On the Memory-Tightness of Hashed ElGamal*

Ashrujit Ghoshal and Stefano Tessaro

Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle, USA
{ashrujit,tessaro}@cs.washington.edu

Abstract. We study the memory-tightness of security reductions in public-key cryptography, focusing in particular on Hashed ElGamal. We prove that any *straightline* (i.e., without rewinding) black-box reduction needs memory which grows linearly with the number of queries of the adversary it has access to, as long as this reduction treats the underlying group generically. This makes progress towards proving a conjecture by Auerbach *et al.* (CRYPTO 2017), and is also the first lower bound on memory-tightness for a concrete cryptographic scheme (as opposed to generalized reductions across security notions). Our proof relies on compression arguments in the generic group model.

Keywords: Public-key cryptography, memory-tightness, lower bounds, generic group model, foundations, compression arguments

1 Introduction

Security proofs rely on *reductions*, i.e., they show how to transform an adversary $\mathcal{A}$ breaking a scheme into an adversary $\mathcal{B}$ solving some underlying assumed-to-be-hard problem. Generally, the reduction ought to be *tight* – the resources used by $\mathcal{B}$, as well as the attained advantage, should be as close as possible to those of $\mathcal{A}$. Indeed, the more resources $\mathcal{B}$ needs, or the smaller its advantage, the weaker the reduction becomes.

Auerbach *et al.* [2] were the first to explicitly point out that *memory* resources have been ignored in reductions, and that this leads to a loss of quality in security results. Indeed, it is conceivable that $\mathcal{A}$'s memory is naturally bounded (say, at most 2^{64} bits), and the underlying problem is very sensitive to memory. For example, the best-known algorithm for discrete logarithms in a 4096-bit prime field runs in time (roughly) 2^{156} using memory 2^{80}. With less memory, the best algorithm is the generic one, requiring time $\Theta(\sqrt{p}) \approx 2^{2048}$. Therefore, if $\mathcal{B}$ also uses memory at most 2^{64}, we can infer a larger lower bound on the necessary time complexity for $\mathcal{A}$ to break the scheme, compared to the case where $\mathcal{B}$ uses 2^{100} bits instead.

What can be memory-tight? One should therefore target reductions that are *memory-tight*, i.e., the memory usage of $\mathcal{B}$ is similar to that of $\mathcal{A}$.[1] The work of Auerbach *et al.* [2], and its follow-up by Wang *et al.* [13], pioneered the study of memory-tight reductions. In particular, and most relevant to this work, they show *negative* results (i.e., that certain reductions cannot be memory tight) using *streaming lower bounds*.

Still, these lower bounds are tailored at general notions (e.g., single- to multi-challenge reductions), and lower bounds follow from a natural connection with classical frequency problems on streams. This paper tackles the more ambitious question of proving impossibility of memory-tight reductions for concrete *schemes*, especially those based on algebraic structures. This was left as an open problem by prior works.

Hashed ElGamal. Motivated by a concrete open question posed in [2], we consider here the CCA-security of Hashed ElGamal. In its KEM variant, the scheme is based on a cyclic group $G = \langle g \rangle$ – the secret key sk is a random element from $\mathbb{Z}_{|G|}$, whereas the public key is $\mathsf{pk} = g^{\mathsf{sk}}$. Then, encapsulation produces a ciphertext-key pair

$$C \leftarrow g^r \ , \quad K \leftarrow \mathsf{H}(\mathsf{pk}^r) \ .$$

for $r \leftarrow \mathbb{Z}_{|G|}$ and a hash function $\mathsf{H} : G \to \{0,1\}^\ell$. Decapsulation occurs by computing $K \leftarrow \mathsf{H}(C^{\mathsf{sk}})$.

* A preliminary version of this paper appears in the proceedings of EUROCRYPT 2020. This is the full version.

[1] Generally, $\mathcal{B} = \mathcal{R}^{\mathcal{A}}$ for a black-box reduction $\mathcal{R}$, and one imposes the slightly stronger requirement that $\mathcal{R}$ uses small memory, independent of that of $\mathcal{A}$.

The CCA-security of Hashed ElGamal in the random-oracle model was proved by Abdalla, Bellare, and Rogaway [1] based on the *Strong Diffie-Hellman* (SDH) assumption (also often called GapDH), and we briefly review the proof.[2] First, recall that in the SDH assumption, the attacker is asked to compute g^{uv} from g^u and g^v, given additionally access to a *decision* oracle O_v which on input $h, y \in G$, tells us whether $h^v = y$.

The reduction sets the Hashed ElGamal public-key to $\mathsf{pk} = g^v$ (setting implicitly $\mathsf{sk} = v$), the challenge ciphertext to be $C^* = g^u$, and the corresponding key K^* to be a random string. Then, it simulates both the random oracle and the decapsulation oracle to the adversary $\mathcal{A}$ (which is run on inputs pk, C^* and K^*), until a random-oracle query for g^{uv} is made (this can be detected using the O_v oracle). The challenge is to simulate both oracles consistently: As the reduction cannot compute discrete logarithms, it uses the oracle O_v to detect whether a random-oracle query X and a decapsulation query C_i satisfy $\mathsf{O}_\mathsf{v}(C_i, X) = \mathsf{true}$, and, if this is the case, answers them with the same value.

This reduction requires memory to store all prior decapsulation and random-oracle queries. Unlike other reductions, the problem here is not to store the random-oracle output values (which could be compressed using a PRF), but the actual *inputs* to these queries, which are under adversarial control. This motivates the conjecture that a reduction using little memory does not exist, but the main challenge is of course to prove this is indeed the case.

Our result, in summary. We provide a *memory* lower bound for reductions that are *generic* with respect to the underlying group G. Specifically, we show the existence of an (inefficient) adversary $\mathcal{A}$ in *the generic group model (GGM)* which breaks the CCA security of Hashed ElGamal via $O(k)$ random oracle/decapsulation queries, but such that no reduction using less than $k \cdot \lambda$ bits of memory can break the SDH assumption *even* with access to $\mathcal{A}$, where λ is the bit-size of the underlying group elements.

Our lower bound is strong in that it shows we do not even have a trade-off between advantage and memory, i.e., if the memory is smaller than $k \cdot \lambda$, then the advantage is very small, as long as the reduction makes a polynomial number of queries to O_v and to the generic group oracle. It is however also important to discuss two limitations of our lower bound. The first one is that the reduction – which receives g, g^v in the SDH game – uses $\mathsf{pk} = g^v$ as the public key to the Hashed ElGamal adversary. The second one is that the reduction is straightline, i.e., it does not perform any rewinding.

We believe that our impossibility result would extend even when the reduction is not straightline. However, allowing for rewinding appears to be out of reach of our techniques. Nonetheless, we *do* conjecture a lower bound on the memory of $\Omega(k \log k)$ bits, and discuss the reasoning behind our conjecture in detail in Appendix C.

We stress that our result applies to reductions in the GGM, but treats the adversary as a black box. This captures reductions which are black-box in their usage of the group and the adversary. (In particular, the reduction cannot see generic group queries made by the adversary, as in a GGM security proofs.) Looking at the GGM reduces the scope of our result. However, it is uncommon for reductions to depend on the specifics of the group, although our result can be bypassed for specific groups, e.g., if the group has a pairing.

Concurrent related work. Concurrently to our work, Bhattacharyya [4] provides memory-tight reductions of KEM-CCA security for variants of Hashed ElGamal. At first glance, the results seem to contradict ours. However, they are entirely complementary – for example, a first result shows a memory tight reduction for the KEM-CCA security of the "Cramer-Shoup" variant of Hashed ElGamal – this variant differs from the (classical) Hashed ElGamal we consider here and is less efficient. The second result shows a memory-tight reduction for the version considered in this paper, but assumes that the underlying group has a pairing. This is a good example showing our result can be bypassed for specific groups i.e. groups with pairings, but we also note that typical instantiations of the scheme are on elliptic curves for which no pairing exists.

[2] Abdalla et al. [1] do not phrase their paper in terms of the KEM/DEM paradigm [12,6], which was introduced concurrently – instead, they prove that an intermediate assumption, called Oracle Diffie-Hellman (ODH), follows from SDH in the ROM. However, the ODH assumption is structurally equivalent to the CCA security of Hashed ElGamal KEM for one challenge ciphertext.

1.1 Our Techniques

We give a high-level overview of our techniques here. We believe some of these to be novel and of broader interest in providing other impossibility results.

THE SHUFFLING GAME. Our adversary against Hashed ElGamal[3] $\mathcal{A}$ first attempts to detect whether the reduction is using a sufficient amount of memory. The adversary $\mathcal{A}$ is given as input the public key g^v, as well as g^u, as well as a string $C^* \in \{0,1\}^\ell$, which is either a real encapsulation or a random string. It first samples k values $i_1, \ldots, i_k$ from $\mathbb{Z}_p$. It then:

(1) Asks for decapsulation queries for $C_j \leftarrow g^{i_j}$, obtaining values K_j, for $j \in [k]$
(2) Picks a random permutation $\pi : [k] \to [k]$.
(3) Asks for RO queries for $H_j \leftarrow \mathsf{H}(V_j)$ for $j \in [k]$, where $V_j \leftarrow g^{v \cdot i_{\pi(j)}}$.

After this, the adversary checks whether $K_j = H_{\pi(j)}$ for all $j \in [k]$, and if so, it continues its execution, breaking the ODH assumption (inefficiently). If not, it just outputs a random guess.

The intuition here is that no reduction using substantially less than $k \cdot \log p$ bits succeeds in passing the above test – in particular, because the inputs C_j and V_j are (pseudo-)random, and thus incompressible. If the test does not pass, the adversary $\mathcal{A}$ is rendered useless, and thus not helpful to break SDH.

Remark 1. The adversary here is described in a way that requires secret randomness, not known to the reduction, and it is easier to think of $\mathcal{A}$ in this way. We will address in the body how to generically make the adversary deterministic.

Remark 2. We stress that this adversary requires memory – it needs to remember the answers $C_1, \ldots, C_k$. However, recall that we adopt a black-box approach to memory-tightness, where our requirement is that the reduction itself uses little memory, regardless of the memory used by the adversary. We also argue this is somewhat necessary – it is not clear how to design a reduction which adapts its memory usage to the adversary, even if given this information in a non-black-box manner. Also, we conjecture different (and much harder to analyze) memory-less adversaries exist enabling a separation. An example is multi-round variant, where each round omits **(2)**, and **(3)** only asks a single query $\mathsf{H}(V_j)$ for a random $j \leftarrow\!\!\$\, [k]$, and checks consistency. Intuitively, the chance of passing each round is roughly $k \log p/s$, but we do not know how to make this formal.

INTRODUCING THE GGM. Our intuition is however *false* for an arbitrary group. For instance, if the discrete logarithm (DL) problem is easy in the group, then the reduction can simply simulate the random oracle via a PRF, as suggested in [2]. Ideally, we could prove that if the DL problem is hard in G, then any PPT reduction given access to $\mathcal{A}$ and with less than $k \cdot \log p$ bits of memory fails to break SDH.[4] Unfortunately, it will be hard to capture a single hardness property of G sufficient for our proof to go through. Instead, we will model the group via the *generic group model* (GGM) [11,9]: We model a group of prime order p defined via a random injection $\sigma : \mathbb{Z}_p \to \mathcal{L}$. An algorithm in the model typically has access to $\sigma(1)$ (in lieu of g) and an evaluation oracle which on input $\mathbf{a}, \mathbf{b} \in \mathcal{L}$ returns $\sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$. (We will keep writing g^i instead of $\sigma(i)$ in the introduction, for better legibility.)

THE PERMUTATION GAME. In order to fool $\mathcal{A}$, the reduction can learn information about π via the O_v oracle. For example, it can try to input $C_j = g^{i_j}$ and $V_{j'} = g^{v i_{\pi(j')}}$ (both obtained from $\mathcal{A}$'s queries), and $\mathsf{O}_\mathsf{v}(C_j, V_{j'}) = \mathsf{true}$ if and only if $\pi(j') = j$. More generally, the reduction can compute, for any $\vec{a} = (a_1, \ldots, a_k)$ and $\vec{b} = (b_1, \ldots, b_k)$,

$$C^* = g^{\sum_{j=1}^k a_j i_j} = \prod_{j=1}^k C_j^{a_j} \;, \quad V^* = g^{\sum_{j=1}^k b_j v \cdot i_{\pi(j)}} = \prod_{j=1}^k V_j^{b_j} \;,$$

[3] The paper will in fact use the cleaner formalization of the ODH assumption, so we stick to Hashed ElGamal only in the introduction.

[4] This statement is somewhat confusing, so note that in general, the existence of a reduction is *not* a contradiction with the hardness of DL, as the reduction is meant to break SDH only given access to an adversary breaking the scheme, and this does not imply the ability to break SDH *without* access to the adversary.

and the query $\mathsf{O}_{\mathsf{v}}(C^*, V^*)$ returns true iff $b_j = a_{\pi(j)}$ for all $j \in [k]$, which we write as $\vec{b} = \pi(\vec{a})$. We abstract this specific strategy in terms of an information-theoretic game – which we refer to as the *permutation game* – which gives the adversary access to an oracle O which takes as inputs pairs of vectors $(\vec{a}, \vec{b})$ from $\mathbb{Z}_p^k$, and returns true iff $\vec{b} = \pi(\vec{a})$ for a secret permutation π. The goal of the adversary is to recover π.

Clearly, a strategy can win with $O(k^2)$ oracle queries $(\vec{e}_i, \vec{e}_j)$ for all i, j, where $\vec{e}_i \in \mathbb{Z}_p^k$ is the unit vector with a 1 in the i-th coordinate, and 0 elsewhere. This strategy requires in particular querying, in its first component, vectors which have rank k. Our first result will prove that this is necessary – namely, assume that an adversary makes a sequence of q queries $(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_q, \vec{y}_q)$ such that the rank of $\vec{x}_1, \ldots, \vec{x}_q$ is at most ℓ, then the probability to win the permutation game is of the order $O(q^\ell/k!)$. We will prove this via a compression argument.

Note that roughly, this bound tells us that to win with probability ϵ and q queries to the oracle, the attacker needs

$$\ell = \Omega\left(\frac{k \log k - \log(1/\epsilon)}{\log(q)}\right) .$$

A REDUCTION TO THE PERMUTATION GAME. We will think of the execution of the reduction against our adversary as consisting of two stages – we refer to them as $\mathcal{R}_1$ and $\mathcal{R}_2$. The former learns the decapsulation queries $g^{i_1}, \ldots, g^{i_k}$, whereas the latter learns the RO queries $g^{i_{\pi(1)} v}, \ldots, g^{i_{\pi(k)} v}$, and (without loss of generality) attempts to guess the permutation π. We will lower bound the size of the state ϕ that $\mathcal{R}_1$ passes on to $\mathcal{R}_2$. Both stages can issue O_{v} and Eval queries.

Note that non-trivial O_{v} queries (i.e., those revealing some information about the permutation), are (except with very small probability) issued by $\mathcal{R}_2$, since no information about π is ever revealed to $\mathcal{R}_1$. As one of our two key steps, we will provide a reduction from the execution of $\mathcal{R}_1, \mathcal{R}_2$ against $\mathcal{A}$ in the GGM to the permutation game – i.e., we build an adversary $\mathcal{D}$ for the latter game simulating the interaction between $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{A}$, and such that $\mathcal{R}_1, \mathcal{R}_2$ "fooling" $\mathcal{A}$ results in $\mathcal{D}$ guessing the permutation.

MEMORY VS. RANK. The main question, however, is to understand the complexity of $\mathcal{D}$ in the permutation game, and in particular, the *rank* ℓ of the first component of its queries – as we have seen above, this affects its chance of winning the game.

To do this, we will take a slight detour, and specifically consider a set $\mathcal{Z} \subseteq \mathcal{L}$ of labels (i.e., outputs of σ) that the reduction $\mathcal{R}_2$ comes up with (as inputs to either of Eval or O_{v}) on its own (in the original execution), i.e., no earlier Eval query of $\mathcal{R}_2$ returned them, and that have been previously learnt by $\mathcal{R}_1$ as an output of its Eval queries. (The actual definition of $\mathcal{Z}$ is more subtle, and this is due to the ability of the adversary to come up with labels *without* knowing the corresponding pre-image.)

Then, we will show two statements about $\mathcal{Z}$:

- **(i)** On the one hand, we show that the rank ℓ of the oracle queries of the adversary $\mathcal{D}$ is upper bound by $|\mathcal{Z}|$ in its own simulation of the execution of $\mathcal{R}_1, \mathcal{R}_2$ with $\mathcal{A}$.
- **(ii)** On the other hand, via a compression argument, we prove that the size of $\mathcal{Z}$ is related to the length of ϕ, and this will give us our final upper bound.

This latter statement is by itself not very surprising – one can look at the execution of $\mathcal{R}_2$, and clearly every label in $\mathcal{Z}$ that appears "magically" in the execution must be the result of storing them into the state ϕ. What makes this different from more standard compression arguments is the handling of the generic group model oracle (which admits non-trivial operations). In particular, our compression argument will compress the underlying map σ, and we will need to be able to figure out the pre-images of these labels in $\mathcal{Z}$. We give a very detailed technical overview in the body explaining the main ideas.

MEMORY-TIGHT AGM REDUCTION. The Algebraic Group Model (AGM) was introduced in [8]. AGM reductions make strong extractability assumptions, and the question of their memory-tightness is an interesting one. In Appendix A we construct a reduction to the discrete logarithm problem that runs an adversary against the KEM-CCA security of Hashed ElGamal in the AGM such that the reduction is memory-tight but not tight with respect to advantage. We note that the model of our reduction is different than a (full-fledged) GGM reduction which is not black-box, in that it can observe the GGM queries made by the adversary. Our

result does not imply any impossibility for these. In turn, AGM reductions are weaker, but our results do not imply anything about them, either.

2 Preliminaries

In this section, we review the formal definition of the generic group model. We also state ODH and SDH as introduced in [1] in the generic group model.

NOTATION. Let $\mathbb{N} = \{0, 1, 2, \cdots\}$ and, for $k \in \mathbb{N}$, let $[k] = \{1, 2, \cdots, k\}$. We denote by $\mathsf{InjFunc}(S_1, S_2)$ the set of all injective function from S_1 to S_2.

