(3k + 3Q + 1)$ queries, and

$$\boldsymbol{Adv}_{\mathcal{C}} := 2\mathbb{P}\left[\mathcal{C}^{\mathcal{H}_{\mathbf{w}}} = \bigoplus_{i=1}^k b_i\right] - 1 = \boldsymbol{Adv}_{\mathcal{B}},$$

where $\mathbf{w} := (x_1, y_1, u_1, \ldots, x_k, y_k, u_k)$, $u_i = b_i z_i + (1 - b_i)x_i y_i$, $\{x_i, y_i, z_i\}_{i=1}^k \leftarrow \mathbb{Z}_N^{3k}$, and $\{b_1, \ldots, b_k\} \leftarrow \{0,1\}^k$.

Proof. We show how to simulate $\mathcal{B}^{\mathcal{O}_\sigma}(\sigma(1), \sigma(\mathbf{w}))$ given only oracle access to $\mathcal{H}_{\mathbf{w}}$. We rely on the fact that a random injective function σ can be sampled lazily (*i.e.* new input-output pairs can be generated on the fly) with limited information from previously generated pairs. Specifically, the simulation algorithm $\mathcal{C}$ will maintain a global list $\mathbf{Cur}_\sigma := \big\{(L_i, s_i)\big\}_{i\geq 0}$ to help record generated input-output pairs of σ during a lazy sampling process, where L_i is a multi-linear polynomial in variables $\{X_i, Y_i, U_i\}_{i=1}^k$ representing an *implicit* preimage of s_i so that $s_i = \sigma\big(L_i(\mathbf{w})\big)$. Additionally, $\mathcal{C}$ will maintain a global set $I \subseteq \mathbb{Z}_N$ for tracking *explicitly* generated preimages, and a global set J for the image set of σ.

The algorithm $\mathcal{C}$ is initialized by randomly choosing a set $J \subseteq \mathcal{L}$ of size N as the image set of σ and setting $\mathbf{Cur}_\sigma = \varnothing$, $I = \varnothing$. It then performs the following simulation with the help of two subroutines **FindImage** and **FindPreimage** defined later:

1. Compute challenges $\hat{\sigma}_0, \ldots, \hat{\sigma}_{3k}$ as $\mathbf{FindImage}(H)$ for every polynomial $H \in \{1, X_1, Y_1, U_1, \ldots X_k, Y_k, U_k\}$.
2. Run $\mathcal{B}$ on the input $\hat{\sigma}_0, \ldots, \hat{\sigma}_{3k}$ with the following oracle $\hat{\mathcal{O}}_\sigma(t_1, t_2)$: If $t_1 \notin J$ or $t_2 \notin J$, return $\perp$, otherwise, return

$$\mathbf{FindImage}\big(\mathbf{FindPreimage}(t_1) + \mathbf{FindPreimage}(t_2)\big).$$

Intuitively, the two subroutines simulate oracle accesses to a (lazily sampled) σ and its inverse function σ^{-1}. The two subroutines first determine whether the queried input or output has appeared in $\mathbf{Cur}_\sigma$, if yes, then return the assigned (implicit) preimage or image accordingly, and otherwise generate a new pair on the fly. The main observation is that searching existence (in the form of a polynomial) and lazily generating new images or preimages can be done with only a few hyperplane queries. We use $\mathsf{Pre}(\mathbf{Cur}_\sigma)$ (respectively $\mathsf{Im}(\mathbf{Cur}_\sigma)$) to denote the set of L_i's (respectively s_i's) appearing in $\mathbf{Cur}_\sigma$. Formally,

- $\mathbf{FindImage}(H)$:
  1. If $\mathcal{H}_{\mathbf{w}}(H - L) = 1$ for some $(L, s) \in \mathbf{Cur}_\sigma$, return s.
  2. Otherwise, sample $s_{new} \leftarrow J \setminus \mathsf{Im}(\mathbf{Cur}_\sigma)$, add (H, s_{new}) to $\mathbf{Cur}_\sigma$, and return s_{new}.
- $\mathbf{FindPreimage}(t)$:
  1. If $t = s$ for some $(L, s) \in \mathbf{Cur}_\sigma$, return L.
  2. Otherwise, sample $\beta \leftarrow \mathbb{Z}_N \setminus I$ and let $I = I \cup \{\beta\}$ until return:
     - If $\mathcal{H}_{\mathbf{w}}(L - \beta) = 1$ for some $(L, s) \in \mathbf{Cur}_\sigma$, continue.
     - Otherwise, let $L_{new} = \beta$, add (L_{new}, t) to $\mathbf{Cur}_\sigma$, return L_{new}.

We claim that such an algorithm $\mathcal{C}$ fulfills the criteria in the theorem.

Query Complexity: The algorithm $\mathcal{C}$ makes at most $3k+3Q+1$ calls to **FindImage** and **FindPreimage**, each call contributing at most 1 new pair to $\mathbf{Cur}_\sigma$. Therefore the size of $\mathbf{Cur}_\sigma$ and the set of explicitly generated preimages I are at any time upper bounded by $3k + 3Q + 1$. The calls of **FindImage** search the entire list $\mathbf{Cur}_\sigma$ for $(3k + Q + 1)$ times, and the calls of **FindPreimage** search $\mathbf{Cur}_\sigma$ for at most $|I|$ times at the end of the simulation. Because searching in $\mathbf{Cur}_\sigma$ takes at most $(3k + 3Q + 1)$ hyperplane queries, the simulation makes at most $(|I| + 3k + Q + 1)(3k + 3Q + 1) \leq 2(3k + 2Q + 1)(3k + 3Q + 1)$ hyperplane queries.

Correctness: We will show that, for any fixed $\mathbf{w}$ (and $b_1, \ldots, b_k \in \{0, 1\}$), $\mathcal{C}^{\mathcal{H}_{\mathbf{w}}}$ distributes the same as $\mathcal{B}^{\mathcal{O}_\sigma}(\sigma(1), \sigma(\mathbf{w}))$ over the randomness of $\sigma : \mathbb{Z}_N \to \mathcal{L}$ and the internal randomness of $\mathcal{B}$ and $\mathcal{C}$. Notice that $\mathcal{B}^{\mathcal{O}_\sigma}(\sigma(1), \sigma(\mathbf{w}))$ can be computed by the following process $\mathcal{D}$: first randomly choose a set $J \subseteq \mathcal{L}$ of size N as the simulated image set of σ and set the list of generated input-output pairs $\mathbf{List}_\sigma = \varnothing$, then perform the following,

1. Compute challenges $\sigma_0, \ldots, \sigma_{3k}$ as $\mathbf{Lazy}_\sigma(\alpha)$ for every $\alpha \in \{1, x_1, y_1, u_1, \ldots x_k, y_k, u_k\}$.

2. Run $\mathcal{B}$ on the input $\sigma_0, \ldots, \sigma_{3k}$ with access to $\mathbf{Lazy}\mathcal{O}_\sigma(t_1, t_2)$: If $t_1 \notin J$ or $t_2 \notin J$, return $\perp$; otherwise, return

$$\mathbf{Lazy}_\sigma\big(\mathbf{Lazy}_{\sigma^{-1}}(t_1) + \mathbf{Lazy}_{\sigma^{-1}}(t_2)\big),$$

where $\mathbf{Lazy}_\sigma(\alpha)$ returns s (respectively $\mathbf{Lazy}_{\sigma^{-1}}(s)$ returns α) if (α, s) appears in $\mathbf{List}_\sigma$, and otherwise samples a new s from $J \setminus \mathsf{Im}(\mathbf{List}_\sigma)$ (respectively a new α from $\mathbb{Z}_N \setminus \mathsf{Pre}(\mathbf{List}_\sigma)$), then adds the new pair (α, s) to $\mathbf{List}_\sigma$.

It suffices to prove the following claim:

Claim 1. *For any fixed* $\mathbf{w}$ *and* J*, the following two distributions* $(\hat{\sigma}_0, \ldots, \hat{\sigma}_{3k}, \hat{\sigma}_{3k+1}, \ldots, \hat{\sigma}_{3k+Q}, \mathbf{Cur}_\sigma^{3k+Q}(\mathbf{w}))$ *and* $(\sigma_0, \ldots, \sigma_{3k}, \sigma_{3k+1}, \ldots, \sigma_{3k+Q}, \mathbf{List}_\sigma^{3k+Q})$ *are distributed identically, where* σ_{3k+i} *(respectively* $\hat{\sigma}_{3k+i}$*) is the* i*-th response from* $\mathbf{Lazy}\mathcal{O}_\sigma$ *(respectively* $\hat{\mathcal{O}}_\sigma$*), and* $\mathbf{Cur}_\sigma^i(\mathbf{w}) := \{(L_j(\mathbf{w}), \hat{s}_j)\}_{j \geq 0}$ *(respectively* $\mathbf{List}_\sigma^i := \{(\alpha_j, s_j)\}_{j \geq 0}$*) denote the maintained list after* $\hat{\sigma}_i$ *(respectively* σ_i*) has been generated and all polynomials (if applicable) have been evaluated at input* $\mathbf{w}$*.*

Proof. We will prove by induction. $(\hat{\sigma}_0, \mathbf{Cur}_\sigma^0(\mathbf{w})) = (\hat{s}, (1, \hat{s}))$, and $(\sigma_0, \mathbf{List}_\sigma^0) = (s, (1, s))$ where both s and s' are uniformly distributed over J. Therefore the above claim holds for $i = 0$. For $i \geq 1$, assume that the claim holds for $0 \leq j < i$. Now conditioning on, $\sigma_j = \hat{\sigma}_j$ for any $j < i$ and $\mathbf{Cur}_\sigma^{i-1}(\mathbf{w}) = \mathbf{List}_\sigma^{i-1}$, we prove that $(\hat{\sigma}_i, \mathbf{Cur}_\sigma^i(\mathbf{w}))$ distributes the same as $(\sigma_i, \mathbf{List}_\sigma^i)$.

If $i \leq 3k$, then $\sigma_i = \mathbf{Lazy}_\sigma(\alpha)$ and $\hat{\sigma}_i = \mathbf{FindImage}(H)$ for some $\alpha \in \{x_1, y_1, u_1, \ldots, x_k, y_k, u_k\}$ and $H \in \{X_1, Y_1, U_1 \ldots, X_K, Y_K, U_K\}$ such that $H(\mathbf{w}) = \alpha$. Given that $\mathbf{Cur}_\sigma^{i-1}(\mathbf{w}) = \mathbf{List}_\sigma^{i-1}$, (α, s) appears in $\mathbf{List}_\sigma^{i-1}$ if and only if there exists an L such that (L, s) appears in $\mathbf{Cur}_\sigma^{i-1}$ and $H(\mathbf{w}) = L(\mathbf{w}) = \alpha$ (i.e., $\mathcal{H}_\mathbf{w}(H - L) = 1$). If (α, s) appears in $\mathbf{List}_\sigma^{i-1}$, then $\sigma = \hat{\sigma}_i = s$ and $\mathbf{Cur}_\sigma^i(\mathbf{w}) = \mathbf{Cur}_\sigma^{i-1}(\mathbf{w}) = \mathbf{List}_\sigma^{i-1} = \mathbf{List}_\sigma^i$. If α is new, then $(\sigma_i, \mathbf{List}_\sigma^i) = (s, \mathbf{List}_\sigma^{i-1} \cup \{(\alpha, s)\})$ and $(\hat{\sigma}_i, \mathbf{Cur}^i(\mathbf{w})) = (\hat{s}, \mathbf{Cur}^{i-1}(\mathbf{w}) \cup \{(H(\mathbf{w}), \hat{s})\})$ are distributed identically because both $s, \hat{s}$ are uniformly distributed over $J \setminus \mathsf{Im}(\mathbf{List}_\sigma^{i-1})$ and $H(\mathbf{w}) = \alpha$.