We also let $*$ denote a wildcard element. For example $\exists t : (t, *) \in T$ is true if the set T contains an ordered pair whose first element is t (the type of the wildcard element shall be clear from the context). Let $\mathcal{S}_k$ denote the set of all permutations on $[k]$. We use $f : \mathsf{D} \to \mathsf{R} \cup \{\perp\}$ to denote a partial function, where $f(x) = \perp$ indicates the value of $f(x)$ is undefined. Define in particular $D(f) = \{d \in \mathsf{D} : f(d) \neq \perp\}$ and $R(f) = \{r \in \mathsf{R} : \exists d \in \mathsf{D} : \sigma(d) = r\}$. Moreover, we let $\overline{D(f)} = \mathsf{D} \backslash D(f)$ and $\overline{R(f)} = \mathsf{R} \backslash R(f)$.

We shall use pseudocode descriptions of games inspired by the code-based framework of [3]. We make use of pairs of games being identical-until-bad (i.e., the games are identical until a bad flag is set), and apply the so-called "Fundamental Lemma" to upper bound the difference in the probability of the game outputs in terms of the probability of bad being set to true in either of the games. The output of a game is denoted using the symbol $\Rightarrow$. In all games we assume the flag bad is set to false initially. In pseudocode, we denote random sampling using $\leftarrow\!\!\$$, assignment using $\leftarrow$ and equality check using $=$. In games that output boolean values, we use the term"winning" the game to mean that the output of the game is true.

We also introduce some linear-algebra notation. Let S be a set vectors with equal number of coordinates. We denote the rank and the linear span of the vectors by $\mathsf{rank}(S)$ and $\mathsf{span}(S)$ respectively. Let $\vec{x}, \vec{y}$ be vectors of dimension k. We denote $\vec{z}$ of dimension $2k$ which is the concatenation of $\vec{x}, \vec{y}$ as $\vec{z} = (\vec{x}, \vec{y})$. We denote the element at index i of a vector $\vec{x}$ as $\vec{x}[i]$.

2.1 Generic Group Model

The *generic group model* [11] captures algorithms that do not use any special property of the encoding of the group elements, other than assuming every element of the group has a unique representation, and that the basic group operations are allowed. This model is useful in proving lower bounds for some problems, but we use it here to capture reductions that are not specific to the underlying group.

More formally, let the order of the group be a large prime p. Let $\mathbb{Z}_p = \{0, 1, 2, \cdots, p-1\}$. Let $\mathcal{L} \subset \{0,1\}^*$ be a set of size p, called the set of *labels*. Let σ be a random injective mapping from $\mathbb{Z}_p$ to $\mathcal{L}$. The idea is that now every group element in $\mathbb{Z}_p$ is represented by a label in $\mathcal{L}$. An algorithm in this model takes as input $\sigma(1), \sigma(x_1), \sigma(x_2), \cdots, \sigma(x_n)$ for some $x_1, \cdots, x_n \in \mathbb{Z}_p$ (and possibly other inputs which are not group elements). The algorithm also has access to an oracle named Eval which takes as input two labels $\mathbf{a}, \mathbf{b} \in \mathcal{L}$ and returns $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$. Note that for any d, given $\sigma(x_i)$, $\sigma(d \cdot x_i)$ can be computed using $O(\log d)$ queries to Eval. We denote this operation as $\mathsf{Exp}(\sigma(x_i), d)$. We assume that all labels queried by algorithms in the generic group model are valid i.e. all labels queried by algorithms in the generic group model are in $\mathcal{L}$.[5]

ORACLE DIFFIE-HELLMAN ASSUMPTION (ODH). We first formalize the Oracle Diffie-Hellman Assumption (ODH) [1], which we are going to use in lieu of the CCA security of Hashed ElGamal. Suppose, a group has generator g and order p. The domain of hash function H is all finite strings and range is $\{0,1\}^{\mathsf{hLen}}$. The assumption roughly states for $u, v \leftarrow\!\!\$\, \mathbb{Z}_p, W \leftarrow\!\!\$\, \{0,1\}^{\mathsf{hLen}}$, the distributions $(g^u, g^v, \mathsf{H}(g^{uv}))$ and (g^u, g^v, W) are indistinguishable to an adversary who has access to the oracle H_{v} where $\mathsf{H}_{\mathsf{v}}(g^x)$ returns $\mathsf{H}(g^{xv})$ with the restriction that it is not queried on g^u.

[5] We stress that we assume a strong version of the model where the adversary *knows* $\mathcal{L}$.

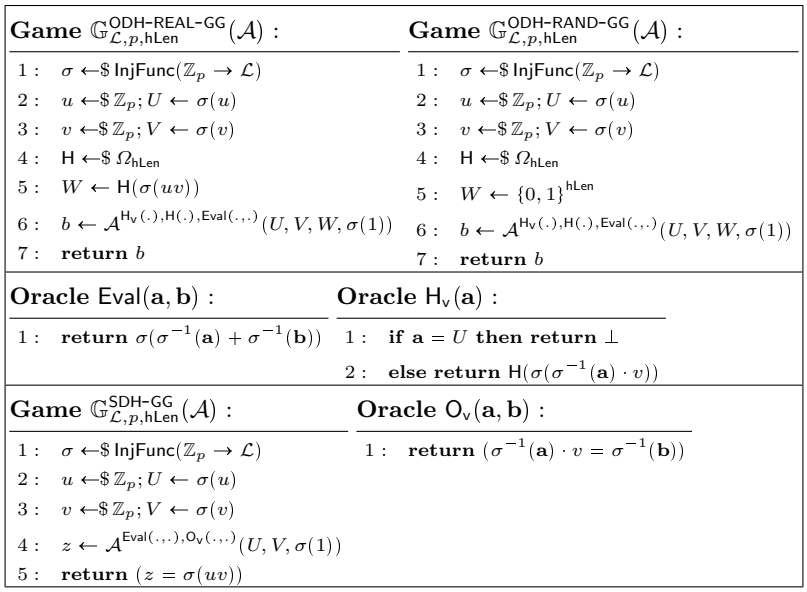

Fig. 1. Games for ODH and SDH assumptions

We give a formalization of this assumption in the random-oracle and generic group models. For a fixed $\mathsf{hLen} \in \mathbb{N}$, let Ω_{hLen} be the set of hash functions mapping $\{0,1\}^*$ to $\{0,1\}^{\mathsf{hLen}}$. In Figure 1, we formally define the Games $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}$, $\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}$. The advantage of violating ODH is defined as

$$\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) = \left|\Pr\left[\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right]\right| .$$

Strong Diffie-Hellman Assumption (SDH). This is a stronger version of the classical CDH assumption. This assumption roughly states that CDH is hard even in the presence of a DDH-oracle $\mathsf{O_v}$ where $\mathsf{O_v}(g^x, g^y)$ is true if and only if $x \cdot v = y$.

We formally define the game $\mathbb{G}^{\mathsf{SDH\text{-}GG}}$ in the generic group model in Figure 1. The advantage of violating SDH is defined as

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) = \left|\Pr\left[\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow \mathsf{true}\right]\right| .$$

Note in particular that one *can* upper bound this advantage unconditionally.
We shall drop the $\mathcal{L}$ from the subscript of advantages and games henceforth since the set of labels $\mathcal{L}$ remains the same throughout our paper.

Black Box reductions in the GGM. We consider black-box reductions in the generic group model. We will limit ourselves to an informal description, but this can easily be formalized within existing formal frameworks for reductions (see e.g. [10]). We let the reduction $\mathcal{R}$ access an adversary $\mathcal{A}$, and denote by $\mathcal{R}^{\mathcal{A}}$ the resulting algorithm – understood here is that $\mathcal{R}$ supplies inputs, answers queries, etc. In addition, we let $\mathcal{R}$ and $\mathcal{A}$ access the Eval oracle available in the GGM. We stress that the GGM oracle is not under the reduction's control here – typically, the reduction itself will break a (hard) problem in the GGM with help of $\mathcal{A}$. We will allow (for simplicity) $\mathcal{A}$ to be run depending on some secret private coins[6] not accessible by $\mathcal{R}$. Reductions can run $\mathcal{A}$ several times (with fresh private coins). We call a reduction *straigthline* if it only runs $\mathcal{A}$ *once*.

In our case, the reduction $\mathcal{R}$ will be playing $\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$. It receives as inputs $\sigma(1)$, $U = \sigma(u)$, $V = \sigma(v)$, and has access to the Eval, $\mathsf{O_v}$ oracles, as well as an adversary $\mathcal{A}$ for $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}$ or $\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}$. The reduction needs therefore to supply inputs $(\sigma(1), U', V', W)$ to $\mathcal{A}$, and to answer its queries to the oracles $\mathsf{H_v}$, as well as queries to H. We will call such a reduction *restricted* if it is straightline *and* $V' = V$.

[6] If we want to allow the reduction to control random bits, we model them explicitly as an additional input.

2.2 Compression Lemma

In our lower bound proof we use the compression lemma that was formalized in [7] which roughly means that it is impossible to compress every element in a set with cardinality c to a string less than $\log c$ bits long, even relative to a random string. We state the compression lemma here as a proposition.

Proposition 1. *Suppose, there is a (not necessarily efficient) procedure* $\mathsf{Encode} : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ *and a (not necessarily efficient) decoding procedure* $\mathsf{Decode} : \mathcal{Y} \times \mathcal{R} \to \mathcal{X}$ *such that*

$$\Pr_{x \in \mathcal{X}, r \in \mathcal{R}} [\mathsf{Decode}(\mathsf{Encode}(x, r), r) = x] \geqslant \epsilon \, ,$$

then $\log |\mathcal{Y}| \geqslant \log |\mathcal{X}| - \log(1/\epsilon)$.

2.3 Polynomials and Schwartz-Zippel Lemma

Let $\mathsf{p}(X_1, \cdots, X_n)$ be a n variate polynomial. We denote by $\mathsf{p}(x_1, \cdots, x_n)$ the evaluation of p at the point $(x_1, \cdots, x_n)$ throughout the paper. The polynomial ring in variables $X_1, \cdots, X_n$ over the field $\mathbb{Z}_p$ is denoted by $\mathbb{Z}_p[X_1, \cdots, X_n]$. We state the Schwartz-Zippel Lemma in $\mathbb{Z}_p$ which will be useful later.

Proposition 2. *Let* p *be a non-zero* n *variate polynomial with degree* d. *Then*

$$\Pr_{x_1, \cdots, x_n \leftarrow\$ \mathbb{Z}_p} [Q(x_1, \cdots, x_n) = 0 \pmod p] \leqslant \frac{d}{p} \, .$$

2.4 Key Encapsulation Mechanism (KEM)

A key-encapsulation mechanism (KEM) consists of three probabilistic polynomial time (PPT) algorithms $\mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap}$. The key generation algorithm Gen is probabilistic and outputs a key-pair $(\mathsf{pk}, \mathsf{sk})$. The encapsulation algorithm Encap is a probabilistic algorithm that takes pk as input and outputs a ciphertext c and a key K where $K \in \mathcal{K}$ for some non-empty set $\mathcal{K}$. The decapsulation algorithm Decap is a deterministic algorithm that takes as input the secret key sk and a ciphertext c outputs a key $K \in \mathcal{K}$ if (sk, c) is a valid secret key-ciphertext pair and $\perp$ otherwise. For correctness, it is required that for all pairs $(\mathsf{pk}, \mathsf{sk})$ output by Gen, if (K, c) is output by $\mathsf{Encap}(\mathsf{pk})$ then K is the output of $\mathsf{Decap}(\mathsf{sk}, c)$.

Single challenge KEM-CCA security. The single challenge CCA security of a KEM is defined by a pair of games called $\mathbb{G}^{\mathsf{KEM\text{-}CCA\text{-}REAL}}, \mathbb{G}^{\mathsf{KEM\text{-}CCA\text{-}RAND}}$. In both games a $(\mathsf{pk}, \mathsf{sk})$ pair is generated by Gen, and (c, K) is output by the encapsulation algorithm Encap on input pk. The adversary is provided with (pk, c, K) in $\mathbb{G}^{\mathsf{KEM\text{-}CCA\text{-}REAL}}$ and with (pk, c, K') in $\mathbb{G}^{\mathsf{KEM\text{-}CCA\text{-}RAND}}$ where K' is a randomly sampled element of $\mathcal{K}$. The adversary has access to the decapsulation oracle with sk as the secret key and it can make decapsulation queries on any ciphertext except the ciphertext c and has to output a bit. We define the advantage of violating single challenge KEM-CCA security is defined as the absolute value of the difference of probabilities of the adversary outputting 1 in the two games. A KEM is single challenge CCA-secure if for all non-uniform PPT adversaries the advantage of violating single challenge KEM-CCA security is negligible.

Single challenge KEM-CCA of Hashed ElGamal. We describe the KEM for Hashed ElGamal in a group with order p and generator g and a hash function H. The function Gen samples v at random from $\mathbb{Z}_p$, and returns (g^v, v) as the $(\mathsf{pk}, \mathsf{sk})$ pair. The function Encap on input v, samples u at random from $\mathbb{Z}_p$ and returns g^u as the ciphertext and $\mathsf{H}(g^{uv})$ as the key K. The function Decap on input c, returns $\mathsf{H}(c^v)$. Note that Decap in KEM of Hashed ElGamal is identical to the H_v function as defined in the ODH assumption. It follows that the single challenge KEM-CCA security of Hashed ElGamal is equivalent to the ODH assumption. In particular, in the generic group model when H is modeled as a random oracle, the single challenge KEM-CCA security of Hashed ElGamal is equivalent to the ODH assumption in the random oracle and generic group model.

3 Memory Lower Bound on the ODH-SDH Reduction

3.1 Result and Proof Outline

In this section, we prove a memory lower bound for restricted black-box reductions from ODH to SDH. We stress that the restricted reduction has access only to the $\mathsf{H}, \mathsf{H}_\mathsf{v}$ queries of the adversary. As discussed above, the ODH assumption is equivalent to the single-challenge KEM-CCA security of Hashed ElGamal, this proves a memory lower-bound for (restricted) black-box reductions of single challenge KEM-CCA security of Hashed ElGamal to the SDH assumption.

Theorem 1 (Main Theorem). *In the generic group model, with group order p, there exists an ODH adversary $\mathcal{A}$ that makes k H queries and k H_v queries (where k is a polynomial in $\log p$), a function $\epsilon_1(p, \mathsf{hLen})$ which is negligible in $\log p, \mathsf{hLen}$, and a function $\epsilon_2(p)$ which is negligible in $\log p$, such that,*

1. $\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) = 1 - \epsilon_1(p, \mathsf{hLen})$.
2. *For all restricted black-box reductions $\mathcal{R}$, with s bits of memory and making a total of q (assuming $q \geqslant k$ and $6q \leqslant p - 4k - 4$) queries to O_v, Eval,*

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{48q^3}{p}\right)^{\frac{k}{8c}} \left(1 + \frac{6q}{p}\right)^q + \frac{4q^2 \log p + 13q^2 + 5q}{p} + \epsilon_2(p) \ ,$$

where $c = 4\lceil \frac{\log q}{\log k} \rceil$.

This result implies that if $\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}})$ is non-negligible for a reduction $\mathcal{R}$ making q queries where q is a polynomial in $\log p$, then $s = \Omega(k \log p)$ i.e. the memory required by any restricted black-box reduction grows with the number of queries by $\mathcal{A}$. Hence, there does not exist any efficient restricted black-box reduction from ODH to SDH that is memory-tight.

In Appendix C, we discuss how rewinding can slightly improve the memory complexity to (roughly) $O(k \log k)$, with heavy computational cost (essentially, one rewinding per oracle query of the adversary). We conjecture this to be optimal, but a proof seems to evade current techniques.

De-randomization. Before we turn to the proof – which also connects several technical lemmas presented across the next sections, let us discuss some aspects of the results. As explained above, our model allows for the adversary $\mathcal{A}$ to be run with randomness unknown to $\mathcal{R}$. This aspect *may* be controversial, but we note that there is a generic way for $\mathcal{A}$ to be made deterministic. Recall that $\mathcal{A}$ must be inefficient for the separation to even hold true. For example, $\mathcal{A}$ can use the injection σ from the generic group model to generate its random coin – say, using $\sigma^{-1}(\mathbf{a}_i)$ as coins a priori fixed labels $\mathbf{a}_1, \mathbf{a}_2, \ldots$. It is a standard – albeit tedious and omitted – argument to show that unless the reduction ends up querying the pre-images (which happens with negligible probability only), the $\sigma^{-1}(\mathbf{a}_i)$'s are good random coins.

Strengthening beyond SDH. We would like to note that our result can be strengthened without much effort to a reduction between ODH and a more general version of SDH. Informally, we can extend our result to every problem which is hard in the generic group model in presence of an O_v oracle. For example, this could be a problem where given g, g^u, and g^v, the attacker needs to output $g^{f(u,v)}$, where f is (a fixed) two-variate polynomial with degree at least 2. We do not include the proof for the strengthened version for simplicity. However, it appears much harder to extend our result to different types of oracles than O_v, as our proof is tailored at this oracle.

Proof. Here, we give the overall structure, the key lemmas, and how they are combined – quantitatively – to obtain the final result.

First off, Lemma 1 establishes that there exists an adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ is close to 1, which we will fix (i.e., when we refer to $\mathcal{A}$, we refer to the one guaranteed to exist by the lemma). The proof of Lemma 1 is in Section 4.1 and the proof of Lemma 2 is in Section 4.2.

Lemma 1. *There exists an adversary $\mathcal{A}$ and a function $\epsilon_1(p, \mathsf{hLen})$ such that is negligible in $\log p, \mathsf{hLen}$, and*

$$\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) = 1 - \epsilon_1(p, \mathsf{hLen}) \ .$$

After that, we introduce a game, called $\mathbb{G}_1$ and described in Figure 3 in Section 4.2. Very informally, this is a game played by a two-stage adversary $\mathcal{R}_1, \mathcal{R}_2$ which can pass a state to each other of size s bits and have access to the $\mathsf{Eval}, \mathsf{O_v}$ oracles. The game captures the essence of the reduction $\mathcal{R}$ the adversary $\mathcal{A}$ of having a sufficient amount of memory. This is made formal in Lemma 2, where we show that the probability of the reduction $\mathcal{R}$ winning $\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$ while running $\mathcal{A}$ is bounded by the probability of winning $\mathbb{G}_1$.

Lemma 2. *For every restricted black box reduction $\mathcal{R}$ that runs $\mathcal{A}$ while playing $\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$, there exist adversaries $\mathcal{R}_1, \mathcal{R}_2$ playing $\mathbb{G}_1$, such that the number of queries made by $\mathcal{R}_1, \mathcal{R}_2$ to $\mathsf{Eval}, \mathsf{O_v}$ is same as the number of queries made by $\mathcal{R}$ to $\mathsf{Eval}, \mathsf{O_v}$, the state passed from $\mathcal{R}_1$ to $\mathcal{R}_2$ is upper bounded by the memory used by $\mathcal{R}$ and,*

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] + \frac{4k^2(\log p)^2}{p} + \frac{4qk\log p + q^2}{p} \ .$$

We introduce Games $\mathbb{G}_2, \mathbb{G}_3$ in Figure 5 in Section 4.2. These games are identical to $\mathbb{G}_1$ except for the condition to output true. The condition to output true in these games are disjoint and the disjunction of the two conditions is equivalent to the condition to output true in $\mathbb{G}_1$. A little more specifically, both games depend on a parameter l, which can be set arbitrarily, and in $\mathbb{G}_3$ and $\mathbb{G}_2$ the winning condition of $\mathbb{G}_1$ is strengthened by additional ensuring that a certain set defined during the execution of the game is smaller or larger than l, respectively. Therefore, tautologically,

$$\Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] + \Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] \ . \tag{1}$$

We now prove the following two lemmas below, in Sections 4.3 and 4.4,

Lemma 3. *If $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries to their oracles in total in the game $\mathbb{G}_2$ as defined in Figure 5, then*

$$\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} \ .$$

Lemma 4. *If the size of the state ϕ output by $\mathcal{R}_1$ is s bits and $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries in total in the game $\mathbb{G}_3$ as defined in Figure 5, then*

$$\Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1 + \frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{k^2+k+2}{p} \ .$$

Combining (1) and the result of Lemmas 3 and 4 we get,

$$\Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1 + \frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{2(k^2+k+2)}{p} + \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} \ . \tag{2}$$

Since $\left(1 + \frac{6q}{p}\right)^{\frac{2q-l}{2}} \leqslant \left(1 + \frac{6q}{p}\right)^{q}$, combining Lemma 2, (2) we get,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1 + \frac{6q}{p}\right)^{q} + \frac{2(k^2+k+2)}{p} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{4k^2(\log p)^2}{p} + \frac{4qk\log p + q^2}{p} + \frac{q^l}{k!} \ .$$

We let,

$$\epsilon_2(p) = \frac{q^l}{k!} + \frac{2(k^2+k+2)}{p} + \frac{4k^2(\log p)^2}{p} \ .$$

Setting $c = \lceil \frac{\log q}{\log k} \rceil$ and $l = \frac{k}{4c}$, $\frac{q^l}{k!} \leqslant \frac{k^{k/4}}{k!}$. By Sterling's approximation $k! \geqslant k^{k+1/2}e^{-k}$. Therefore,

$$\frac{k^{k/4}}{k!} = \frac{k^{k/4}}{k^{k/4}} \frac{e^k}{k^{k/4}} \frac{1}{k^{k/2+1/2}} \ .$$

For $k > e^4$ (we can set $k > e^4$), $\frac{q^l}{k!} \leqslant \frac{1}{k^{k/2+1/2}}$ i.e. $\frac{q^l}{k!}$ is negligible in $\log p$ for k polynomial in $\log p$. Also, $\frac{2(k^2+k+2)}{p} + \frac{4k^2(\log p)^2}{p}$ is negligible in $\log p$ (since k is a polynomial in $\log p$). So, $\epsilon_2(p)$ is negligible in $\log p$. We have that,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant & 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{k}{8c}} \left(1+\frac{6q}{p}\right)^q + \\ & \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{4qk\log p + q^2}{p} + \epsilon_2(p) \ .\end{aligned}$$

where $c = 4\lceil \frac{\log q}{\log k} \rceil$. Assuming $q \geqslant k$ (and thus $q > e^4 > 2$), we get,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{48q^3}{p}\right)^{\frac{k}{8c}} \left(1+\frac{6q}{p}\right)^q + \frac{4q^2\log p + 13q^2 + 5q}{p} + \epsilon_2(p) \ .$$

□

4 Proof of Theorem

4.1 Adversary $\mathcal{A}$ against ODH

In this section, we construct the ODH adversary $\mathcal{A}$ needed for the proof.

Lemma 1. *There exists an adversary $\mathcal{A}$ and a function $\epsilon_1(p, \mathsf{hLen})$ such that is negligible in $\log p, \mathsf{hLen}$, and*

$$\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) = 1 - \epsilon_1(p, \mathsf{hLen}) \ .$$

Proof. The adversary $\mathcal{A}$ is formally defined in Figure 2. Adversary $\mathcal{A}$ samples $i_1, \cdots, i_k$ from $\mathbb{Z}_p$, and computes $\sigma(i_j), \sigma(i_j \cdot v)$ for all j in $[k]$. It then makes $\mathsf{H_v}$ queries on $\sigma(i_j)$'s for all j in $[k]$. Adversary $\mathcal{A}$ then samples a permutation π on $[k] \to [k]$, and then makes H queries on $\sigma(i_{\pi(j)} \cdot v)$'s for all j in $[k]$. If answers of all the H queries are distinct and the answers of all the $\mathsf{H_v}$ queries are distinct and for all j in $[k]$, $\mathsf{H_v}(\sigma(i_j)) = \mathsf{H}(\sigma(i_j \cdot v))$, $\mathcal{A}$ computes the discrete logarithm of V outputs the correct answer. Otherwise it returns a bit sampled uniformly at random. Note that $\mathcal{A}$ is inefficient, but only if it is satisfied from the responses it gets from the reduction using it.

First off, we note that adversary $\mathcal{A}$ does not use its input W until after the flag honest is set. So, W does not affect the setting of honest in any way. Since W is the only difference among the inputs of $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ and $\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ to $\mathcal{A}$, $\Pr[\mathsf{honest} = 0]$ and $\Pr[\mathsf{honest} = 1]$ are equal in both games. The flag honest is set to 0 in the following two cases.

1. for some distinct $j, l \in [k]$, $\mathsf{ans}_1[j] = \mathsf{ans}_1[l]$ or $\mathsf{ans}_2[j] = \mathsf{ans}_2[l]$). i.e.

$$\exists j, l \in [k], j \neq l : \mathsf{H}(\sigma(i_j)) = \mathsf{H}(\sigma(i_l)) \vee \mathsf{H}(\sigma(i_j \cdot v)) = \mathsf{H}(\sigma(i_l \cdot v)) \ .$$

We name this event E_1. Note that if for some distinct $j, l \in [k]$, $i_j = i_l$, then the event E_1 happens with probability 1 since $\mathsf{H}(i_j) = \mathsf{H}(i_l)$. We compute the probability of that there exists some distinct $j, l \in [k]$, such that $i_j = i_l$.

$$\Pr[\exists j, l \in [k], j \neq l : i_j = i_l] = \sum_{j=1}^{k} \frac{j-1}{p} = \frac{k(k-1)}{p} \ .$$

Also, note that if $v = 0$, E_1 happens with probability 1 since $\mathsf{H}(\sigma(i_j \cdot 0)) = \mathsf{H}(\sigma(0)) = \mathsf{H}(\sigma(i_l \cdot 0))$. Since v is sampled uniformly at random from $\mathbb{Z}_p$, $\Pr\left[v = 0\right] = \frac{1}{p}$.

We define the event E_2 as follows.

$$(\nexists j, l \in [k], j \neq l : i_j = i_l) \wedge (v \neq 0) \ .$$

Using the union bound,

$$\Pr\left[\neg E_2\right] \leqslant \frac{k(k-1)}{p} + \frac{1}{p} \ .$$

Since, σ is an injective function, E_2 implies

$$\nexists j, l \in [k], j \neq l : \sigma(i_j) = \sigma(i_l) \vee \sigma(i_j \cdot v) = \sigma(i_l \cdot v) \ .$$

Since H is a random oracle, the collision probability is,

$$\Pr\left[\mathsf{H}(r) = \mathsf{H}(s) \,\middle|\, r \neq s\right] = \frac{1}{2^{\mathsf{hLen}}} \ .$$

Therefore, using the union bound we have,

$$\Pr\left[E_1 \,\middle|\, E_2\right] \leqslant \frac{k(k-1)}{2^{\mathsf{hLen}}} \ .$$

Hence,

$$\Pr\left[E_1\right] \leqslant \Pr\left[\neg E_2\right] + \Pr\left[E_1 \,\middle|\, E_2\right] \leqslant \frac{k(k-1)+1}{p} + \frac{k(k-1)}{2^{\mathsf{hLen}}} \ .$$

2. for some $j \in [k]$, $\mathsf{ans}_1[j] \neq \mathsf{ans}_2[j]$ i.e.

$$\exists j \in [k] : \mathsf{H}_{\mathsf{v}}(\sigma(i_j)) \neq \mathsf{H}(\sigma(i_j \cdot v)) \ .$$

This is an impossible event in $\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}REAL\text{-}GG}}(\mathcal{A})$ and $\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}RAND\text{-}GG}}(\mathcal{A})$ by the definition of H_{v}.

Therefore,

$$\Pr\left[\mathsf{honest} = 0\right] = \Pr\left[E_1\right] \leqslant \frac{k(k-1)+1}{p} + \frac{k(k-1)}{2^{\mathsf{hLen}}} \ .$$

In $\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}REAL\text{-}GG}}(\mathcal{A})$, if $\mathsf{honest} = 1$, the output of $\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}REAL\text{-}GG}}(\mathcal{A})$ will always be 1 because $W = \mathsf{H}(\sigma(uv)) = \mathsf{H}(\mathsf{inp}) = W'$. Therefore,

$$\Pr\left[\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}REAL\text{-}GG}}(\mathcal{A}) \Rightarrow 1\right] \geqslant \Pr\left[\mathsf{honest} = 1\right] \geqslant 1 - \left(\frac{k(k-1)+1}{p} + \frac{k(k-1)}{2^{\mathsf{hLen}}}\right) \ .$$

In $\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}RAND\text{-}GG}}(\mathcal{A})$, if $\mathsf{honest} = 1$, the probability that $W = W'$ is at most $\frac{1}{2^{\mathsf{hLen}}}$ because W is sampled uniformly at random from the range of H in $\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}RAND\text{-}GG}}$. So we have,

$$\begin{aligned}\Pr\left[\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}RAND\text{-}GG}}(\mathcal{A}) \Rightarrow 1\right] &\leqslant \Pr\left[\mathsf{honest} = 0\right] \cdot \frac{1}{2} + \Pr\left[\mathsf{honest} = 1\right] \cdot \frac{1}{2^{\mathsf{hLen}}} \\ &\leqslant \frac{k(k-1)+1}{2p} + \frac{k(k-1)}{2^{\mathsf{hLen}+1}} + \frac{1}{2^{\mathsf{hLen}}}\left(1 - \frac{k(k-1)+1}{p} - \frac{k(k-1)}{2^{\mathsf{hLen}}}\right) \ .\end{aligned}$$

Hence,

$$\begin{aligned}&\Pr\left[\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}REAL\text{-}GG}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}RAND\text{-}GG}}(\mathcal{A}) \Rightarrow 1\right] \\ &\qquad \geqslant 1 - \left[\left(\frac{3k(k-1)+1}{2p} + \frac{3k(k-1)}{2^{\mathsf{hLen}+1}}\right) + \frac{1}{2^{\mathsf{hLen}}}\left(1 - \frac{k(k-1)+1}{p} - \frac{k(k-1)}{2^{\mathsf{hLen}}}\right)\right] \ .\end{aligned}$$

We let, $\epsilon_1(p, \mathsf{hLen}) = \left(\frac{3k(k-1)+1}{2p} + \frac{3k(k-1)}{2^{\mathsf{hLen}+1}}\right) + \frac{1}{2^{\mathsf{hLen}}}\left(1 - \frac{k(k-1)+1}{p} - \frac{k(k-1)}{2^{\mathsf{hLen}}}\right)$. Since k is polynomial in $\log p$, $\epsilon_1(p, \mathsf{hLen})$ is negligible in $\log p, \mathsf{hLen}$. Therefore,

$$\mathsf{Adv}_{p,\mathsf{hLen}}^{\mathsf{ODH\text{-}GG}}(\mathcal{A}) \geqslant 1 - \epsilon_1(p, \mathsf{hLen}) \ .$$

□

Adversary $\mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.),\mathsf{Eval}(.,.)}(U, V, W, \sigma(1))$:

$i_1, \cdots, i_k \leftarrow\$ \mathbb{Z}_p$
foreach $j \in [k]$ **do**
  $Q_1[j] \leftarrow \mathsf{Exp}(\sigma(1), i_j); Q_2[j] \leftarrow \mathsf{Exp}(V, i_j)$
$\mathsf{honest} \leftarrow 1$
foreach $j \in [k]$ **do**
  $\mathsf{ans}_1[j] \leftarrow \mathsf{H_v}(Q_1[j])$
$\pi \leftarrow\$ \mathcal{S}_k$
foreach $j \in [k]$ **do**
  $\mathsf{ans}_2[\pi(j)] \leftarrow \mathsf{H}(Q_2[\pi(j)])$
if $\exists j, l \in [k], j \neq l : (\mathsf{ans}_1[j] = \mathsf{ans}_1[l] \vee \mathsf{ans}_2[j] = \mathsf{ans}_2[l])$ **then** $\mathsf{honest} \leftarrow 0$
if $\exists j \in [k] : \mathsf{ans}_1[j] \neq \mathsf{ans}_2[j]$ **then** $\mathsf{honest} \leftarrow 0$
if $\mathsf{honest} = 1$ **then**
  $\mathsf{temp} \leftarrow \sigma(1); v \leftarrow 1$
  while $(\mathsf{temp} \neq V)$
    $\mathsf{temp} \leftarrow \mathsf{Eval}(\mathsf{temp}, \sigma(1)); v \leftarrow v + 1$
  $\mathsf{inp} \leftarrow \mathsf{Exp}(U, v); W' \leftarrow \mathsf{H}(\mathsf{inp}); b \leftarrow (W' = W)$
else $b \leftarrow\$ \{0, 1\}$
return b

Fig. 2. The adversary $\mathcal{A}$

4.2 The Shuffling Games

The Game $\mathbb{G}_1$. We first introduce the two-stage game $\mathbb{G}_1$ played by a pair of adversaries $\mathcal{R}_1$ and $\mathcal{R}_2$. (With some foresight, these are going to be two stages of the reduction.) It is formally described in Figure 3. Game $\mathbb{G}_1$ involves sampling $\sigma, i_1, \cdots, i_k, v$ from $\mathbb{Z}_p$, then running $\mathcal{R}_1$, followed by sampling permutation π from $\mathcal{S}_k$ and then running $\mathcal{R}_2$. The first stage $\mathcal{R}_1$ has inputs $\sigma(i_1), \cdots, \sigma(i_k)$ and it outputs a state ϕ of s bits along with k strings in $\{0,1\}^{\mathsf{hLen}}$. The second stage $\mathcal{R}_2$ has inputs $\phi, \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v)$ and it outputs k strings in $\{0,1\}^{\mathsf{hLen}}$. Both the stages $\mathcal{R}_1, \mathcal{R}_2$ have access to oracles $\mathsf{Eval}, \mathsf{O_v}$. Game $\mathbb{G}_1$ outputs true if all the k strings output by $\mathcal{R}_1$ are distinct, and if all the k strings output by $\mathcal{R}_2$ are distinct, and if for all $j \in [k]$, the j^{th} string output by $\mathcal{R}_2$ is identical to the $\pi(j)^{\text{th}}$ string output by $\mathcal{R}_1$. Additionally, $\mathbb{G}_1$ involves some bookkeeping. The $\mathsf{Eval}, \mathsf{O_v}$ oracles in $\mathbb{G}_1$ take an extra parameter named from as input which indicates whether the query was from $\mathcal{R}_1$ or $\mathcal{R}_2$.

We introduce the phrase "seen by" before describing the bookkeeping. A label has been "seen by" $\mathcal{R}_1$ if it was an input to $\mathcal{R}_1$, queried by $\mathcal{R}_1$ or an answer to a previously made $\mathsf{Eval}(.,.,1)$ query. A label has been "seen by" $\mathcal{R}_2$ if it was an input to $\mathcal{R}_2$, queried by $\mathcal{R}_2$ or an answer to a previously made $\mathsf{Eval}(.,.,2)$ query. We describe the sets $\mathcal{X}, \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Z}$ which are used for bookkeeping in $\mathbb{G}_1$.

- The labels in $\mathcal{X}$ are answers to $\mathsf{Eval}(.,.,1)$ queries such that it has not yet been "seen by" $\mathcal{R}_1$ before the query.
- $\mathcal{Y}_1$ contains all the labels that are input to $\mathcal{R}_1$, queried by $\mathcal{R}_1$ or answers to $\mathsf{Eval}(.,.,1)$ queries i.e. it is the set of labels "seen by" $\mathcal{R}_1$.
- $\mathcal{Y}_2$ contains all the labels that are input to $\mathcal{R}_2$, queried by $\mathcal{R}_1$ or answers to $\mathsf{Eval}(.,.,2)$ queries i.e. it is the set of labels "seen by" $\mathcal{R}_2$.
- All labels in $\mathcal{Z}$ are queried by $\mathcal{R}_2$ and have not been "seen by" $\mathcal{R}_2$ before the query and are in $\mathcal{X}$

The following lemma tells us that we can (somewhat straightforwardly) take a reduction as in the theorem statement, and transform it into an equivalent pair $\mathcal{R}_1, \mathcal{R}_2$ of adversaries for $\mathbb{G}_1$. The point here is that the reduction is very unlikely to succeed in breaking the SDH assumption without doing an effort equivalent to winning $\mathbb{G}_1$ to get $\mathcal{A}$'s help – otherwise, it is left with breaking SDH directly in the generic group model, which is hard.

Game $\mathbb{G}_1$:

1 : $\sigma \leftarrow\$ \mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L}); i_1, \cdots, i_k, v \leftarrow\$ \mathbb{Z}_p$
2 : $\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$
3 : $\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$
4 : $\pi \leftarrow\$ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$
5 : $s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$
6 : $\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$
7 : **return** win

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : $\mathbf{c} \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$
2 : **if** from $= 1$ **then**
3 : **if** $\mathbf{c} \notin \mathcal{Y}_1$ **then** $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$
4 : $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$
5 : **if** from $= 2$ **then**
6 : **if** $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$
7 : **if** $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$
8 : $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$
9 : **return** $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** from $= 1$ **then** $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$
2 : **if** from $= 2$ **then**
3 : **if** $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$
4 : **if** $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$
5 : $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$
6 : **return** $(v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b}))$

Fig. 3. Game $\mathbb{G}_1$. We use the phrase $\mathcal{R}_1, \mathcal{R}_2$ win $\mathbb{G}_1$ to mean $\mathbb{G}_1 \Rightarrow \mathsf{true}$. We shall use this convention for all games in the paper that output boolean values.