If $3k < i \leq 3k + Q$, then $\sigma_i = \mathbf{Lazy}\mathcal{O}_\sigma(t_1, t_2)$ and $\hat{\sigma}_i = \hat{\mathcal{O}}_\sigma(t_1, t_2)$ for some t_1 and t_2 computed based on $\sigma_0, \ldots, \sigma_{i-1}$ and the internal randomness of $\mathcal{B}$ (here we implicitly use that $\hat{\sigma}_j = \sigma_j$ for $j < i$ so the i-th query admits the same conditional distribution in both simulations). Notice that $\mathbf{Cur}_\sigma^{i-1}(\mathbf{w}) = \mathbf{List}_\sigma^{i-1}$. If t_1 or t_2 is not in J, then $\sigma = \hat{\sigma} = \perp$ and no updates are made to either list. If $t_1 \in J \setminus \mathsf{Im}(\mathbf{List}_\sigma^{i-1})$, then (α_1, t_1) will be added to $\mathbf{List}_\sigma^{i-1}$ by calling $\mathbf{Lazy}_{\sigma^{-1}}(t_1)$, and (α_1', t_1) will be added into $\mathbf{Cur}_\sigma^{i-1}$ by calling $\mathbf{FindPreimage}(t_1)$ where both α_1, α_1' are uniformly distributed over $\mathbb{Z}_N \setminus \mathsf{Pre}(\mathbf{List}_\sigma^{i-1})$. Similarly, if $t_2 \in J \setminus \mathsf{Im}(\mathbf{List}_\sigma^{i-1})$, we can update both lists in a similar way. Moreover, importantly, the two updated lists, denoted as $\mathbf{List}_\sigma^*$ and $\mathbf{Cur}_\sigma^*(\mathbf{w})$ (with t_1, t_2 in the list now), are still distributed identically. Conditioning on that $\mathbf{List}_\sigma^* = \mathbf{Cur}_\sigma^*(\mathbf{w})$ and t_1, t_2 appear in $\mathbf{List}_\sigma^*$, we have that $L_1(\mathbf{w}) = \alpha_1$ and $L_2(\mathbf{w}) = \alpha_2$ for some (L_1, t_1), $(L_2, t_2) \in \mathbf{Cur}_\sigma^*$ and some $(\alpha_1, t_1), (\alpha_2, t_2) \in \mathbf{List}_\sigma^*$. If for some s, $(\alpha_1 + \alpha_2, s) \in \mathbf{List}_\sigma^*$, then $\sigma = \hat{\sigma} = s$ and no updates are made to either list. Otherwise, $(\alpha_1 + \alpha_2, s)$ will be added to $\mathbf{List}_\sigma^*$ and $(L_1 + L_2, s')$ will be added to $\mathbf{Cur}_\sigma^*$ where both s, s' are distributed uniformly over $J \setminus \mathsf{Im}(\mathbf{List}_\sigma^*)$ and $L_1(\mathbf{w}) + L_2(\mathbf{w}) = \alpha_1 + \alpha_2$. For both cases, $(\hat{\sigma}_i, \mathbf{Cur}_\sigma^i(\mathbf{w}))$ is distributed the same as $(\sigma_i, \mathbf{List}_\sigma^i)$. □

□

3.3 Hardness of k-XOR DDH Problem in the HQM

In this section, we prove the generic hardness of k-XOR DDH problem in the hyperplane query model. The following lemma will be proven by induction over the number of DDH instances and queries needed by the optimal algorithm for any given case.

Lemma 3. *Let* N *be a prime, and* $k \geq 1$ *be an integer. For any hyperplane query algorithm* $\mathcal{C}$ *which makes at most* q *queries, it holds that*

$$\boldsymbol{Adv}_\mathcal{C} := 2\mathbb{P}_\mathbf{w}\left[\mathcal{C}^{\mathcal{H}_\mathbf{w}} = \bigoplus_{i=1}^k b_i\right] - 1 \leq \left(\frac{12q}{Nk} + \frac{12}{N}\right)^k,$$

where $\mathbf{w} = (x_1, y_1, u_1, \ldots, x_k, y_k, u_k)$*,* $u_i = b_i z_i + (1 - b_i) x_i y_i$*,* $\{x_i, y_i, z_i\}_{i=1}^k \leftarrow \mathbb{Z}_N^{3k}$*, and* $\{b_1, \ldots, b_k\} \leftarrow \{0, 1\}^k$*.*

For any integer $k \geq 1$ and $q \geq 0$, we let $\mathcal{C}_{k,q}$ denote the hyperplane query algorithm with q queries for the k-XOR DDH problem in the HQM which maximizes the following quantity

$$\boldsymbol{Adv}(k,q) := \mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = \bigoplus_{i=1}^{k} b_i\right] - \mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} \neq \bigoplus_{i=1}^{k} b_i\right].$$

Similarly for any $k, q \geq 1$, we consider the optimal algorithm $\mathcal{D}_{k,q}$, making q queries where its first hyperplane query is denoted by H, which maximizes the following quantity

$$\Delta(k,q) := \mathbb{P}_{\mathbf{w}}\left[\mathcal{D}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = \bigoplus_{i=1}^{k} b_i \cap E_H\right] - \mathbb{P}_{\mathbf{w}}\left[\mathcal{D}_{k,q}^{\mathcal{H}_{\mathbf{w}}} \neq \bigoplus_{i=1}^{k} b_i \cap E_H\right],$$

where $E_H := \{\mathcal{H}_{\mathbf{w}}(H) = 1\}$. This is to say that $\mathcal{D}_{k,q}$ is the optimal q query algorithm in the case that the first query output is 1.

Our proof of the lemma will follow by induction on (k, q). We first state our claims (see claim 2 and 3) for the bounds on the $\boldsymbol{Adv}$ and Δ functions, which we will be using in the inductive step, present our inductive proof of the lemma, and then prove the claims. We consider our proof for claim 3 to be one of the main technical contributions of this work, which intuitively states that the advantage of any algorithm for k-XOR DDH problem, making a single successful hyperplane query (oracle output 1), is bounded by the product of the probability of finding the one successful hyperplane query and the advantage of some algorithm for $(k-1)$-XOR DDH problem.

Claim 2. *For any $k \geq 1$, and $q \geq 1$, $\boldsymbol{Adv}(k,q) \leq \boldsymbol{Adv}(k,q-1) + 2\Delta(k,q)$.*

Claim 3. *For any $k \geq 2$ and $q \geq 1$, $\Delta(k,q) \leq 3\boldsymbol{Adv}(k-1,q-1)/2N + \Delta(k-1,q)/2N$.*

Proof of Lemma 3. Given Claim 2 and 3, we prove the following by induction on (k, q).

$$\boldsymbol{Adv}(k,q) \leq \left(\frac{4}{N}\right)^k \binom{q+k}{k}, \tag{1}$$

then Lemma 3 follows from the fact that $\binom{q+k}{k} \leq (3(q+k)/k)^k$.

The base cases $(k, 0)$ and $(1, q)$. For any $k \geq 1$, (1) holds trivially for $(k, 0)$ by the fact that $\boldsymbol{Adv}(k, 0) = 0$. For any $q \geq 1$, by representing the first hyperplane query $H \in \mathbb{Z}_N[x_1, y_1, u_1]$ as a polynomial with coefficients $\alpha, \beta, \gamma, \lambda \in \mathbb{Z}_N$, we have that

$$\Delta(1,q) \leq \mathbb{P}_{\mathbf{w}}\left[\mathcal{D}_{1,q}^{\mathcal{H}_{\mathbf{w}}} = b \cap \mathcal{H}_{\mathbf{w}}(H) = 1\right] \leq \mathbb{P}_{\mathbf{w}}[\mathcal{H}_{\mathbf{w}}(H) = 1] = \mathbb{P}_{\mathbf{w}}\left[\alpha x_1 + \beta y_1 + \gamma u_1 = \lambda\right] \leq \frac{3}{2N} \tag{2}$$

where the last inequality is because by the Schwartz–Zippel lemma [Sch80, Zip79], the hyperplane query H holds with probability at most $1/N$ conditioning on $u_1 = z_1$ and holds with probability at most $2/N$ conditioning on $u_1 = x_1 y_1$ (as $x_1, y_1, z_1 \leftarrow \mathbb{Z}_N$ and H is not a constant polynomial by the optimality of $\mathcal{D}_{1,q}$). Then, by applying Claim 2 with $k = 1$ recursively together with (2), we obtain that

$$\boldsymbol{Adv}(1,q) \leq \boldsymbol{Adv}(1,0) + 2\sum_{i=0}^{q-1} \Delta(1,q-i) = 2\sum_{i=0}^{q-1} \Delta(1,q-i) \leq \frac{3q}{N} \leq \frac{4}{N}\binom{q+1}{1}.$$

The inductive step. For any fixed $k \geq 2$ and $q \geq 1$, assume that (1) holds for all tuples (ℓ, r) such that $\ell \in \{1, \ldots, k-1\}$, $r \in \{0, \ldots, q-1\}$. Then, recursively apply Claim 3 $(k-1)$ times to the final term in Claim 2, yielding

$$\begin{aligned}
\boldsymbol{Adv}(k,q) &\leq \boldsymbol{Adv}(k,q-1) + 3\left(\sum_{i=1}^{k-1} \frac{\boldsymbol{Adv}(k-i,q-1)}{2^{i-1}N^i}\right) + \frac{\Delta(1,q)}{2^{k-2}N^{k-1}} \\
&\leq \boldsymbol{Adv}(k,q-1) + 3\left(\sum_{i=1}^{k-1} \frac{\boldsymbol{Adv}(k-i,q-1)}{N^i}\right) + \frac{3}{N^k}
\end{aligned}$$

where the last inequality uses (2). Then, applying the inductive assumption,

$$\begin{aligned}
\boldsymbol{Adv}(k,q) &\le \left(\frac{4}{N}\right)^k \binom{q+k-1}{k} + 3\left(\sum_{i=1}^{k-1} \frac{1}{N^i} \cdot \frac{4^{k-i}}{N^{k-i}} \binom{q+(k-i)-1}{k-i}\right) + \frac{3}{N^k} \\
&\le \left(\frac{4}{N}\right)^k \left(\binom{q+k-1}{k} + \binom{q+k-1}{k-1} \sum_{i=1}^{k} \frac{3}{4^i}\right) \\
&\le \left(\frac{4}{N}\right)^k \left(\binom{q+k-1}{k} + \binom{q+k-1}{k-1}\right) \\
&= \left(\frac{4}{N}\right)^k \binom{q+k}{k}
\end{aligned}$$

where the second inequality uses the fact that $\binom{q+i-1}{i-1} \le \binom{q+k-1}{k-1}$ for $1 \le i \le k-1$, the third inequality uses the fact that $\sum_{i=1}^{k}(1/4^i) \le 1/3$, and the last equality uses the fact that $\binom{q+k}{k} = \binom{q+k-1}{k-1} + \binom{q+k-1}{k}$.

□

Proof of Claim 2. For any $k, q \ge 1$, let $\mathcal{C}_{k,q}$ be as in the definition of $\boldsymbol{Adv}(k,q)$ above, and let H denote the first hyperplane query made by $\mathcal{C}_{k,q}$. We denote the XORed bits of k DDH instances by b, i.e., $b = \bigoplus_{i=1}^{k} b_i$.

It is useful to consider a hyperplane algorithm as a depth-q decision tree, with each branch corresponding to the output of a single oracle query. Once the branch is selected at the root of a depth-q tree (depending on the response to the first query), the remaining depth-$(q-1)$ subtree corresponds to a sub-algorithm making $q-1$ queries. When analyzing $\mathcal{C}_{k,q}$, we define $\underline{\mathcal{C}}_{k,q-1}$ to be the sub-algorithm that gets executed when $\mathcal{H}_{\mathbf{w}}(H) = 0$ and $\overline{\mathcal{C}}_{k,q-1}$ to be the sub-algorithm that gets executed when $\mathcal{H}_{\mathbf{w}}(H) = 1$. In other words, the algorithms $\underline{\mathcal{C}}_{k,q-1}$ and $\overline{\mathcal{C}}_{k,q-1}$ can be thought of as the "left" and "right" subtrees of $\mathcal{C}_{k,q}$ corresponding to the output of the first query H being 0 or 1 respectively. Note that by the definition of the subtrees,

$$\mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_H\right] = \mathbb{P}_{\mathbf{w}}\left[\overline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_H\right] \text{ and } \mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = b \cap \bar{E}_H\right] = \mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b \cap \bar{E}_H\right].$$