Adversary $\mathcal{B}^{\mathsf{H_v}(.), \mathsf{H}(.), \mathsf{Eval}(.,.)}(U, V, W, \sigma(1))$:

1 : $i_1, \cdots, i_k \leftarrow\$ \mathbb{Z}_p$
2 : **foreach** $j \in [k]$ **do**
3 : $Q_1[j] \leftarrow \mathsf{Exp}(\sigma(1), i_j); Q_2[j] \leftarrow \mathsf{Exp}(V, i_j)$
4 : $\mathsf{honest} \leftarrow 1$
5 : **foreach** $j \in [k]$ **do**
6 : $\mathsf{ans}_1[j] \leftarrow \mathsf{H_v}(Q_1[j])$
7 : $\pi \leftarrow\$ \mathcal{S}_k$
8 : **foreach** $j \in [k]$ **do**
9 : $\mathsf{ans}_2[\pi(j)] \leftarrow \mathsf{H}(Q_2[\pi(j)])$
10 : **if** $\exists j, l \in [k], j \neq l : (\mathsf{ans}_1[j] = \mathsf{ans}_1[l] \vee \mathsf{ans}_2[j] = \mathsf{ans}_2[l])$ **then** $\mathsf{honest} \leftarrow 0$
11 : **if** $\exists j \in [k], j \neq l : \mathsf{ans}_1[j] \neq \mathsf{ans}_2[j]$ **then** $\mathsf{honest} \leftarrow 0$
12 : $b \leftarrow\$ \{0, 1\}$
13 : **return** b

Fig. 4. Adversary $\mathcal{B}$

Proof. We introduce the adversary $\mathcal{B}$ which is identical to $\mathcal{A}$ till setting of honest (line 11) and then returns a bit uniformly at random. It is formally defined in Figure 4. Since the value honest does not change after line 11 in either $\mathcal{A}$ or $\mathcal{B}$, the value of honest will be identical in $\mathcal{A}, \mathcal{B}$. Also, observe that if honest is set to 0 in $\mathcal{A}$, $\mathcal{A}$'s output is identical to that of $\mathcal{B}$.

Consider any restricted black-box reduction $\mathcal{R}$ from ODH to SDH in the random oracle and generic group model that has oracle access to $\mathcal{A}$. During the execution of $\mathcal{R}^{\mathcal{A}}$ if $\mathsf{honest} = 0$ in $\mathcal{A}$, then it has the same output as $\mathcal{B}$. So, Therefore,

$$\Pr\left[\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{SDH\text{-}GG}}(\mathcal{R}^{\mathcal{A}}) \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{p,\mathsf{hLen}}^{\mathsf{SDH\text{-}GG}}(\mathcal{R}^{\mathcal{B}}) \Rightarrow \mathsf{true}\right] + \Pr\left[\mathsf{honest} = 1 \text{ in } \mathcal{A} \text{ during the execution of } \mathcal{R}^{\mathcal{A}}\right]. \tag{3}$$

The adversary $\mathcal{R}^{\mathcal{B}}$ makes at most $2k \log p + q$ queries to its oracles. From the hardness of SDH in the generic group model [1], it follows

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{B}}) \leqslant \frac{4k^2(\log p)^2}{p} + \frac{4qk \log p + q^2}{p} \ . \tag{4}$$

We show that if $\mathsf{honest} = 1$ in $\mathcal{A}$ when $\mathcal{R}$ runs $\mathcal{A}$, $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2)$ can run $\mathcal{R}$ and win $\mathbb{G}_1$. We next describe how $\mathcal{R}_1, \mathcal{R}_2$ run and simulate $\mathcal{A}$, $\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$ to $\mathcal{R}$.

Recall that $\mathcal{R}_1$ has inputs $\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)$. First off, $\mathcal{R}_1$ samples u uniformly at random from $\mathbb{Z}_p$ and computes $\sigma(u) = \mathsf{Exp}(\sigma(1), u)$. It then starts running $\mathcal{R}$ with inputs $\sigma(1), \sigma(u), \sigma(v)$ and simulates the oracles $\mathsf{Eval}, \mathsf{O_v}$ to $\mathcal{R}$. For every $\mathsf{Eval}(\mathbf{a}, \mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_1$ makes an $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 1)$ query to its own oracle and forwards the answer to $\mathcal{R}$. For every $\mathsf{O_v}(\mathbf{a}, \mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_1$ makes an $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 1)$ query to its own oracle and forwards the answer to $\mathcal{R}$. $\mathcal{R}_1$ makes $\mathsf{H_v}$ queries to $\mathcal{R}$ on $\sigma(i_1), \cdots, \sigma(i_k)$ and receives responses $s_1, \cdots, s_k$. $\mathcal{R}_1$ stops the execution of $\mathcal{R}$ after making the $\mathsf{H_v}$ queries, records the local state st of $\mathcal{R}$. Finally, $\mathcal{R}_1$ outputs st along with $s_1, \cdots, s_k$. Recall that $\mathcal{R}_2$ has inputs $\mathsf{st}, \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v)$. First, $\mathcal{R}_2$ restarts $\mathcal{R}$ from state st. For every $\mathsf{Eval}(\mathbf{a}, \mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_2$ makes an $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 2)$ query to its own oracle and forwards the answer to $\mathcal{R}$. For every $\mathsf{O_v}(\mathbf{a}, \mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_2$ makes an $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 2)$ query to its own oracle and forwards the answer to $\mathcal{R}$. $\mathcal{R}_2$ makes H queries to $\mathcal{R}$ on $\sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v)$ and receives responses $s'_1, \cdots, s'_k$. Finally, $\mathcal{R}_2$ outputs $s'_1, \cdots, s'_k$.

Observe that σ, u, v are sampled identically in $\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}), \mathbb{G}_1$, $i_1, \cdots, i_k, \pi$ are sampled identically in $\mathcal{A}, \mathbb{G}_1$ and $\mathcal{R}_1, \mathcal{R}_2$ make the same queries to $\mathcal{R}$ as $\mathcal{A}$ does in the same order. It follows that $\mathcal{R}_1, \mathcal{R}_2$ perfectly simulate $\mathcal{A}$, $\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$ to $\mathcal{R}$. Also note that the number of queries made by $\mathcal{R}_1, \mathcal{R}_2$ to $\mathsf{Eval}, \mathsf{O_v}$ is same as the number of queries made by $\mathcal{R}$ to $\mathsf{Eval}, \mathsf{O_v}$ and the state passed from $\mathcal{R}_1$ to $\mathcal{R}_2$ is upper bounded by the memory used by $\mathcal{R}$ because ϕ is the local state st of $\mathcal{R}$ at a certain point in its execution.

Next we relate the probabilities of $\mathsf{honest} = 1$ in $\mathcal{A}$ and $\mathbb{G}_1 \Rightarrow \mathsf{true}$. If $\mathsf{honest} = 1$ in $\mathcal{A}$ we have the following.

1. There does not exist distinct $j, l \in [k]$ such that $\mathsf{ans}_1[j] = \mathsf{ans}_1[l]$. The equivalent condition in $\mathbb{G}_1$ is $\forall j, l \in [k] : j \neq l \implies s_j \neq s_l$.
2. There does not exist distinct $j, l \in [k]$ such that $\mathsf{ans}_2[j] = \mathsf{ans}_2[l]$. The equivalent condition in $\mathbb{G}_1$ is $\forall j, l \in [k] : j \neq l \implies s'_j \neq s'_l$.
3. For all $j \in [k]$, $\mathsf{ans}_1[j] = \mathsf{ans}_2[j]$. The equivalent condition in $\mathbb{G}_1$ is $\forall j \in [k] : s_{\pi(j)} = s'_j$.

It follows that whenever $\mathsf{honest} = 1$ in $\mathcal{A}$, $\mathbb{G}_1 \Rightarrow \mathsf{true}$. Hence,

$$\Pr\left[\mathsf{honest} = 1 \text{ in } \mathcal{A} \text{ during the execution of } \mathcal{R}^{\mathcal{A}}\right] \leqslant \Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] \ . \tag{5}$$

From (3) and (5), it follows that,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{B}}) + \Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] \ . \tag{6}$$

Combining (4) and (6), we have,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] + \frac{4k^2(\log p)^2}{p} + \frac{4qk \log p + q^2}{p} \ . \tag{7}$$

□

Lemma 2. *For every restricted black box reduction $\mathcal{R}$ that runs $\mathcal{A}$ while playing $\mathbb{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$, there exist adversaries $\mathcal{R}_1, \mathcal{R}_2$ playing $\mathbb{G}_1$, such that the number of queries made by $\mathcal{R}_1, \mathcal{R}_2$ to $\mathsf{Eval}, \mathsf{O_v}$ is same as the number of queries made by $\mathcal{R}$ to $\mathsf{Eval}, \mathsf{O_v}$, the state passed from $\mathcal{R}_1$ to $\mathcal{R}_2$ is upper bounded by the memory used by $\mathcal{R}$ and,*

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] + \frac{4k^2(\log p)^2}{p} + \frac{4qk \log p + q^2}{p} \ .$$

Game $\boxed{\mathbb{G}_2}, \boxed{\mathbb{G}_3}$:

1 : $\sigma \leftarrow\$ \mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L}); i_1, \cdots, i_k, v \leftarrow\$ \mathbb{Z}_p$

2 : $\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$

3 : $\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(\cdot,\cdot,\cdot,1), \mathsf{O_v}(\cdot,\cdot,\cdot,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

4 : $\pi \leftarrow\$ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$

5 : $s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(\cdot,\cdot,\cdot,2), \mathsf{O_v}(\cdot,\cdot,\cdot,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

6 : $\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

7 : $\boxed{\textbf{return } (\mathsf{win} \wedge |\mathcal{Z}| < l)}$ $\boxed{\textbf{return } (\mathsf{win} \wedge |\mathcal{Z}| \geqslant l)}$

Fig. 5. Games $\mathbb{G}_2, \mathbb{G}_3$. The $\mathsf{Eval}, \mathsf{O_v}$ oracles in $\mathbb{G}_2, \mathbb{G}_3$ are identical to those in $\mathbb{G}_1$ and hence we do not rewrite it here. The newly introduced changes compared to $\mathbb{G}_1$ are highlighted. The statement within the thinner box is present only in $\mathbb{G}_3$ and the statement within the thicker box is present only in $\mathbb{G}_2$.

Game $\mathbb{PG}(\mathcal{A})$:	**Oracle** $O(\vec{x}, \vec{y})$: $\quad /\!/\ \vec{x} \in \mathbb{Z}_p^k, \vec{y} \in \mathbb{Z}_p^k$
1 : $\pi \leftarrow\$ \mathcal{S}_k$	1 : **return** $(\forall i \in [k] : \vec{x}[\pi(i)] = \vec{y}[i])$
2 : $\pi' \leftarrow \mathcal{A}^{O(\cdot,\cdot)}$	
3 : **return** $(\pi = \pi')$	

Fig. 6. The permutation game $\mathbb{PG}$ being played by adversary $\mathcal{A}$ is denoted by $\mathbb{PG}(\mathcal{A})$

The Games $\mathbb{G}_2$ and $\mathbb{G}_3$. In Figure 5 we define $\mathbb{G}_2, \mathbb{G}_3$ which have an added check on the cardinality of $\mathcal{Z}$ to output true. Everything else remains unchanged (in particular $\mathsf{Eval}, \mathsf{O_v}$ are unchanged from $\mathbb{G}_1$ and we do not specify them again here). The statement within the thinner box is present only in $\mathbb{G}_3$ and statement within the thicker box is present only in $\mathbb{G}_2$. The changes from $\mathbb{G}_1$ have been highlighted. We shall follow these conventions of using boxes and highlighting throughout the paper.

The games $\mathbb{G}_2, \mathbb{G}_3$ are identical to $\mathbb{G}_1$ except for the condition to output true. Since this disjunction of the conditions to output true in $\mathbb{G}_2, \mathbb{G}_3$ is equivalent to the condition to output true in $\mathbb{G}_1$, and the conditions to output true in $\mathbb{G}_2, \mathbb{G}_3$ are disjoint, we have,

$$\Pr\left[\mathbb{G}_1 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] + \Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] .$$

4.3 Proof of Lemma 3

Recall we are going to prove the following lemma.

Lemma 3. *If $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries to their oracles in total in the game $\mathbb{G}_2$ as defined in Figure 5, then*

$$\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} .$$

Proof. We introduce a new game – called the *permutation game* and denoted $\mathbb{PG}$ – in order to upper bound $\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right]$. In the rest of this proof, we are going to first define the game, and upper bound the winning probability of an adversary. Then, we are going to reduce an adversary for $\mathbb{G}_2$ to one for $\mathbb{PG}$.

The Permutation Game. In Game $\mathbb{PG}$, an adversary has to guess a randomly sampled permutation π over $[k]$. The adversary has access to an oracle that takes as input two vectors of length k and returns true if the elements of the first vector, when permuted using π, results in the second vector and false otherwise. Figure 6 formally describes the game $\mathbb{PG}$.

In the following, we say an adversary playing $\mathbb{PG}$ is a (q, l)-query adversary if it makes at most q queries to O, and the rank of the vectors that were the first argument to the O queries returning true is at most l.

The following lemma – which we prove via a compression argument – yields an upper bound on the probability of winning the game for a (q, l)-query adversary.

Procedure Encode(π) :	**Oracle** $O(\vec{x}, \vec{y})$:
1 : $c \leftarrow 0$	1 : $c \leftarrow c + 1$
2 : $S \leftarrow \varnothing$	2 : **if** $(\exists i \in [k] : \vec{x}[\pi(i)] \neq \vec{y}[i])$ **then**
3 : enc $\leftarrow \varnothing$	3 : **return** false
4 : $\pi' \leftarrow \mathcal{A}^{O(.,.)}$	4 : **else**
5 : **return** enc	5 : **if** $\vec{x} \notin \mathsf{span}(S)$ **then**
	6 : $S \leftarrow S \cup \{\vec{x}\}$
	7 : enc $\leftarrow$ enc $\cup \{c\}$
	8 : **return** true
Procedure Decode(enc) :	**Oracle** $O(\vec{x}, \vec{y})$:
1 : $c \leftarrow 0$	1 : $c \leftarrow c + 1$
2 : $S' \leftarrow \varnothing$	2 : **if** $c \in$ enc **then**
3 : $\pi' \leftarrow \mathcal{A}^{O(.,.)}$	3 : $S' \leftarrow S' \cup \{(\vec{x}, \vec{y})\}$
4 : **return** π'	4 : **return** true
	5 : **return** $((\vec{x}, \vec{y}) \in \mathsf{span}(S'))$

Fig. 7. Encoding and decoding π using $\mathcal{A}$

Lemma 5. *For a (q, l)-query adversary $\mathcal{A}$ playing $\mathbb{PG}$ the following is true.*

$$\Pr\left[\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} .$$

Proof. We construct an encoding of π by running adversary $\mathcal{A}$. In order to run $\mathcal{A}$, all the O queries need to be correctly answered. This can be naively done by storing the sequence number of queries whose answers are true. In fact, of all such queries, we need to just store the sequence number of just those whose first argument is not in the linear span of vectors which were the first argument of previous such queries i.e. we store the sequence number of only those O queries returning true whose first argument form a basis of the first argument of all O queries returning true. This approach works because for every vector $\vec{x}$, there is only a unique vector $\vec{y}$ such that $O(\vec{x}, \vec{y}) = 1$. The random tape of the adversary can be derived using the common randomness of Encode, Decode and hence the adversary produces identical queries and output. For simplicity, we do not specify this explicitly in the algorithms and treat $\mathcal{A}$ as deterministic. The formal description of the algorithms Encode, Decode are in Figure 7.

Observe that S is a basis of vectors $\vec{x}$ such that $O(\vec{x}, \vec{y}) = \mathsf{true}$. Note that for an $O(\vec{x}, \vec{y})$ query returning true, if $\vec{x} \in S$ then the sequence number of the query is stored in enc. Therefore, $(\vec{x}, \vec{y}) \in S'$ in Decode. Again, for an $O(\vec{x}, \vec{y})$ query returning true, if $\vec{x} \notin S$ then the sequence number of the query is not stored in enc and therefore $(\vec{x}, \vec{y}) \notin S'$. So, for an $O(\vec{x}, \vec{y})$ query returning true, $(\vec{x}, \vec{y}) \in S'$ iff $\vec{x} \in S$. Since, for all $(\vec{x}, \vec{y})$ such that $O(\vec{x}, \vec{y}) = \mathsf{true}$ we have that for all $i \in [k]$, $\vec{y}[i] = \vec{x}[\pi^{-1}(i)]$, it follows that S' forms a basis of vectors $(\vec{x}, \vec{y})$ such that $O(\vec{x}, \vec{y}) = \mathsf{true}$.

In Decode(enc), the simulation of $O(\vec{x}, \vec{y})$ is perfect because

- If c is in enc, then $\vec{x} \in S$ in Encode. From the definition of S in Encode, it follows that $O(\vec{x}, \vec{y})$ should return true.
- Otherwise we check if $(\vec{x}, \vec{y}) \in \mathsf{span}(S')$ and return true if the check succeeds, false otherwise. This is correct since in S' is a basis of vectors $(\vec{x}, \vec{y})$ such that $O(\vec{x}, \vec{y}) = \mathsf{true}$.

The encoding is a set of $|S|$ query sequence numbers. Since there are at most q queries, the encoding space is at most $\binom{q}{|S|}$. Using $\mathcal{X}$ to be the set S_k, $\mathcal{Y}$ to be the set of all possible encodings, $\mathcal{R}$ to be the set of random tapes of $\mathcal{A}$, it follows from Proposition 1 that,

$$\Pr\left[\text{Decoding is sucessful}\right] \leqslant \frac{\binom{q}{|S|}}{k!} .$$

Procedure $\mathsf{PopulateSetsEval}(\mathbf{a},\mathbf{b},\mathbf{c},\mathsf{from})$:	**Procedure** $\mathsf{PopulateSetsO_v}(\mathbf{a},\mathbf{b},\mathsf{from})$:
1 : **if** $\mathsf{from}=1$ **then**	1 : **if** $\mathsf{from}=1$ **then** $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a},\mathbf{b}\}$
2 : **if** $\mathbf{c}\notin\mathcal{Y}_1$ **then** $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$	2 : **if** $\mathsf{from}=2$ **then**
3 : $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a},\mathbf{b},\mathbf{c}\}$	3 : **if** $\mathbf{a}\in\mathcal{X}\backslash\mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$
4 : **if** $\mathsf{from}=2$ **then**	4 : **if** $\mathbf{b}\in\mathcal{X}\backslash\mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$
5 : **if** $\mathbf{a}\in\mathcal{X}\backslash\ \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$	5 : $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a},\mathbf{b}\}$
6 : **if** $\mathbf{b}\in\mathcal{X}\backslash\mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$	
7 : $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a},\mathbf{b},\mathbf{c}\}$	

Fig. 8. Subroutines $\mathsf{PopulateSetsEval}, \mathsf{PopulateSetsO_v}$

Since the simulation of $O(\vec{x},\vec{y})$ is perfect in Decode, decoding is successful if $\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}$. Therefore,

$$\Pr\left[\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}\right] \leqslant \frac{\binom{q}{|S|}}{k!} \leqslant \frac{q^{|S|}}{k!} .$$

Since $\mathcal{A}$ is a (q,l)-query adversary, $|S| \leqslant l$. Thus, we have,

$$\Pr\left[\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} \tag{8}$$

□

Reduction to $\mathbb{PG}$. We next show that the $\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right]$ is upper bounded in terms of the probability of a (q,l)-query adversary winning the game $\mathbb{PG}$.

Lemma 6. *There exists a (q,l)-query adversary $\mathcal{D}$ against the permutation game $\mathbb{PG}$ such that*

$$\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}\right] + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} .$$

Proof. We transform $\mathcal{R}_1,\mathcal{R}_2$ playing $\mathbb{G}_2$ to an adversary $\mathcal{D}$ playing the game $\mathbb{PG}$ through a sequence of intermediate games and use the upper bound on the probability of winning the game $\mathbb{PG}$ established previously to prove an upper bound on $\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right]$. In order to make the pseudocode for subsequent games compact we define the two subroutines $\mathsf{PopulateSetsEval}, \mathsf{PopulateSetsO_v}$ and invoke them from $\mathsf{Eval}, \mathsf{O_v}$. The subroutines $\mathsf{PopulateSetsEval}, \mathsf{PopulateSetsO_v}$ are formally described in Figure 8.

The game $\mathbb{G}_4$. We next describe game $\mathbb{G}_4$ where we introduce some additional bookkeeping. In $\mathbb{G}_4$, every valid label that is an input to $\mathcal{R}_1,\mathcal{R}_2$ or queried by $\mathcal{R}_1,\mathcal{R}_2$ or an answer to a query of $\mathcal{R}_1,\mathcal{R}_2$, is mapped to a polynomial in $\mathbb{Z}_p[I_1,\cdots,I_k,V,T_1,\cdots,T_{2q}]$ where q is the total number of $\mathsf{Eval}, \mathsf{O_v}$ queries made by $\mathcal{R}_1,\mathcal{R}_2$. The polynomial associated with label $\mathbf{a}$ is denoted by $\mathsf{p}_{\mathbf{a}}$. Similarly, we define Λ to be a mapping from polynomials to labels. For all labels $\mathbf{a}\in\mathcal{L}$, $\Lambda(\mathsf{p}_{\mathbf{a}}) = \mathbf{a}$. The mapping from labels to polynomials is done such that for every label $\mathbf{a}$ mapped to $\mathsf{p}_{\mathbf{a}}$,

$$\sigma^{-1}(\mathbf{a}) = \mathsf{p}_{\mathbf{a}}(i_1,\cdots,i_k,v,t_1,\cdots,t_{2q}) .$$

For compactness, let us denote $(i_1,\cdots,i_k,v,t_1,\cdots,t_{2q})$ by $\vec{i}$. Prior to running $\mathcal{R}_1$, polynomials $1,V,I_1,\cdots,I_k$, $I_1V,\cdots,I_kV$ are assigned to $\mathsf{p}_{\sigma(1)},\mathsf{p}_{\sigma(v)},\mathsf{p}_{\sigma(i_1)},\cdots,\mathsf{p}_{\sigma(i_k)},\mathsf{p}_{\sigma(i_1\cdot v)},\cdots,\mathsf{p}_{\sigma(i_k\cdot v)}$ respectively and for all other labels $\mathbf{a}\in\mathcal{L}$, $\mathsf{p}_{\mathbf{a}} = \perp$. The function Λ is defined accordingly. For labels $\mathbf{a}$ queried by $\mathcal{R}_1,\mathcal{R}_2$ that have not been previously mapped to any polynomial (i.e. $\mathsf{p}_{\mathbf{a}} = \perp$), $\mathsf{p}_{\mathbf{a}}$ is assigned T_{new} (new starting from 1 and being incremented for every such label queried), the variable t_{new} is assigned the pre-image of the label and $\Lambda(T_{\mathsf{new}})$ is assigned $\mathbf{a}$. Since there are q queries (each with two inputs), there can be at most $2q$ labels that had not previously been mapped to any polynomial. Hence, the polynomials have variables $I_1,\cdots,I_k,V,T_1,\cdots,T_{2q}$.

For an $\mathsf{Eval}(\mathbf{a},\mathbf{b},.)$ query where $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$, let $\mathsf{p}' = \mathsf{p}_{\mathbf{a}} + \mathsf{p}_{\mathbf{b}}$. From the definition of p, we have that $\mathsf{p}'(\vec{i}) = \sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b})$. If $\Lambda(\mathsf{p}') \neq \perp$, then by definition of Λ, we have $\Lambda(\mathsf{p}') = \mathbf{c}$. If $\Lambda(\mathsf{p}') = \perp$, then exactly one of the following two must be true.

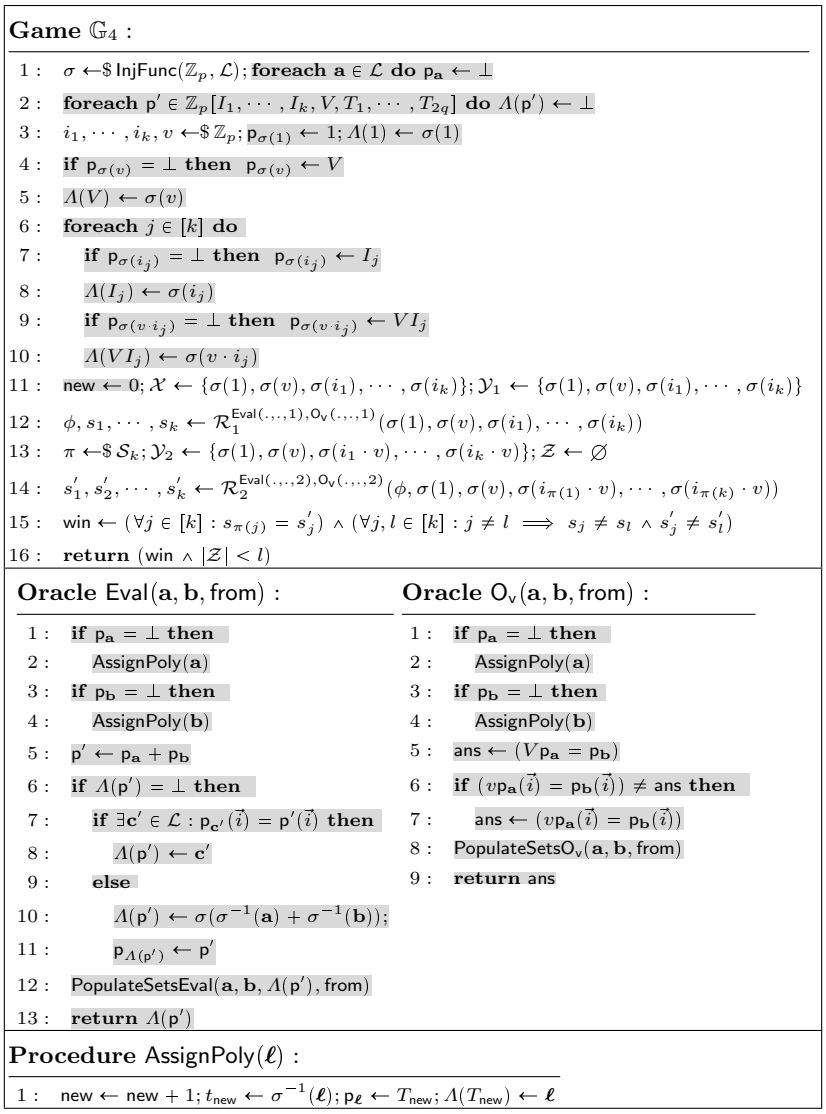

Fig. 9. $\mathbb{G}_4$ introduces additional bookkeeping. The newly introduced changes compared to $\mathbb{G}_2$ are highlighted.

1. The label $\mathbf{c}$ has been mapped to a polynomial which is different from p'. In this case $\mathsf{p}_{\mathbf{c}}(\vec{i}) = \mathsf{p}'(\vec{i})$ and $\Lambda(\mathsf{p}')$ is assigned $\mathbf{c}$.
2. The label $\mathbf{c}$ has not been mapped to any polynomial. In this case, $\mathsf{p}_{\mathbf{c}}$ is assigned p' and $\Lambda(\mathsf{p}')$ is assigned $\mathbf{c}$.

The label $\Lambda(\mathsf{p}')$ is returned as the answer of the Eval query. Note that the output of Eval is $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$ in all cases, i.e. it is the same as the output of Eval in $\mathbb{G}_2$.

For an $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, .)$ query, we first assign the boolean value $V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}}$ to ans. Note that if ans is true, then $v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b})$. However, we might have that $v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b})$ and $V\mathsf{p}_{\mathbf{a}} \neq \mathsf{p}_{\mathbf{b}}$. When this happens, the boolean value $v(\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))$ is assigned to ans. Oracle $\mathsf{O_v}$ returns ans. From the definition of p, it follows that the value returned by $\mathsf{O_v}$ in $\mathbb{G}_4$ is $(v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b}))$ i.e. it is the same as the output of $\mathsf{O_v}$ in $\mathbb{G}_2$.

Game $\mathbb{G}_5$:

1 : **foreach** $i \in \mathbb{Z}_p$ **do** $\sigma(i) \leftarrow \perp$; **foreach** $\mathbf{a} \in \mathcal{L}$ **do** $\mathsf{p}_{\mathbf{a}} \leftarrow \perp$
2 : **foreach** $\mathsf{p}' \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$ **do** $\Lambda(\mathsf{p}') \leftarrow \perp$
3 : $i_1, \cdots, i_k, v \leftarrow\$ \mathbb{Z}_p$; $\sigma(1) \leftarrow\$ \mathcal{L}$; $\mathsf{p}_{\sigma(1)} \leftarrow 1$; $\Lambda(1) \leftarrow \sigma(1)$
4 : **if** $\mathsf{p}_{\sigma(v)} = \perp$ **then** $\sigma(v) \leftarrow\$ \overline{R(\sigma)}$; $\mathsf{p}_{\sigma(v)} \leftarrow V$
5 : $\Lambda(V) \leftarrow \sigma(v)$
6 : **foreach** $j \in [k]$ **do**
7 : **if** $\mathsf{p}_{\sigma(i_j)} = \perp$ **then** $\sigma(i_j) \leftarrow\$ \overline{R(\sigma)}$; $\mathsf{p}_{\sigma(i_j)} \leftarrow I_j$
8 : $\Lambda(I_j) \leftarrow \sigma(i_j)$
9 : **if** $\mathsf{p}_{\sigma(v \cdot i_j)} = \perp$ **then** $\sigma(v \cdot i_j) \leftarrow\$ \overline{R(\sigma)}$; $\mathsf{p}_{\sigma(v \cdot i_j)} \leftarrow VI_j$
10 : $\Lambda(VI_j) \leftarrow \sigma(v \cdot i_j)$
11 : $\mathsf{new} \leftarrow 0$; $\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$; $\mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$
12 : $\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$
13 : $\pi \leftarrow\$ \mathcal{S}_k$; $\mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}$; $\mathcal{Z} \leftarrow \varnothing$
14 : $s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$
15 : $\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$
16 : **return** $(\mathsf{win} \wedge |\mathcal{Z}| < l)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\mathsf{p}_{\mathbf{a}} = \perp$ **then**
2 : $\mathsf{AssignPoly}(\mathbf{a})$
3 : **if** $\mathsf{p}_{\mathbf{b}} = \perp$ **then**
4 : $\mathsf{AssignPoly}(\mathbf{b})$
5 : $\mathsf{p}' \leftarrow \mathsf{p}_{\mathbf{a}} + \mathsf{p}_{\mathbf{b}}$
6 : **if** $\Lambda(\mathsf{p}') = \perp$ **then**
7 : **if** $\exists \mathbf{c}' \in \mathcal{L} : \mathsf{p}_{\mathbf{c}'}(\vec{i}) = \mathsf{p}'(\vec{i})$ **then**
8 : $\Lambda(\mathsf{p}') \leftarrow \mathbf{c}'$
9 : **else**
10 : $\sigma(\mathsf{p}'(\vec{i})) \leftarrow\$ \overline{R(\sigma)}$; $\Lambda(\mathsf{p}') \leftarrow \sigma(\mathsf{p}'(\vec{i}))$
11 : $\mathsf{p}_{\Lambda(\mathsf{p}')} \leftarrow \mathsf{p}'$
12 : $\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}'), \mathsf{from})$
13 : **return** $\Lambda(\mathsf{p}')$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\mathsf{p}_{\mathbf{a}} = \perp$ **then**
2 : $\mathsf{AssignPoly}(\mathbf{a})$
3 : **if** $\mathsf{p}_{\mathbf{b}} = \perp$ **then**
4 : $\mathsf{AssignPoly}(\mathbf{b})$
5 : $\mathsf{ans} \leftarrow (V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}})$
6 : **if** $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})) \neq \mathsf{ans}$ **then**
7 : $\mathsf{ans} \leftarrow (v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))$
8 : $\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$
9 : **return** ans

Procedure $\mathsf{AssignPoly}(\boldsymbol{\ell})$:

1 : $\mathsf{new} \leftarrow \mathsf{new} + 1$; $t_{\mathsf{new}} \leftarrow\$ \overline{D(\sigma)}$; $\sigma(t_{\mathsf{new}}) \leftarrow \boldsymbol{\ell}$; $\mathsf{p}_{\boldsymbol{\ell}} \leftarrow T_{\mathsf{new}}$; $\Lambda(T_{\mathsf{new}}) \leftarrow \boldsymbol{\ell}$

Fig. 10. $\mathbb{G}_5$ lazily samples σ. The newly introduced changes compared to $\mathbb{G}_4$ are highlighted.

Figure 9 formally describes $\mathbb{G}_4$. The changes in $\mathbb{G}_4$ compared to $\mathbb{G}_2$ have been highlighted. We have already pointed out that the outputs of $\mathsf{O_v}, \mathsf{Eval}$ in $\mathbb{G}_4$ are identical to those in $\mathbb{G}_2$. Since the other changes involve only additional bookkeeping, the outputs of $\mathbb{G}_2, \mathbb{G}_4$ are identical. Therefore

$$\Pr\left[\mathbb{G}_4 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] . \tag{9}$$

THE GAME $\mathbb{G}_5$. We define $\mathbb{G}_5$ in Figure 10 where σ is lazily sampled. Since σ is lazily sampled, it is a partial function and we use previously defined notations $D(\sigma), R(\sigma), \overline{D(\sigma)}, \overline{R(\sigma)}$. There are no other changes in $\mathbb{G}_5$ compared to $\mathbb{G}_4$.

In $\mathbb{G}_4$, σ is sampled uniformly at random from $\mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L})$. In $\mathbb{G}_5$, if $\sigma(i)$ is previously not defined, it is sampled uniformly from the values in the range that do not have a pre-image and similarly if $\sigma^{-1}(\mathbf{x})$ is previously not defined, it is sampled uniformly from the values in the domain that do not have a image. Therefore, the distribution of σ on the points in $D(\sigma)$ in $\mathbb{G}_5$ is identical to the distribution of σ on the

Game $\boxed{\mathbb{G}_6}, \mathbb{G}_7$:

Procedure AssignPoly($\boldsymbol{\ell}$) :

1 : $\mathsf{new} \leftarrow \mathsf{new} + 1; t_{\mathsf{new}} \leftarrow\$ \mathbb{Z}_p$

2 : **if** $\sigma(t_{\mathsf{new}}) \neq \perp$ **then** $\mathsf{bad} \leftarrow \mathsf{true}; \boxed{t_{\mathsf{new}} \leftarrow\$ \overline{D(\sigma)}}$

3 : $\sigma(t_{\mathsf{new}}) \leftarrow \boldsymbol{\ell}; \mathsf{p}_{\boldsymbol{\ell}} \leftarrow T_{\mathsf{new}}; \Lambda(T_{\mathsf{new}}) \leftarrow \boldsymbol{\ell}$

Fig. 11. In $\mathbb{G}_7$ we remove the restriction on t_j's. The main procedure of $\mathbb{G}_6, \mathbb{G}_7$ are identical to $\mathbb{G}_5$ and the only changes in the oracles are the AssignPoly procedure, hence we write only the AssignPoly procedure. The changes in the procedure from that in $\mathbb{G}_5$ have been highlighted. The boxed statement is present only in $\mathbb{G}_6$.

points in $D(\sigma)$ if σ was sampled from $\mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L})$. So, in $\mathbb{G}_5$, the distribution of σ on the points in $D(\sigma)$ is identical to that of $\mathbb{G}_4$. Since σ is never evaluated on points outside $D(\sigma)$ in either games and there are no other changes in $\mathbb{G}_5$, the outputs of $\mathbb{G}_4, \mathbb{G}_5$ are identical.

$$\Pr\left[\mathbb{G}_4 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_5 \Rightarrow \mathsf{true}\right] . \tag{10}$$

The Games $\mathbb{G}_6$ and $\mathbb{G}_7$. Next, we define Games $\mathbb{G}_6, \mathbb{G}_7$. The changes introduced in $\mathbb{G}_6$ from $\mathbb{G}_5$ have been highlighted (as before). In $\mathbb{G}_7$ we want to sample t_j's from $\mathbb{Z}_p$ instead of $\overline{D(\sigma)}$. So in $\mathbb{G}_6$ we first sample them from $\mathbb{Z}_p$ and define a bad event if $\sigma(t_j) \neq \perp$ i.e. $t_j \in D(\sigma)$. If this bad event happens, then we resample t_j from $\overline{D(\sigma)}$. We then remove this resampling in $\mathbb{G}_7$. The main procedure of $\mathbb{G}_6, \mathbb{G}_7$ are identical to $\mathbb{G}_5$ and the only changes in the oracles are the AssignPoly procedure, hence we rewrite only the AssignPoly procedure in Figure 11.