Thus, the law of total probability gives the relationship

$$\begin{aligned}
\mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = b\right] &= \mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_H\right] + \mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = b \cap \bar{E}_H\right] \\
&= \mathbb{P}_{\mathbf{w}}\left[\overline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_H\right] + \mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b \cap \bar{E}_H\right] \\
&= \mathbb{P}_{\mathbf{w}}\left[\overline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_H\right] + \mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b\right] - \mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_H\right].
\end{aligned}$$

Similarly,

$$\mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} \ne b\right] = \mathbb{P}_{\mathbf{w}}\left[\overline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} \ne b \cap E_H\right] + \mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} \ne b\right] - \mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} \ne b \cap E_H\right].$$

By the definition of $\Delta(k,q)$, we have that for any depth-$(q-1)$ subtree algorithm $C \in \{\underline{\mathcal{C}}_{k,q-1}, \overline{\mathcal{C}}_{k,q-1}\}$,

$$\left|\mathbb{P}_{\mathbf{w}}\left[\mathcal{C}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_H\right] - \mathbb{P}_{\mathbf{w}}\left[\mathcal{C}^{\mathcal{H}_{\mathbf{w}}} \ne b \cap E_H\right]\right| \le \Delta(k,q).$$

By the definition of $\boldsymbol{Adv}(k, q-1)$, we have that,

$$\left|\mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} = b\right] - \mathbb{P}_{\mathbf{w}}\left[\underline{\mathcal{C}}_{k,q-1}^{\mathcal{H}_{\mathbf{w}}} \ne b\right]\right| \le \boldsymbol{Adv}(k, q-1).$$

Therefore, by the definition of $\boldsymbol{Adv}(k,q)$ and the triangle inequality,

$$\boldsymbol{Adv}(k,q) := \left|\mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} = b\right] - \mathbb{P}_{\mathbf{w}}\left[\mathcal{C}_{k,q}^{\mathcal{H}_{\mathbf{w}}} \ne b\right]\right| \le 2\Delta(k,q) + \boldsymbol{Adv}(k,q-1).$$

□

Proof of Claim 3. Let $\mathcal{D}$ denote the sub-algorithm of that gets executed when $\mathcal{H}_{\mathbf{w}}(H) = 1$ by $\mathcal{D}_{k,q}$, and H denote the first hyperplane query made by $\mathcal{D}_{k,q}$, described by a linear equation $L(x_1, \ldots, u_k) = c$ for some constant c. Notice that $\mathcal{D}$ makes at most $q-1$ queries. Let $b := \bigoplus_{i=1}^{k} b_i$, we further denote for any events E_1, E_2,

$$\boldsymbol{Adv}_{\mathcal{D}}\big(E_1 \,\big|\, E_2\big) = \mathbb{P}_{\mathbf{w}}\big[\mathcal{D}^{\mathcal{H}_{\mathbf{w}}} = b \cap E_1 \,\big|\, E_2\big] - \mathbb{P}_{\mathbf{w}}\big[\mathcal{D}^{\mathcal{H}_{\mathbf{w}}} \neq b \cap E_1 \,\big|\, E_2\big] \ .$$

Thus, by definition $\Delta(k,q) = \boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, \varnothing\big)$. Because L is not a constant polynomial (by the optimality of $\mathcal{D}_{k,q}$), there must be an $i \in [k]$ such that the coefficients of x_i, y_i, u_i cannot be all zeros. Without loss of generality, we assume that $i = 1$, i.e. , the hyperplane query H can be written as

$$\alpha x_1 + \beta y_1 + \gamma u_1 + L'\big(\{x_i, y_i, u_i\}_{i=2}^{k}\big) = c$$

where $(\alpha, \beta, \gamma) \neq 0^3$ and L' is some linear polynomial in $\{x_i, y_i, u_i\}_{i=2}^{k}$. It suffices to show

$$\boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, b_1 = 1\big) \leq \frac{1}{N} \cdot \boldsymbol{Adv}(k-1, q-1) \ , \tag{3}$$

and

$$\boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, b_1 = 0\big) \leq \frac{2}{N} \cdot \boldsymbol{Adv}(k-1, q-1) + \frac{1}{N} \cdot \Delta(k-1, q) \ . \tag{4}$$

Then the desired conclusion follows from averaging over b_1 and applying (3) and (4).

When $b_1 = 1$, then $u_1 = z_1$ where z_1 is independent of $x_1, y_1, x_2, y_2, u_2, \ldots, x_k, y_k, u_k$. Without loss of generality, we assume that $\alpha \neq 0$ (the cases of $\beta \neq 0$ or $\gamma \neq 0$ are the same reasoning by conditioning on fixed values of other two variables),

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, b_1 = 1\big) &= \mathbb{E}_{\delta_1, \delta_2 \sim \mathbb{Z}_N}\big[\boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, y_1 = \delta_1 \cap z_1 = \delta_2, b_1 = 1\big)\big] \\
&= \mathbb{E}_{\delta_1, \delta_2 \sim \mathbb{Z}_N}\big[\boldsymbol{Adv}_{\mathcal{D}}\big(\,\big|\, E_H \cap y_1 = \delta_1 \cap z_1 = \delta_2 \cap b_1 = 1\big)\mathbb{P}\big[E_H \,\big|\, y_1 = \delta_1 \cap z_1 = \delta_2 \cap b_1 = 1\big]\big] \\
&= \frac{1}{N} \cdot \mathbb{E}_{\delta_1, \delta_2 \sim \mathbb{Z}_N}\big[\boldsymbol{Adv}_{\mathcal{D}}\big(\,\big|\, E_H \cap y_1 = \delta_1 \cap z_1 = \delta_2 \cap b_1 = 1\big)\big] \\
&\leq \frac{1}{N} \cdot \boldsymbol{Adv}(k-1, q-1)
\end{aligned}$$

where the third line is because that for $b_1 = 1$ and any fixed $y_1 = \delta_1, z_1 = \delta_2$, $(\alpha x_1 + \beta y_1 + \gamma z_1)$ is uniformly distributed over $\mathbb{Z}_N$ and independent from any $\{x_i, y_i, u_i\}_{i=2}^{k}$ so E_H holds with probability $1/N$; and the fourth line is because that further conditioning on that E_H happens, by substituting x_1, y_1, z_1 with either constants or linear polynomials in variables $\{x_i, y_i, u_i\}_{i=2}^{k}$ in its hyperplane queries, $\mathcal{D}$ becomes an algorithm for solving $(k-1)$-XOR DDH with $(q-1)$ queries, and thus has advantage bounded by $\boldsymbol{Adv}(k-1, q-1)$. (3) follows.