Observe from the pseudocode that outputs of $\mathbb{G}_5, \mathbb{G}_6$ are identical. Therefore,

$$\Pr\left[\mathbb{G}_6 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_5 \Rightarrow \mathsf{true}\right] . \tag{11}$$

After $\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)$ are sampled, the size of $D(\sigma)$ is at most $2k+2$. For every query $D(\sigma)$ can grow by size at most 3 (in case of Eval queries). Therefore, the size of $D(\sigma)$ is at most $2k + 3q + 2$. Since bad is set only when an element from $D(\sigma)$ is sampled,

$$\Pr\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathbb{G}_7\right] \leqslant \frac{2q(2k+3q+2)}{p} .$$

It is evident from their pseudocodes that $\mathbb{G}_6, \mathbb{G}_7$ are identical-until-bad. From the Fundamental Lemma Game Playing we get,

$$\Pr\left[\mathbb{G}_6 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_7 \Rightarrow \mathsf{true}\right] + \frac{2q(2k+3q+2)}{p} . \tag{12}$$

In $\mathbb{G}_7$, note that $R(\sigma) = R(\Lambda)$ and $D(\sigma)$ is not used anywhere. Also the condition $\mathsf{p}_{\sigma(j)} \neq \perp$ is equivalent to the condition that there exists some p such that $\Lambda(\mathsf{p}) \neq \perp$ and $\mathsf{p}(\vec{i}) = j$ and the condition there exists $\mathbf{c}' \in \mathcal{L}$ such that $\mathsf{p}_{\mathbf{c}'}(\vec{i}) = \mathsf{p}'(\vec{i})$ is equivalent to the condition there exists some p such that $\Lambda(\mathsf{p}) \neq \perp$ and $\mathsf{p}(\vec{i}) = \mathsf{p}'(\vec{i})$. Hence, $\mathbb{G}_7$ can be rewritten without σ.

The Games $\mathbb{G}_8$ and $\mathbb{G}_9$. Next, we introduce games $\mathbb{G}_8, \mathbb{G}_9$. In $\mathbb{G}_8$ we remove σ (we had previously pointed out $\mathbb{G}_7$ can be rewritten without σ), remove some redundant code and the bad event in $\mathbb{G}_7$ and add new bad events when there is some p such that $\Lambda(\mathsf{p}) \neq \perp$ and $\mathsf{p}(\vec{i}) = \mathsf{p}'(\vec{i})$ in Eval and $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})) \neq \mathsf{ans}$ in $\mathsf{O_v}$. Hence outputs of $\mathbb{G}_7, \mathbb{G}_8$ are identical.

$$\Pr\left[\mathbb{G}_7 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_8 \Rightarrow \mathsf{true}\right] . \tag{13}$$

Since $\vec{i}$ affects only the setting of bad in $\mathsf{Eval}, \mathsf{O_v}$ we see that $\vec{i}$ may well have been sampled at the very end of $\mathbb{G}_9$ and it can be checked if any of the $\mathsf{Eval}, \mathsf{O_v}$ queries would have set bad to true. The degree of any

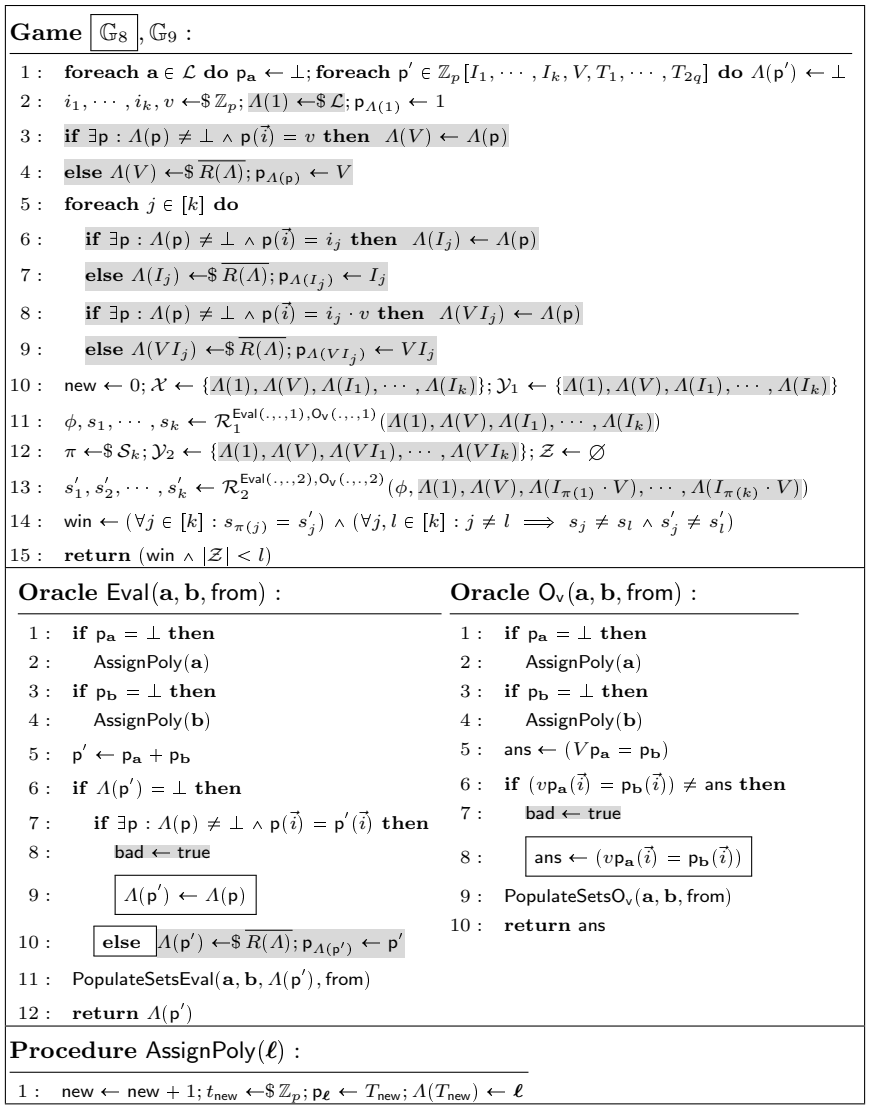

Fig. 12. Games $\mathbb{G}_8, \mathbb{G}_9$ introduce new bad events. We remove σ from these games. The changes from $\mathbb{G}_7$ have been highlighted. The boxed statements are present only in $\mathbb{G}_8$.

polynomial is at most 2 since every polynomial has monomials only of the form I_j, VI_j, T_j, V and constants. We compute the probability of bad $\leftarrow$ true in $\mathbb{G}_9$ in $\mathsf{Eval}, \mathsf{O_v}$.

In Eval, bad is set to true if $\Lambda(\mathsf{p}') = \bot$ and for some label $\mathbf{c}'$ $\mathsf{p}_{\mathbf{c}'}(\vec{i}) = \mathsf{p}'(\vec{i})$. From the definition of Λ, it follows that $\Lambda(\mathsf{p}') = \bot$ implies $\mathsf{p}_{\mathbf{c}'} \neq \mathsf{p}'$. We therefore have $\mathsf{p}'(\vec{i}) - \mathsf{p}_{\mathbf{c}'}(\vec{i}) = 0$. Note that $\mathsf{p}' - \mathsf{p}_{\mathbf{c}'}$ is not identically zero since $\Lambda(\mathsf{p}') = \bot$ and the degree of $\mathsf{p}' - \mathsf{p}_{\mathbf{c}'}$ is at most 2. Observe that the elements of $\vec{i}$ are sampled from $\mathbb{Z}_p$ and could have been sampled at the very end of $\mathbb{G}_9$ because $\vec{i}$ does not affect anything other than setting bad. So, the probability $\mathsf{p}'(\vec{i}) - \mathsf{p}_{\mathbf{c}'}(\vec{i}) = 0$ is bounded by $\frac{2}{p}$ by Proposition 2. Since there are at most q Eval queries, the probability that bad is set to true due to this condition in any of the Eval queries in $\mathbb{G}_9$ is bounded by $\frac{2q}{p}$ using the union bound.

In $\mathsf{O_v}$, bad is set to true is when $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})), (V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}})$ are different boolean values. Note that when $(V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}}) =$ true then $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))$. The only case when the two boolean values differ is when $(V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}}) =$ false and $v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})$ i.e. $(V\mathsf{p}_{\mathbf{a}} - \mathsf{p}_{\mathbf{b}})$ is not identically zero but $v\mathsf{p}_{\mathbf{a}}(\vec{i}) - \mathsf{p}_{\mathbf{b}}(\vec{i}) = 0$. Note that

the degree of $(V\mathsf{p_a} - \mathsf{p_b})$ is at most 3 (since $V\mathsf{p}_a$ can have degree at most 3) . Like the previous analysis since the elements of $\vec{i}$ are sampled from $\mathbb{Z}_p$ and could have been sampled at the very end of $\mathbb{G}_9$, the probability $v\mathsf{p_a}(\vec{i}) - \mathsf{p_b}(\vec{i}) = 0$ is bounded by $\frac{3}{p}$ by Proposition 2. Since there are at most q $\mathsf{O_v}$ queries, the probability that bad is set to true due to this condition in any of the $\mathsf{O_v}$ queries in $\mathbb{G}_9$ is bounded by $\frac{3q}{p}$ using the union bound. Therefore,

$$\Pr\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathbb{G}_9\right] \leqslant \frac{5q}{p} .$$

Note that $\mathbb{G}_8$ and $\mathbb{G}_9$ are identical-until-bad. Therefore, using the Fundamental Lemma of Game Playing we get,

$$\Pr\left[\mathbb{G}_8 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_9 \Rightarrow \mathsf{true}\right] + \frac{5q}{p} . \tag{14}$$

We note that in $\mathbb{G}_9$, Λ becomes an injective function since for $\mathsf{p} \neq \mathsf{p}'$, $\Lambda(\mathsf{p}) \neq \Lambda(\mathsf{p}')$. Therefore, Λ^{-1} is well defined. This means that for all labels $\boldsymbol{\ell}$, $\mathsf{p}_{\boldsymbol{\ell}}$ can be replaced by $\Lambda^{-1}(\boldsymbol{\ell})$ throughout $\mathbb{G}_9$. Also,$(V\mathsf{p_a} = \mathsf{p_b})$ is always returned from $\mathsf{O_v}$ and hence the check on ans can be omitted. The bad events can be removed from $\mathbb{G}_9$ because it does not affect its execution in any way. That in turn implies t_j's do not affect the execution of $\mathbb{G}_9$ and can also be removed. Since $\mathbb{G}_{10}$ only introduces a new bad event compared to $\mathbb{G}_9$ and the bad event does not affect the output,

$$\Pr\left[\mathbb{G}_9 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_{10} \Rightarrow \mathsf{true}\right] . \tag{15}$$

We need to compute the probability that bad is set to true in procedure $\mathsf{RestrictedSample}$ of $\mathbb{G}_{10}$. First we compute the probability that $v \in \{0,1\}$,

$$\Pr_{v \leftarrow\$ \mathbb{Z}_p}\left[v \in \{0,1\}\right] = \frac{2}{p} .$$

Note that just before i_j sampled in $\mathbb{G}_{10}$ the size of $\mathcal{S} \cup \mathcal{S}'$ is at most $2j$. We compute the probability that $i_j \in \mathcal{S} \cup \mathcal{S}'$ for some $j \in [k]$,

$$\Pr_{i_j \leftarrow\$ \mathbb{Z}_p}\left[i_j \in \mathcal{S} \cup \mathcal{S}'\right] \leqslant \frac{2j}{p} .$$

In $\mathbb{G}_{10}$, since bad can be set to true only in $\mathsf{RestrictedSample}$, the probability that bad is set to true. Therefore, using the union bound,

$$\Pr\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathbb{G}_{10}\right] \leqslant \frac{2}{p} + \sum_{j=1}^{k} \frac{2j}{p} \leqslant \frac{k^2 + k + 2}{p} .$$

Note that $\mathbb{G}_{10}$ and $\mathbb{G}_{11}$ are identical-until-bad. Therefore, using the Fundamental Lemma of Game Playing we get,

$$\Pr\left[\mathbb{G}_{10} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right] + \frac{k^2 + k + 2}{p} . \tag{16}$$

The Games $\mathbb{G}_{10}$ and $\mathbb{G}_{11}$. We introduce a procedure named $\mathsf{RestrictedSample}$ in order to sample $i_1, \cdots, i_k, v$ instead of sampling them uniformly at random, replace p by Λ^{-1}, remove t_j's and the bad events in $\mathbb{G}_9$. In $\mathbb{G}_{11}$, $\mathsf{RestrictedSample}$ sets bad to true if $v \in \{0,1\}$ or if the cardinality of the set $\{1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v\}$ is less than $2k+2$. In $\mathbb{G}_{11}$, $\mathsf{RestrictedSample}$ samples these values with the restriction that $|\{1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v\}| = 2k+2$.

We note that since $i_1, \cdots, i_k, v$ sampled in $\mathbb{G}_{11}$ satisfy $|1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v| = 2k+2$, the if statements in lines 3, 6, 8 will always evaluate to true. Since $i_1, \cdots, i_k, v$ do not affect the execution of $\mathbb{G}_{11}$ at any other point, the sampling of $i_1, \cdots, i_k, v$ and the if statements in lines 3, 6, 8 can be omitted from $\mathbb{G}_{11}$. Therefore, $\mathbb{G}_{11}$ can be rewritten as shown in Figure 14. We next upper bound $\Pr\left[\mathbb{G}_4 \Rightarrow \mathsf{true}\right]$ in terms of $\Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right]$ in Lemma 7.

Game $\mathbb{G}_{10}$, $\boxed{\mathbb{G}_{11}}$:

1 : **foreach** $\mathsf{p} \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$ **do** $\Lambda(\mathsf{p}) \leftarrow \bot$
2 : $i_1, \cdots, i_k, v \leftarrow$ RestrictedSample(); $\Lambda(1) \leftarrow\$ \mathcal{L}$
3 : **if** $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \bot \wedge \mathsf{p}(\vec{i}) = v$ **then** $\Lambda(V) \leftarrow \Lambda(\mathsf{p})$
4 : **else** $\Lambda(V) \leftarrow\$ \overline{R(\Lambda)}$
5 : **foreach** $j \in [k]$ **do**
6 : **if** $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \bot \wedge \mathsf{p}(\vec{i}) = i_j$ **then** $\Lambda(I_j) \leftarrow \Lambda(\mathsf{p})$
7 : **else** $\Lambda(I_j) \leftarrow\$ \overline{R(\Lambda)}$
8 : **if** $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \bot \wedge \mathsf{p}(\vec{i}) = i_j \cdot v$ **then** $\Lambda(VI_j) \leftarrow \Lambda(\mathsf{p})$
9 : **else** $\Lambda(VI_j) \leftarrow\$ \overline{R(\Lambda)}$
10 : new $\leftarrow 0$; $\mathcal{X} \leftarrow \Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$; $\mathcal{Y}_1 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$
11 : $\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(\cdot,\cdot,1),\mathsf{O_v}(\cdot,\cdot,1)}(\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k))$
12 : $\pi \leftarrow\$ \mathcal{S}_k$; $\mathcal{Y}_2 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(VI_1), \cdots, \Lambda(VI_k)\}$; $\mathcal{Z} \leftarrow \varnothing$
13 : $s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(\cdot,\cdot,2),\mathsf{O_v}(\cdot,\cdot,2)}(\phi, \Lambda(1), \Lambda(V), \Lambda(I_{\pi(1)} \cdot V), \cdots, \Lambda(I_{\pi(k)} \cdot V))$
14 : win $\leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$
15 : **return** (win $\wedge\ |\mathcal{Z}| < l$)

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\Lambda^{-1}(\mathbf{a}) = \bot$ **then**
2 : new $\leftarrow$ new $+ 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$
3 : **if** $\Lambda^{-1}(\mathbf{b}) = \bot$ **then**
4 : new $\leftarrow$ new $+ 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$
5 : $\mathsf{p} \leftarrow \Lambda^{-1}(\mathbf{a}) + \Lambda^{-1}(\mathbf{b})$
6 : **if** $\Lambda(\mathsf{p}) = \bot$ **then**
7 : $\Lambda(\mathsf{p}) \leftarrow\$ \overline{R(\Lambda)}$
8 : PopulateSetsEval($\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}), \mathsf{from}$)
9 : **return** $\Lambda(\mathsf{p})$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\Lambda^{-1}(\mathbf{a}) = \bot$ **then**
2 : new $\leftarrow$ new $+ 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$
3 : **if** $\Lambda^{-1}(\mathbf{b}) = \bot$ **then**
4 : new $\leftarrow$ new $+ 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$
5 : PopulateSets$\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$
6 : **return** $(V\Lambda^{-1}(\mathbf{a}) = \Lambda^{-1}(\mathbf{b}))$

Procedure RestrictedSample() :

1 : $v \leftarrow\$ \mathbb{Z}_p$; **if** $v \in \{0, 1\}$ **then** bad $\leftarrow$ true; $\boxed{v \leftarrow\$ \mathbb{Z}_p \backslash \{0, 1\}}$
2 : $\mathcal{S} \leftarrow \{1\}$; $\mathcal{S}' \leftarrow \{v^{-1}\}$
3 : **foreach** $j \in [k]$ **do**
4 : $i_j \leftarrow\$ \mathbb{Z}_p$; **if** $i_j \in \mathcal{S} \cup \mathcal{S}'$ **then** bad $\leftarrow$ true; $\boxed{i_j \leftarrow\$ \mathbb{Z}_p \backslash (\mathcal{S} \cup \mathcal{S}')}$
5 : $\mathcal{S} \xleftarrow{\cup} \{i_j\}$; $\mathcal{S}' \xleftarrow{\cup} \{v^{-1} \cdot i_j\}$
6 : **return** $i_1, \cdots, i_k, v$

Fig. 13. In games $\mathbb{G}_{10}, \mathbb{G}_{11}$ $\mathsf{p_a}$ has been replaced by $\Lambda^{-1}(\mathbf{a})$ for labels $\mathbf{a}$ and t_j's are removed. The procedure AssignPoly is no longer written separately, its code is written inline instead. In $\mathbb{G}_{11}$, i_j's, v are sampled with the restriction that the set $\{1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v\}$ has cardinality $2k + 2$. The changes from $\mathbb{G}_9$ have been highlighted. The boxed statements are present only in $\mathbb{G}_{11}$.

Lemma 7. *For the games $\mathbb{G}_4, \mathbb{G}_{11}$, we have,*

$$\Pr\left[\mathbb{G}_4 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right] + \frac{2q(2k + 3q + 2)}{p} + \frac{5q}{p} + \frac{k^2 + k + 2}{p}.$$

Proof. Combining (10) to (16) we get,

$$\Pr\left[\mathbb{G}_4 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right] + \frac{2q(2k + 3q + 2)}{p} + \frac{5q}{p} + \frac{k^2 + k + 2}{p}.$$

□

Game $\mathbb{G}_{11}$:

1 : **foreach** $\mathsf{p} \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$ **do** $\Lambda(\mathsf{p}) \leftarrow \perp; \Lambda(1) \leftarrow\$\, \mathcal{L}$
2 : $\Lambda(V) \leftarrow\$\, \overline{R(\Lambda)}$
3 : **foreach** $j \in [k]$ **do**
4 : $\Lambda(I_j) \leftarrow\$\, \overline{R(\Lambda)}; \Lambda(VI_j) \leftarrow\$\, \overline{R(\Lambda)}$
5 : $\mathsf{new} \leftarrow 0; \mathcal{X} \leftarrow \Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}; \mathcal{Y}_1 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$
6 : $\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(\cdot,\cdot,1),\mathsf{O_v}(\cdot,\cdot,1)}(\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k))$
7 : $\pi \leftarrow\$\, \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(VI_1), \cdots, \Lambda(VI_k)\}; \mathcal{Z} \leftarrow \varnothing$
8 : $s_1', s_2', \cdots, s_k' \leftarrow \mathcal{R}_2^{\mathsf{Eval}(\cdot,\cdot,2),\mathsf{O_v}(\cdot,\cdot,2)}(\phi, \Lambda(1), \Lambda(V), \Lambda(I_{\pi(1)} \cdot V), \cdots, \Lambda(I_{\pi(k)} \cdot V))$
9 : $\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s_j') \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s_j' \neq s_l')$
10 : **return** $(\mathsf{win} \wedge |\mathcal{Z}| < l)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\Lambda^{-1}(\mathbf{a}) = \perp$ **then**
2 : $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$
3 : **if** $\Lambda^{-1}(\mathbf{b}) = \perp$ **then**
4 : $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$
5 : $\mathsf{p} \leftarrow \Lambda^{-1}(\mathbf{a}) + \Lambda^{-1}(\mathbf{b})$
6 : **if** $\Lambda(\mathsf{p}) = \perp$ **then**
7 : $\Lambda(\mathsf{p}) \leftarrow\$\, \overline{R(\Lambda)}$
8 : $\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}), \mathsf{from})$
9 : **return** $\Lambda(\mathsf{p})$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\Lambda^{-1}(\mathbf{a}) = \perp$ **then**
2 : $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$
3 : **if** $\Lambda^{-1}(\mathbf{b}) = \perp$ **then**
4 : $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$
5 : $\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$
6 : **return** $(V\Lambda^{-1}(\mathbf{a}) = \Lambda^{-1}(\mathbf{b}))$

Fig. 14. Game $\mathbb{G}_{11}$ has been written more compactly than the one in Figure 13 by removing redundant code.

Procedure $\mathsf{PolyMultCheck}(\mathsf{p_a}, \mathsf{p_b})$:

1 : **if** $\exists j$: $(\mathsf{coefficient}(\mathsf{p_a}, T_j) \neq 0 \vee \mathsf{coefficient}(\mathsf{p_a}, VI_j) \neq 0)$ **then return** false
2 : **if** $\exists j$: $(\mathsf{coefficient}(\mathsf{p_b}, T_j) \neq 0 \vee \mathsf{coefficient}(\mathsf{p_b}, I_j) \neq 0)$ **then return** false
3 : **if** $\mathsf{coefficient}(\mathsf{p_b}, V) \neq \mathsf{coefficient}(\mathsf{p_a}, 1)$ **then return** false
4 : **foreach** $j \in [k]$ **do** $\vec{x}[j] \leftarrow \mathsf{coefficient}(\mathsf{p_a}, I_j)$; $\vec{y}[j] \leftarrow \mathsf{coefficient}(\mathsf{p_b}, VI_j)$
5 : **if** $O(\vec{x}, \vec{y}) = \mathsf{true}$ **then**
6 : **if** $\vec{x} \notin \mathsf{span}(S)$ **then** $S \xleftarrow{\cup} \{\vec{x}\}; \mathcal{Z}' \xleftarrow{\cup} \{\mathbf{a}\}$
7 : **if** $|S| = l$ **then** ABORT
8 : **return** true
9 : **else return** false

Fig. 15. Subroutine $\mathsf{PolyMultCheck}$ **for simulating** $\mathsf{O_v}$**.** In particular, $\mathsf{coefficient}(\mathsf{p}, M)$ returns the coefficient of the monomial M in the polynomial p. The sets S and $\mathcal{Z}'$ have no effect on the behavior, and are only used in the analysis of $\mathcal{D}$. The symbol ABORT indicates that $\mathcal{D}$ aborts and outputs $\perp$.

The Adversary $\mathcal{D}$. Next, we construct the adversary $\mathcal{D}$ that plays $\mathbb{PG}$ by simulating $\mathbb{G}_{11}$ to $\mathcal{R}_1, \mathcal{R}_2$, where the permutation π is the secret permutation from $\mathbb{PG}$. As we will discuss below, the core of the adversary $\mathcal{D}$ will boil down to properly simulating the $\mathsf{O_v}$ oracle using the O oracle from $\mathbb{PG}$ *and* simulating the labels $\sigma(i_{\pi(j)})$ (and the associated polynomials) correctly without knowing π. After a correct simulation, $\mathcal{D}$ will simply extract the permutation π.

To see how this can be done, let us first have a closer look at $\mathbb{G}_{11}$. Let us introduce the shorthand $K_j = VI_{\pi(j)}$ for $j \in [k]$. With this notation, every polynomial input to or output from Eval is a linear combination of the monomials $1, I_1, \ldots, I_k, V, K_1, \ldots, K_k, T_1, T_2, \ldots$. Now, it is convenient to slightly rethink the check of whether $V\mathsf{p_a} = \mathsf{p_b}$ within $\mathsf{O_v}$ with this notation. First off, we observe that if either of the polynomial contains a monomial of the form T_i, the check fails. In fact, it is immediately clear that the check can only possibly succeed is if $\mathsf{p_a}$ is a linear combination of 1 and the I_j's and $\mathsf{p_b}$ is a linear combination of

Adversary $\mathcal{D}$:

1 : **foreach** $\mathsf{p} \in \mathbb{Z}_p[I_1, \cdots, I_k, V, K_1, \cdots, K_k, T_1, \cdots, T_{2q}]$ **do** $\Lambda(\mathsf{p}) \leftarrow \bot$
2 : $\Lambda(1) \leftarrow\$\, \mathcal{L}$; $\Lambda(V) \leftarrow\$\, \overline{R(\Lambda)}$
3 : **foreach** $j \in [k]$ **do**
4 : $\quad \Lambda(I_j) \leftarrow\$\, \overline{R(\Lambda)}$; $\Lambda(K_j) \leftarrow\$\, \overline{R(\Lambda)}$
5 : $\mathsf{new} \leftarrow 0$; $\mathcal{X} \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$; $\mathcal{Y}_1 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$
6 : $\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(\cdot,\cdot,1),\mathsf{O_v}(\cdot,\cdot,1)}(\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k))$
7 : $\mathcal{Y}_2 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(K_1), \cdots, \Lambda(K_k)\}$; $\mathcal{Z} \leftarrow \emptyset$; $\mathcal{Z}' \leftarrow \emptyset$; $S \leftarrow \emptyset$
8 : $s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(\cdot,\cdot,2),\mathsf{O_v}(\cdot,\cdot,2)}(\phi, \Lambda(1), \Lambda(V), \Lambda(K_1), \cdots, \Lambda(K_k))$
9 : $\mathsf{win}' \leftarrow (\{s_1, \cdots, s_l\} = \{s'_1, \cdots, s'_l\}) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$
10 : **if** $\mathsf{win}' = \mathsf{true}$ **then**
11 : $\quad$ **foreach** $i, j \in [k]$ **do if** $s_i = s'_j$ **then** $\pi(i) = j$
12 : $\quad$ **return** π
13 : **else return** $\bot$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\Lambda^{-1}(\mathbf{a}) = \bot$ **then**
2 : $\quad \mathsf{new} \leftarrow \mathsf{new} + 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$
3 : **if** $\Lambda^{-1}(\mathbf{b}) = \bot$ **then**
4 : $\quad \mathsf{new} \leftarrow \mathsf{new} + 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$
5 : $\mathsf{p} \leftarrow \mathsf{p_a} + \mathsf{p_b}$
6 : **if** $\Lambda(\mathsf{p}) = \bot$ **then** $\Lambda(\mathsf{p}) \leftarrow\$\, \overline{R(\Lambda)}$
7 : $\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}), \mathsf{from})$
8 : **return** $\Lambda(\mathsf{p})$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\Lambda^{-1}(\mathbf{a}) = \bot$ **then**
2 : $\quad \mathsf{new} \leftarrow \mathsf{new} + 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$
3 : **if** $\Lambda^{-1}(\mathbf{b}) = \bot$ **then**
4 : $\quad \mathsf{new} \leftarrow \mathsf{new} + 1$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$
5 : $\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$
6 : $\mathsf{PolyMultCheck}(\mathsf{p_a}, \mathsf{p_b})$

Procedure $\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathsf{from})$:

1 : **if** $\mathsf{from} = 1$ **then**
2 : $\quad$ **if** $\mathbf{c} \notin \mathcal{Y}_1$ **then** $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$
3 : $\quad \mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$
4 : **if** $\mathsf{from} = 2$ **then**
5 : $\quad$ **if** $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$
6 : $\quad\quad$ **if** $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$
7 : $\quad\quad \mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

Procedure $\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

1 : **if** $\mathsf{from} = 1$ **then** $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$
2 : **if** $\mathsf{from} = 2$ **then**
3 : $\quad$ **if** $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$
4 : $\quad$ **if** $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$
5 : $\quad \mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

Fig. 16. Adversary $\mathcal{D}$ which plays the permutation game $\mathbb{PG}$. The changes in $\mathcal{D}$ compared to $\mathbb{G}_{11}$ have been highlighted.

V and the K_j's. Now, assume that

$$\mathsf{p_a}(I_1, \ldots, I_k) = a_0 + \sum_{j=1}^{k} \vec{x}[j] \cdot I_j \ ,$$

$$\mathsf{p_b}(V, K_1, \ldots K_k) = b_0 \cdot V + \sum_{j=1}^{k} \vec{y}[j] \cdot K_j \ .$$

Then, $V \cdot \mathsf{p}_a = \mathsf{p}_b$ if and only if $a_0 = b_0$ *and* $\vec{y}[j] = \vec{x}[\pi(j)]$ for all $j \in [k]$. If we are now in Game $\mathbb{PG}$, and π is the chosen permutation, then this is equivalent to $O(\vec{x}, \vec{y}) = \mathsf{true}$ and $a_0 = b_0$.

This leads naturally to the adversary $\mathcal{D}$, which we formally describe in Figure 16. The adversary will simply sample labels $\mathbf{f}_1, \ldots, \mathbf{f}_k$ for $\sigma(v \cdot i_{\pi(1)}), \ldots, \sigma(v \cdot i_{\pi(k)})$, and associate with them polynomials in the variables $K_1, \ldots, K_j$. Other than that, it simulates the game $\mathbb{G}_{11}$, with the exception that the check $V \cdot \mathsf{p_a} = \mathsf{p_b}$ is not implemented using the above approach – summarized in Figure 15. Note that $\mathcal{D}$ aborts when $|S| = l$

and makes at most q queries to O. Thus $\mathcal{D}$ is a (q, l)-query adversary against $\mathbb{PG}$. If $\mathcal{D}$ does not abort, then its simulation of $\mathbb{G}_{11}$ is perfect. If $\mathbb{G}_{11} \Rightarrow \mathsf{true}$ and $\mathcal{D}$ does not abort, then win' shall be true and $\mathcal{D}$ will output the correct π.

The rest of the proof will now require proving that whenever $\mathbb{G}_{11}$ outputs true our adversary $\mathcal{D}$ will never abort due to the check $|S| = l$. Since $\mathbb{G}_{11} \Rightarrow \mathsf{true}$ only if $|\mathcal{Z}| < l$, the following lemma implies that $\mathcal{D}$ does not abort if $\mathbb{G}_{11} \Rightarrow \mathsf{true}$.

Lemma 8. *Let $(\vec{x}_1, \vec{y}_1), \cdots, (\vec{x}_u, \vec{y}_u)$ be the queries made by $\mathcal{D}$ to O which return true. Then,*

$$\mathsf{rank}(\vec{x}_1, \cdots, \vec{x}_u) \leqslant |\mathcal{Z}| \ .$$

Proof. We introduce an operator $\mathsf{proj}(\mathsf{p})$ which maps a polynomial to a vector in $\mathbb{Z}_p^k$ consisting of the coefficients of the monomials $I_1, \ldots, I_k$. We will show below that for all $i \in [u]$, there exists a set $\mathcal{Z}_i \subseteq \mathcal{Z}$ such that $\vec{x}_i \in \mathsf{span}(\mathsf{proj}(\Lambda^{-1}(\mathcal{Z}_i)))$. From this, the claim is immediate, because

$$\begin{aligned}\mathsf{rank}(\vec{x}_1, \cdots, \vec{x}_u) &\leqslant \mathsf{rank}\left(\bigcup_{i=1}^{u} \mathsf{proj}(\Lambda^{-1}(\mathcal{Z}_i))\right) \\ &\leqslant \left|\bigcup_{i=1}^{u} \mathsf{proj}(\Lambda^{-1}(\mathcal{Z}_i))\right| \leqslant \left|\bigcup_{i=1}^{u} \mathcal{Z}_i\right| \leqslant |\mathcal{Z}| \ .\end{aligned}$$

First off, note that every non-zero query $(\vec{x}_i, \vec{y}_i)$ that returns true is made during the simulation of the execution of $\mathcal{R}_2$. This is because such a query requires the polynomial associated with $\vec{y}_i$ to contain a monomial in K_j for at least some $j \in [k]$, and such polynomials can only occur in the execution of $\mathcal{R}_2$. Also note that every vector $\vec{x}_i$ is associated with a polynomial p_i. This polynomial *must* have form $a_0 + \sum_{j=1}^{k} \vec{x}_i[j] \cdot I_j$, as discussed before.

Let us look at an execution of $\mathcal{D}$ after-the-fact, and more specifically, the simulated execution of $\mathcal{R}_1, \mathcal{R}_2$ within $\mathcal{D}$. Let $\mathbf{a}_i$ be the label associated with the polynomial p_i. If it has not been output by an Eval query ever, then it must be that $\mathbf{a}_i = \Lambda(I_j)$ for some $j \in [k]$. Because $\mathbf{a}_i \in \mathcal{X}$, then $\mathbf{a}_i \in \mathcal{Z}$, and thus $\mathcal{Z}_i = \{\mathbf{a}_i\}$. Similarly, assume that $\mathbf{a}$ was output last by an Eval query of $\mathcal{R}_1$. Then, it must be that $\mathbf{a}_i$ has been added to $\mathcal{X}$, because every output of an Eval query by $\mathcal{R}_1$ is aded to $\mathcal{X}$, *unless* the string $\mathbf{a}_i$ was associated earlier with one of the T_z monomials. But this label would not lead the oracle query to output 1. Therefore, $\mathbf{a}_i$ is added to $\mathcal{Z}$ upon $\mathcal{R}_2$'s corresponding $\mathsf{O_v}$ query, and we can set $\mathcal{Z}_i = \{\mathbf{a}_i\}$.

We are left with the harder case that $\mathbf{a}_i$ is the output of an Eval query of $\mathcal{R}_2$. Note that because of this, it is not hard to see that we can write

$$\mathsf{p}_i = \sum_{j=1}^{r} \gamma_j \cdot \Lambda^{-1}(\mathbf{b}_j) \ ,$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_j$ are inputs to Eval queries made by $\mathcal{R}_2$. This in turn implies that

$$\vec{x}_i = \mathsf{proj}(\mathsf{p}_i) = \sum_{j=1}^{r} \gamma_j \cdot \mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \ .$$

We now claim that if $\mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \neq \vec{0}$, then it must be that $\mathbf{b}_j \in \mathcal{X}$, and therefore it is added to $\mathcal{Z}$. (Either within the Eval query, or in an earlier $\mathsf{O_v}$ query using it.)

Note that if $\mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \neq 0$, then $\mathbf{b}_j$ is not a fresh label input to an Eval query by either of $\mathcal{R}_1$ or $\mathcal{R}_2$ (as it would have been associated with T_z, for some z), and it is not one of the labels $\Lambda(1), \Lambda(V), \Lambda(K_1), \ldots, \Lambda(K_k)$ input to $\mathcal{R}_2$. So, $\mathbf{b}_j$ is either $\Lambda(I_1), \ldots, \Lambda(I_k)$, *or* the output of an Eval query by $\mathcal{R}_1$ which is added to $\mathcal{X}$. In both cases, $\mathbf{b}_j \in \mathcal{X}$. Hence, we can now define

$$\mathcal{Z}_i = \left\{\mathbf{b}_j \ : \ j \in [k] \ , \ \ \mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \neq \vec{0}\right\} \ .$$

This concludes the proof. □

We have established that if $\mathbb{G}_{11}$ outputs true, then $\mathcal{D}$ will not abort and hence $\mathcal{D}$ simulates $\mathbb{G}_{11}$ to $\mathcal{R}_1, \mathcal{R}_2$ perfectly. If $\mathsf{win} = \mathsf{true}$ in $\mathbb{G}_{11}$, the checks by $\mathcal{D}$ succeed and $\mathcal{D}$ outputs the correct permutation and wins $\mathbb{PG}$. Therefore, $\mathcal{D}$ is a (q, l)-query adversary such that $\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}$ if $\mathbb{G}_{11} \Rightarrow \mathsf{true}$. Hence,

$$\Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}\right] . \tag{17}$$

Combining Lemma 7 and (9),(17) we get,

$$\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}\right] + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} . \tag{18}$$

□

Combining (8) and (18) we get,

$$\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} .$$

□

4.4 Memory Lower Bound when $|\mathcal{Z}| \geqslant l$ (Proof of Lemma 4)

Recall that we need to prove the following lemma, which we do by using a compression argument.