When $b = 0$, then $u_1 = x_1 y_1$. Without loss of generality, we assume that at least one of γ, α is non-zero (if not we work with β instead of $\gamma y_1 + \alpha$ in the following reasoning).Then we can describe H by the following equation

$$(\gamma y_1 + \alpha) x_1 + \beta y_1 + L'\big(\{x_i, y_i, u_i\}_{i=2}^{k}\big) = c \ .$$

By similar argument as before, for any δ_1 such that $\gamma\delta_1 + \alpha \neq 0$, we have

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, y_1 = \delta_1 \cap b_1 = 0\big) &= \boldsymbol{Adv}_{\mathcal{D}}\big(\,\big|\, E_H \cap y_1 = \delta_1 \cap b_1 = 0\big)\mathbb{P}\big[E_H \,\big|\, y_1 = \delta_1 \cap b_1 = 0\big] \\
&\leq \frac{1}{N} \cdot \boldsymbol{Adv}(k-1, q-1)
\end{aligned}$$

where the inequality is because for $\gamma\delta_1 + \alpha \neq 0$, $(\gamma\delta_1 + \alpha)x_1$ is uniformly and distributed over $\mathbb{Z}_N$ and independent from any $\{x_i, y_i, u_i\}_{i=2}^{k}$ so E_H holds with probability $1/N$; and further conditioning on that E_H happens, by replacing y_1,x_1,u_1 by either a constant or a linear polynomial only in variables $\{x_i, y_i, u_i\}_{i=2}^{k}$ (according to $y_1 = \delta_1$ and the equation by H and $u_1 = \delta_1 x_1$) in its hyperplane queries, $\mathcal{D}$ becomes an algorithm for $(k-1)$-XOR DDH problem, and thus has advantage bounded by $\boldsymbol{Adv}(k-1, q-1)$.

For δ' such that $\gamma\delta' + \alpha = 0$, let H' denote a new hyperplane query $L'\big(\{x_i, y_i, u_i\}_{i=2}^k\big) = c - \delta'\beta$. Then,

$$\begin{aligned}\boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, y_1 = \delta' \cap b_1 = 0\big) &= \boldsymbol{Adv}_{\mathcal{D}}\big(E_{H'} \,\big|\, y_1 = \delta' \cap b_1 = 0\big) \\ &= \mathbb{E}_{\delta_2 \sim \mathbb{Z}_N}\big[\boldsymbol{Adv}_{\mathcal{D}}\big(E_{H'} \,\big|\, y_1 = \delta' \cap x_1 = \delta_2 \cap b_1 = 0\big)\big] \\ &\leq \boldsymbol{Adv}(k-1, q-1) + \Delta(k-1, q)\end{aligned}$$

where the inequality is because by substituting $y_1 = \delta'$, $x_1 = \delta_2$ and $u_2 = \delta' \cdot \delta_2$, $\mathcal{D}$ becomes a $(k-1)$-XOR DDH hyperplane query algorithm in $\{x_i, y_i, u_i\}_{i=2}^k$. If E'_H is a non-trivial hyperplane query in $\{x_i, y_i, u_i\}_{i=2}^k$ (i.e., the degree of L' is non-zero), then its advantage is upper bounded by $\Delta(k-1, q)$ otherwise is at most $\boldsymbol{Adv}(k-1, q-1)$. Combining both cases and the fact that $\Pr[\gamma y_1 + \alpha = 0] \leq 1/N$ given $(\gamma, \alpha) \neq 0^2$.

$$\begin{aligned}\boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, b_1 = 0\big) &= \boldsymbol{Adv}_{\mathcal{D}}\big(E_H \cap \gamma y_1 + \alpha \neq 0 \,\big|\, b_1 = 0\big) + \boldsymbol{Adv}_{\mathcal{D}}\big(E_H \cap \gamma y_1 + \alpha = 0 \,\big|\, b_1 = 0\big) \\ &\leq \boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, \gamma y_1 + \alpha \neq 0 \cap b_1 = 0\big) + \Pr[\gamma y_1 + \alpha = 0] \cdot \boldsymbol{Adv}_{\mathcal{D}}\big(E_H \,\big|\, \gamma y_1 + \alpha = 0 \cap b_1 = 0\big) \\ &\leq \frac{1}{N} \cdot \boldsymbol{Adv}(k-1, q-1) + \frac{1}{N}(\boldsymbol{Adv}(k-1, q-1) + \Delta(k-1, q))\end{aligned}$$

implying (4).

3.4 Putting It All Together

We can use the above reductions and the final security bound to prove our two main theorems. At this point, both proofs follow in a relatively straightforward manner.

Theorem 1. *Let N be a prime and $S, T \geq 1$ be integers. For any generic algorithm with preprocessing $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{A}_0$ outputs an S-bit advice string, and $\mathcal{A}_1$ makes at most T oracle queries, it holds that*

$$\mathbb{P}\left[\mathcal{A}_1^{\mathcal{O}_\sigma}\Big(\mathcal{A}_0^{\mathcal{O}_\sigma}\big(\sigma(1)\big), \sigma(1), \sigma(x), \sigma(y), \sigma(u)\Big) = b\right] \leq \frac{1}{2} + 2^{11}\frac{ST^2}{N},$$

where $x, y, z \leftarrow \mathbb{Z}_N$, $b \leftarrow \{0,1\}$, $u = bz + (1-b)xy$, and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ is a random injective function.

Proof. Let $\mathcal{A}'$ be a generic algorithm with preprocessing such that $\mathcal{A}'$ produces a $(2S)$-bit advice string and makes at most T online queries. The advantage of $\mathcal{A}'$ is at least that of $\mathcal{A}$.

Now, let $k = 2S, Q = 2ST, q = 2(4ST + 6S + 1)(6ST + 6S + 1)$, apply Lemma 1 and Lemma 2 to construct a $(2S)$-XOR DDH algorithm $\mathcal{B}$ and a hyperplane query algorithm $\mathcal{C}$, then apply Lemma 3, such that

$$\boldsymbol{Adv}^{2S}_{\mathcal{A}'} \leq 2^{2S}\boldsymbol{Adv}_{\mathcal{B}} = 2^{2S}\boldsymbol{Adv}_{\mathcal{C}} \leq \left(\frac{24(4ST + 6S + 1)(6ST + 6S + 1)}{SN} + \frac{24}{N}\right)^{2S},$$

and thus

$$\boldsymbol{Adv}_{\mathcal{A}} \leq \boldsymbol{Adv}_{\mathcal{A}'} \leq \frac{3432(ST)^2}{SN} + \frac{24}{N} \leq 2^{12}\left(\frac{ST^2}{N}\right).$$

□

Theorem 2. *Let N be a prime and $k, T \geq 1$ be integers. For any generic algorithm $\mathcal{B}$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{B}$ makes at most kT oracle queries, it holds that*

$$\mathbb{P}\left[\mathcal{B}^{\mathcal{O}_\sigma}\big(\sigma(1), \sigma(\mathbf{w})\big) = \bigoplus_{i=1}^k b_i\right] \leq \frac{1}{2} + \frac{1}{2}\Big(2^{11}\frac{kT^2}{N}\Big)^k,$$

where $\sigma(\mathbf{w}) := \big\{\sigma(x_i), \sigma(y_i), \sigma(u_i)\big\}_{i=1}^k$ for $u_i = b_i z_i + (1-b_i)x_i y_i$, $\big\{x_i, y_i, z_i\big\}_{i=1}^k \leftarrow \mathbb{Z}_N^{3k}$, $\{b_1, \ldots, b_k\} \leftarrow \{0,1\}^k$, and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ a uniformly random injective function.

Proof. Let $\mathcal{B}$ be a k-XOR DDH algorithm such that $\mathcal{B}$ makes at most kT queries. Let $q = 2(2kT + 3k + 1)(3kT + 3k + 1)$, apply Lemma 2 to construct a hyperplane query algorithm $\mathcal{C}$ and apply Lemma 3, such that

$$\boldsymbol{Adv}_{\mathcal{B}} = \boldsymbol{Adv}_{\mathcal{C}} \leq \left(\frac{1008k^2T^2}{kN} + \frac{12}{N}\right)^k \leq \left(2^{11}\frac{kT^2}{N}\right)^k.$$

□

References

[ACDW20] Akshima, David Cash, Andrew Drucker, and Hoeteck Wee. Time-space tradeoffs and short collisions in merkle-damgård hash functions. In *CRYPTO*, 2020.

[ADGL23] Akshima, Xiaoqi Duan, Siyao Guo, and Qipeng Liu. On time-space lower bounds for finding short collisions in sponge hash functions. In *TCC*, 2023.

[AGL22] Akshima, Siyao Guo, and Qipeng Liu. Time-space lower bounds for finding collisions in merkle-damgård hash functions. In *CRYPTO*, 2022.

[BL13] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: the power of free precomputation. In *ASIACRYPT*, 2013.

[BMZ19] James Bartusek, Fermi Ma, and Mark Zhandry. The distinction between fixed and random generators in group-based assumptions. In *CRYPTO*, 2019.

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In *CRYPTO*, 2018.

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In *EUROCRYPT*, 2018.

[CGLQ20] Kai-Min Chung, Siyao Guo, Qipeng Liu, and Luowen Qian. Tight quantum time-space tradeoffs for function inversion. In *FOCS*, 2020.

[CHM20] Dror Chawin, Iftach Haitner, and Noam Mazor. Lower bounds on the time/memory tradeoff of function inversion. In *TCC*, 2020.

[CK18] Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In *EUROCRYPT*, 2018.

[CK19] Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In *TCC*, 2019.

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *EUROCRYPT*, 2017.

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

[FGK22] Cody Freitag, Ashrujit Ghoshal, and Ilan Komargodski. Time-space tradeoffs for sponge hashing: Attacks and limitations for short collisions. In *CRYPTO*, 2022.

[FGK23] Cody Freitag, Ashrujit Ghoshal, and Ilan Komargodski. Optimal security for keyed hash functions: Avoiding time-space tradeoffs for finding collisions. In *EUROCRYPT*, 2023.

[FJM14] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logarithm, even-mansour and PRINCE. In *ASIACRYPT*, 2014.

[FN91] Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *STOC*, 1991.

[GGKL21] Nick Gravin, Siyao Guo, Tsz Chiu Kwok, and Pinyan Lu. Concentration bounds for almost k-wise independence with applications to non-uniform security. In *SODA*, 2021.

[GGPS23] Alexander Golovnev, Siyao Guo, Spencer Peters, and Noah Stephens-Davidowitz. Revisiting time-space tradeoffs for function inversion. In *CRYPTO*, 2023.

[GK22] Ashrujit Ghoshal and Ilan Komargodski. On time-space tradeoffs for bounded-length collisions in merkle-damgård hashing. In *CRYPTO*, 2022.

[GLLZ21] Siyao Guo, Qian Li, Qipeng Liu, and Jiapeng Zhang. Unifying presampling via concentration bounds. In *TCC*, 2021.

[Hel80] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[Imp11] Russell Impagliazzo. Relativized separations of worst-case and average-case complexities for NP. In *CCC*, 2011.

[KS01] Fabian Kuhn and René Struik. Random walks revisited: Extensions of pollard's rho algorithm for computing multiple discrete logarithms. In *SAC*, 2001.

[LCH11] Hyung Tae Lee, Jung Hee Cheon, and Jin Hong. Accelerating id-based encryption based on trapdoor dl using pre-computation. *Cryptology ePrint Archive*, 2011.

[Liu23] Qipeng Liu. Non-uniformity and quantum advice in the quantum random oracle model. In *EUROCRYPT*, 2023.

[Mih10] Joseph P Mihalcik. *An analysis of algorithms for solving discrete logarithms in fixed groups*. PhD thesis, Citeseer, 2010.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In *CRYPTO*, 2007.

[Yun15] Aaram Yun. Generic hardness of the multiple discrete logarithm problem. In *EUROCRYPT*, 2015.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, 1979.

A Preprocessing algorithms for DDH

Here, we show that any preprocessing algorithm for the discrete logarithm problem yields a preprocessing algorithm for the DDH problem with roughly the same advantage and complexity.

Proposition 1. *Let N be a prime and $S, T \geq 1$ be integers. For any generic algorithm with preprocessing $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{A}_0$ outputs an S-bit advice string, $\mathcal{A}_1$ makes at most T oracle queries, and*

$$\mathbb{P}\left[\mathcal{A}_1^{\mathcal{O}_\sigma}\left(\mathcal{A}_0^{\mathcal{O}_\sigma}\left(\sigma(1)\right), \sigma(1), \sigma(x)\right) = x\right] = \varepsilon,$$

where $x \leftarrow \mathbb{Z}_N$ and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ is a random injective function, there exists a generic algorithm with preprocessing $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ over $\mathcal{L}$ for $\mathbb{Z}_N$ such that $\mathcal{B}_0$ outputs an S-bit advice string, $\mathcal{B}_1$ makes at most $T + 4\lceil \log N \rceil$ oracle queries, and

$$\mathbb{P}\left[\mathcal{B}_1^{\mathcal{O}_\sigma}\left(\mathcal{B}_0^{\mathcal{O}_\sigma}(\sigma(1)), \sigma(1), \sigma(x), \sigma(y), \sigma(u)\right) = b\right] = \frac{1}{2} + \frac{\varepsilon}{2}(1 - \frac{1}{N}),$$

where $x, y, z \leftarrow \mathbb{Z}_N$, $b \leftarrow \{0,1\}$, $u = bz + (1-b)xy$, and $\sigma : \mathbb{Z}_N \to \mathcal{L}$ is a random injective function.

Proof. Let $\mathcal{B}_0$ be the same as $\mathcal{A}_0$ which outputs an S-bit advice, and $\mathcal{B}_1$ be the following algorithm:

1. Simulate $\mathcal{A}_1$ with the output from $\mathcal{B}_0$ on $(\sigma(1), \sigma(x))$ and obtain $\hat{x}$.
2. Compute $\sigma(\hat{x})$ given $\sigma(1)$ and $\hat{x}$. If $\sigma(x) \neq \sigma(\hat{x})$, return a random bit $\hat{b} \leftarrow \{0,1\}$.
3. Compute $\sigma(\hat{x}y)$ given $\sigma(y)$ and $\hat{x}$. If $\sigma(u) = \sigma(\hat{x}y)$, return $\hat{b} := 0$ otherwise $\hat{b} := 1$.

Note that for two integers $x, y \in \mathbb{Z}_N$, we can compute the value of $\sigma(xy)$ given x and $\sigma(y)$ with at most $2\lceil \log N \rceil$ queries to $\mathcal{O}_\sigma$[1]. Therefore $\mathcal{B}_1$ makes at most $T + 4\lceil \log N \rceil$ oracle queries.

Observe that, conditioning on $\sigma(x) \neq \sigma(\hat{x})$, $\hat{b} = b$ happens with probability exactly $1/2$. Conditioning on $\sigma(x) = \sigma(\hat{x})$, by the injectivity of σ, and the independence of y, z, b from $\hat{x}$ and σ,

$$\mathbb{P}\big[\hat{b} = b \,\big|\, \hat{x} = x\big] = \mathbb{P}\big[b = 0\big] + \mathbb{P}\big[z \neq xy \cap b = 1\big] = \frac{1}{2} + \frac{1}{2}\Big(1 - \frac{1}{N}\Big) = 1 - \frac{1}{2N}.$$

Therefore, the overall success probability of $\mathcal{B}$ is

$$\mathbb{P}\big[\hat{b} = b \,\big|\, \hat{x} \neq x\big]\mathbb{P}\big[\hat{x} \neq x\big] + \mathbb{P}\big[\hat{b} = b \,\big|\, \hat{x} = x\big] = \frac{1}{2}(1 - \varepsilon) + \Big(1 - \frac{1}{2N}\Big)\varepsilon = \frac{1}{2} + \frac{\varepsilon}{2}\Big(1 - \frac{1}{N}\Big).$$

□

Combine this proposition with known generic preprocessing algorithms for discrete-log problem [BL13, LCH11, Mih10], we obtain the following corollary:

Corollary. *Let N be a prime and $S, T \geq 1$ be integers, there exists a generic preprocessing algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ over $\mathcal{L}$ for $\mathbb{Z}_N$ for the DDH problem such that $\mathcal{A}_0$ outputs an S-bit advice string, $\mathcal{A}_1$ makes at most T oracle queries with advantage $\widetilde{\Omega}(ST^2/N)$.*

[1]Given $\sigma(y)$ and an integer $x \in \mathbb{Z}_N$, we can compute $\sigma(xy)$ using recursion: if $x = 1$, return $\sigma(y)$; otherwise first compute $Z := \sigma(\lfloor x/2 \rfloor y)$ using recursion, then return $\mathcal{O}_\sigma(Z, Z)$ if x is even, and return $\mathcal{O}_\sigma(\sigma(y), \mathcal{O}_\sigma(Z, Z))$ if x is odd. Because each recursion reduces computing $\sigma(xy)$ to a single sub-problem of computing $\sigma(\lfloor x/2 \rfloor y)$, and makes at most two oracle queries to O_σ. The number of recursion calls is at most $\lceil \log N \rceil$ and the number of oracle queries is at most $2\lceil \log N \rceil$.