Lemma 4. *If the size of the state ϕ output by $\mathcal{R}_1$ is s bits and $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries in total in the game $\mathbb{G}_3$ as defined in Figure 5, then*

$$\Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1 + \frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{k^2+k+2}{p} .$$

Proof. Our proof does initial game hopping, with easy transitions. It first introduces a new game, $\mathbb{G}_{12}$ whose minor difference from game $\mathbb{G}_3$ is that it samples $i_1, \cdots, i_k, v$ using $\mathsf{RestrictedSample}$ which was previously used in game $\mathbb{G}_{11}$. It adds a bad flag while sampling $i_1, \cdots, i_k, v$ which is set to true if v is in $\{0, 1\}$ or if $|1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v| < 2k+2$. The bad event does not affect the output of $\mathbb{G}_{12}$ in any way. Observe that even though the sampling of $i_1, \cdots, i_k, v$ is written in a different manner in $\mathbb{G}_{12}$, it is identical to that in $\mathbb{G}_3$. In all other respects these two games are identical.

$$\Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_{12} \Rightarrow \mathsf{true}\right] . \tag{19}$$

Games $\mathbb{G}_{12}, \mathbb{G}_{13}$ differ in the procedure $\mathsf{RestrictedSample}$ and the condition to return true. Note that the conditions of bad being set to true is identical in $\mathbb{G}_{12}, \mathbb{G}_{13}$ and given that bad is not set to true, $\mathbb{G}_{13}$ returns true whenever $\mathbb{G}_{12}$ returns true. Therefore, using the Fundamental Lemma of Game Playing

$$\Pr\left[\mathbb{G}_{12} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{13} \Rightarrow \mathsf{true}\right] + \Pr\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathbb{G}_{13}\right] .$$

From our analysis in the proof of Lemma 7, we have established that the probability of bad being set to true in $\mathsf{RestrictedSample}$ is at most $\frac{k^2+k+2}{p}$. Since in $\mathbb{G}_{13}$ bad is set only in $\mathsf{RestrictedSample}$, the probability of bad being set to true is the same. Hence, we get,

$$\Pr\left[\mathbb{G}_{12} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{13} \Rightarrow \mathsf{true}\right] + \frac{k^2+k+2}{p} . \tag{20}$$

The compression argument. We assume $\Pr\left[\mathbb{G}_{13} \Rightarrow \mathsf{true}\right] = 2\epsilon$. We say a σ is "good" in $\mathbb{G}_{13}$ if

$$\Pr\left[\mathbb{G}_{13} \Rightarrow \mathsf{true} \,\middle|\, \sigma \text{ was sampled in } \mathbb{G}_{13}\right] \geqslant \epsilon . \tag{21}$$

It follows from Markov's inequality that at least ϵ fraction of σ's are "good". The following lemma captures the essence of our compression argument.

Game $\boxed{\mathbb{G}_{12}}, \boxed{\mathbb{G}_{13}}$:

$\sigma \leftarrow\$ \mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L}); i_1, \cdots, i_k, v \leftarrow \mathsf{RestrictedSample}()$

$\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(\cdot,\cdot,1), \mathsf{O_v}(\cdot,\cdot,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

$\pi \leftarrow\$ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$

$s_1', s_2', \cdots, s_k' \leftarrow \mathcal{R}_2^{\mathsf{Eval}(\cdot,\cdot,2), \mathsf{O_v}(\cdot,\cdot,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s_j') \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s_j' \neq s_l')$

return $(\boxed{\mathsf{win} \wedge} |\mathcal{Z}| \geqslant l)$

Procedure $\mathsf{RestrictedSample}()$:

$v \leftarrow\$ \mathbb{Z}_p$; **if** $v \in \{0, 1\}$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{v \leftarrow\$ \mathbb{Z}_p \backslash \{0, 1\}}$

$\mathcal{S} \leftarrow \{1\}; \mathcal{S}' \leftarrow \{v^{-1}\}$

foreach $j \in [k]$ **do**

  $i_j \leftarrow\$ \mathbb{Z}_p$; **if** $i_j \in \mathcal{S} \cup \mathcal{S}'$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{i_j \leftarrow\$ \mathbb{Z}_p \backslash (\mathcal{S} \cup \mathcal{S}')}$

  $\mathcal{S} \xleftarrow{\cup} \{i_j\}; \mathcal{S}' \xleftarrow{\cup} \{v^{-1} \cdot i_j\}$

return $i_1, \cdots, i_k, v$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

$\mathbf{c} \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$

if $\mathsf{from} = 1$ **then**

  if $\mathbf{c} \notin \mathcal{Y}_1$ **then** $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$

  $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

if $\mathsf{from} = 2$ **then**

  if $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  if $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

return $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

if $\mathsf{from} = 1$ **then** $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

if $\mathsf{from} = 2$ **then**

  if $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  if $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ **then** $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

return $(v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b}))$

Fig. 17. Games $\mathbb{G}_{12}, \mathbb{G}_{13}$. The statement within the thinner box is present only in $\mathbb{G}_{12}$ and the statement within the thicker box is present only in $\mathbb{G}_{13}$. The newly introduced changes compared to $\mathbb{G}_3$ are highlighted.

Lemma 9. *If the state output by $\mathcal{R}_1$ in $\mathbb{G}_{13}$ has size s bits, all the "good" σ's can be encoded such that the size of the encoding space is at most*

$$2^s p! \left(1 + \frac{6q}{p}\right)^{(2q-l)} \left(\frac{p}{8q^2(2k+2+3q)}\right)^{-l},$$

and decoded correctly with probability at least ϵ.

We first give some intuition regarding how we achieve compression and show how Lemma 9 leads to an upper bound on $\Pr[\mathbb{G}_3 \Rightarrow 1]$. We defer the proof of Lemma 9 to Section 4.5.

Intuition regarding compression. Observe in $\mathbb{G}_{13}$, the labels in $\mathcal{Z}$ were queried by $\mathcal{R}_2$ (these labels were not seen by $\mathcal{R}_2$ before they were queried) and were answers to $\mathcal{R}_1$ and were not seen by $\mathcal{R}_1$ before the query. The core idea is that for all $\mathbf{a} \in \mathcal{L} \backslash \mathcal{Z}$, we store exactly one of $\mathbf{a}$ or its pre-image in the encoding and for all labels in $\mathcal{Z}$, we store neither the label nor its pre-image. Since $\mathcal{R}_2$ queries all the labels in $\mathcal{Z}$, these labels can be found by running $\mathcal{R}_2$ while decoding. Since all the labels in $\mathcal{Z}$ are answers to queries of $\mathcal{R}_1$ and were not seen by $\mathcal{R}_1$ before the query, their pre-images can be figured out while running $\mathcal{R}_1$.

High level outlines of Encode, Decode. In Encode, we simulate the steps of $\mathbb{G}_{13}$ to $\mathcal{R}_1, \mathcal{R}_2$, including bookkeeping and then run $\mathcal{R}_1$ again assuming the particular σ we are compressing is sampled in $\mathbb{G}_{13}$. In Decode, we run $\mathcal{R}_2$ and then $\mathcal{R}_1$ to recover σ. We treat the values $i_1, \cdots, i_k, v, \pi$ as part of the common

randomness provided to $\mathsf{Encode}, \mathsf{Decode}$ (we assume they are sampled from the same distribution they are sampled from in $\mathbb{G}_{13}$). The random tapes of $\mathcal{R}_1, \mathcal{R}_2$ can also be derived from the common randomness of $\mathsf{Encode}, \mathsf{Decode}$. For simplicity, we do not specify this explicitly in the algorithms and treat $\mathcal{R}_1, \mathcal{R}_2$ as deterministic.

Running $\mathcal{R}_2$. First off, we assume that $\mathcal{R}_1$ queries labels that it has "seen" before and $\mathcal{R}_2$ queries labels that $\mathcal{R}_1$ has "seen" or it has "seen" before. We shall relax this assumption later. Ideally, we would want to just store only ϕ, the inputs labels to $\mathcal{R}_2$ and the labels that are answers to $\mathcal{R}_2$'s queries. We append the input labels of $\mathcal{R}_2$ and labels that are answers to its Eval queries that it has not "seen" before to a list named Lbls. However, it is easy to see that this information is not enough to answer $\mathsf{O_v}$ queries during decoding, as answering $\mathsf{O_v}$ queries inherently requires knowledge about pre-images of $\mathcal{R}_2$. This naturally leads to the idea of maintaining a mapping of all the labels "seen by" $\mathcal{R}_2$ to their pre-images.

The mapping T of labels to pre-image expressions. The pre-images of input labels and the labels that were results of sequence of Eval queries on its input labels by $\mathcal{R}_2$, are known. However, $\mathcal{R}_2$ might query labels which were neither an input to it nor an answer to one of its Eval queries. Such a label is in $\mathcal{Z}$ since we have assumed that all labels queried by $\mathcal{R}_2$ were "seen by" $\mathcal{R}_1$ or "seen by" $\mathcal{R}_2$ before. We represent the pre-images of labels in $\mathcal{Z}$ using a placeholder variable X_n where n is incremented for every such label. Note that the pre-image of every label seen by $\mathcal{R}_2$ can be expressed as a linear polynomial in the X_n's (these linear polynomials are referred to as pre-image expressions from hereon). Therefore we maintain a mapping of all labels "seen by" and their pre-image expressions in a list of tuples named T. Our approach is inspired by a similar technique used by Corrigan-Gibbs and Kogan in [5]. Like in [5], we *stress* that the mapping T is not a part of the encoding.

For Eval queries, we can check if there is a tuple in T whose pre-image expression is the sum of the pre-image expressions of the input labels. If that is the case, we return the label of such a tuple. Otherwise, we append the answer label to Lbls. For $\mathsf{O_v}$ queries, we can return true if the pre-image expression of the first input label multiplied by v gives the pre-image expression of the second input label. Otherwise we return false.

Surprises. There is a caveat, however. There might arise a situation that the label which is the answer to the Eval query is present in T but its pre-image expression is not the sum of the pre-image expressions of the input labels. We call such a situation a "surprise" and we call the answer label in that case a "surprise label". For $\mathsf{O_v}$ queries, there might be a surprise when the answer of the $\mathsf{O_v}$ query is true but the pre-image expression of the first input label multiplied by v is different pre-image expression of the second input label. In this case we call the second input label the surprise label. We assign a sequence number to each query made by $\mathcal{R}_2$, starting from 1 and an index to each tuple in T, with the indices being assigned to tuples in the order they were appended to T. To detect the query where the surprise happens, we maintain a set named Srps_1 that contains tuples of query sequence numbers and indices of the surprise label in T. This set Srps_1 is a part of the encoding. Note that whenever there is a surprise, it means that two different pre-image expressions evaluate to the same value. Since these two pre-image expressions are linear polynomials, at least one variable can be eliminated from T by equating the two pre-image expressions.

Running $\mathcal{R}_1$. Now that we have enough information in the encoding to run $\mathcal{R}_2$, we consider the information we need to add to the encoding to run $\mathcal{R}_1$ after $\mathcal{R}_2$ is run. First, we need to provide $\mathcal{R}_1$ its input labels. Our initial attempt would be to append the input labels of $\mathcal{R}_1$ (except $\sigma(1), \sigma(v)$, which are already present) to Lbls. However, some of these input labels to $\mathcal{R}_1$ might have already been "seen by" $\mathcal{R}_2$. Since all labels "seen by" $\mathcal{R}_2$ are in T, we need a way to figure out which of $\sigma(i_j)$'s are in T. Note that such a label was either queried by $\mathcal{R}_2$ or an answer to a query of $\mathcal{R}_2$ (cannot have been an input to $\mathcal{R}_2$ given the restrictions on $i_1, \cdots, i_k, v$). Suppose q was the sequence number of the query in which $\sigma(i_j)$ was queried or an answer. The tuple (q, b, j) is added to the set Inputs where b can take values $\{1, 2, 3\}$ depending on whether $\sigma(i_j)$ was the first input label, the second input label or the answer label respectively. This set Inputs is a part of the encoding. The rest of the labels $\sigma(i_j)$, which do not appear in T, are added to T with their pre-images and the labels are appended to Lbls. Note that for all queries of $\mathcal{R}_1$, it follows from our assumption that the input labels will be in T. For every surprise, we add a tuple of sequence number and an index in T to the set Srps_2.

Relaxing the assumption. When we allow $\mathcal{R}_2$ to query labels it has not seen before or $\mathcal{R}_1$ has not seen, there are two issues. First, we need to add a tuple for the label in T (since T, by definition contains a tuple for all labels queried by $\mathcal{R}_2$). We solve this issue by adding the tuple made of the label and its pre-image. We have no hope of recovering the pre-image later, hence, we append the pre-image to a list named Vals. This list needs to be a part of the encoding since the pre-image of the label needs to be figured out to be added to T during decoding. For queries of $\mathcal{R}_1$, if the input label is not present in T, we do the same thing. The second issue that comes up when we relax the assumption is that we need to distinguish whether an input label was in $\mathcal{Z}$ or not. We solve this issue by maintaining a set of tuples named Free. For all labels in $\mathcal{Z}$ that are not an input label to $\mathcal{R}_1$, we add the tuple consisting of the sequence number of the query of $\mathcal{R}_2$ and b to Free where b set to 1 indicates it was the first input label and b set to 2 indicates it was the second input label.

The final steps. The labels the are absent in T are appended to a list named RLbls. If $|\mathcal{Z}| < l$, a fixed encoding D (the output of Encode for some fixed σ when $|\mathcal{Z}| \geqslant l$) is returned. Otherwise the encoding of σ consisting of Lbls, RLbls, Vals, Inputs, Srps_1, Srps_2, Free, ϕ is returned.

Wrapping up. The set of all "good" σ's has size at least $\epsilon p!$ (where we have used that the total number of injective functions from $\mathbb{Z}_p \to \mathcal{L}$ is $p!$). Using $\mathcal{X}$ to be the set of the "good" σ's, $\mathcal{Y}$ to be the set of encodings, $\mathcal{R}$ to be the set of cartesian product of the domains of $i_1, \cdots, i_k, v, \pi$, the set of all random tapes of $\mathcal{R}_1$ the set of all random tapes of $\mathcal{R}_2$ and $\mathcal{L}$, it follows from Lemma 9 and Proposition 1 that

$$\log\left(\Pr\left[\text{Decoding is correct}\right]\right) \leqslant s + (2q-l)\log\left(1+\frac{6q}{p}\right) - l\log\left(\frac{p}{8q^2(2k+2+3q)}\right) - \log\epsilon\,.$$

We have from Lemma 9 that $\Pr\left[\text{Decoding is correct}\right] \geqslant \epsilon$. Therefore,

$$2\log\epsilon \leqslant s + (2q-l)\log\left(1+\frac{6q}{p}\right) - l\log\left(\frac{p}{8q^2(2k+2+3q)}\right)\,.$$

Since $\Pr\left[\mathbb{G}_{13} \Rightarrow \mathsf{true}\right] = 2\epsilon$, using (19) and (20) we have,

$$\Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] \leqslant 2\cdot 2^{\frac{s}{2}}\left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}}\left(1+\frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{k^2+k+2}{p}\,.$$

□

4.5 Proof of Lemma 9

We provided the intuition behind our compression of σ in Section 4.4. We formally define the $\mathsf{Encode}, \mathsf{Decode}$ algorithms here and then analyze the size of the encoding space for "good" σ's.

The Encode procedure. As we have previously mentioned Encode simulates $\mathbb{G}_{13}$ to $\mathcal{R}_1, \mathcal{R}_2$ and then runs $\mathcal{R}_1$ again. The Encode procedure has been formally defined in Figure 18. We shall next explain the pseudocode in detail and relate it to the intuition we provided in Section 4.4. In order to understand the pseudocode of Encode, initially ignore the code involving $\mathsf{T}, \mathsf{Lbls}, \mathsf{Vals}, \mathsf{RLbls}, \mathsf{Free}, \mathsf{Srps}_1, \mathsf{Srps}_2$.

First off, note that the oracles $\mathsf{Eval}(.,.,1), \mathsf{Eval}(.,.,2), \mathsf{Eval}(.,.,3)$ have identical outputs to the oracle $\mathsf{Eval}(.,.)$ in $\mathbb{G}_{13}$ and the oracles $\mathsf{O_v}(.,.,1), \mathsf{O_v}(.,.,2), \mathsf{O_v}(.,.,3)$ have identical outputs to the oracle $\mathsf{O_v}(.,.)$ in $\mathbb{G}_{13}$. As mentioned previously the randomness input to Encode i.e. $i_1, \cdots, i_v, v, \pi$ are distributed according to the distribution of these values in $\mathbb{G}_{13}$. Observe that till line 7, Encode perfectly simulates $\mathbb{G}_{13}$ to $\mathcal{R}_1, \mathcal{R}_2$. Then in line 10, $\mathcal{R}_1$ is run again. Recall that we treat $\mathcal{R}_1, \mathcal{R}_2$ as deterministic without loss of generality (alternatively we could have specified the random tape of these adversaries as an input to $\mathsf{Encode}, \mathsf{Decode}$). Therefore, $\mathcal{R}_1$ shall make the same queries when it is run the second time as it did in its first run since it receives identical answers to its queries (we have previously argued that the outputs of oracles $\mathsf{Eval}(.,.,1), \mathsf{Eval}(.,.,3)$ are identical and the outputs of oracles $\mathsf{O_v}(.,.,1), \mathsf{O_v}(.,.,3)$ are identical on same inputs).

Fig. 18. The Encode procedure. If $|\mathcal{Z}| < l$, a fixed encoding D (the output of Encode for some fixed σ when $|\mathcal{Z}| \geqslant l$) is returned. The function T.append(tuple) appends tuple into T. The first inserted tuple has index 1 and the index increases by 1 for every subsequent insertion. T[i] returns the tuple at index i in T. The function T(**a**) returns pre-image expression A such that $(\mathbf{a}, A, *) \in \mathsf{T}$. The function VarReduce(T, C, C') equates expressions C and C', expresses the variable with highest subscript in the equation (say X_j) in terms of other variables, and substitutes X_j throughout T.

Now that we have explained the high-level structure of Encode and related it to the intuition provided before, we shall delve into the details of its book-keeping. Recall that we say that a label has been seen by

Procedure $\mathsf{Decode}((\mathsf{Lbls}, \mathsf{RLbls}, \mathsf{Vals}, \mathsf{Srps}_1, \mathsf{Srps}_2, \mathsf{Inputs}, \mathsf{Free}, \phi), (i_1, \cdots, i_k, v, \pi))$:

1 : **foreach** $j \in \mathbb{Z}_p$ **do** $\sigma(j) \leftarrow \perp; z_1 \leftarrow 1; z_2 \leftarrow 1; z_3 \leftarrow 1; \mathsf{T} \leftarrow []$
2 : $\sigma(1) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1; \mathsf{T}.\mathsf{append}((\sigma(1), 1, 1))$
3 : $\sigma(v) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1; \mathsf{T}.\mathsf{append}((\sigma(v), v, 1))$
4 : **foreach** $j \in [k]$ **do**
5 : $\quad \sigma(i_j \cdot v) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1; \mathsf{T}.\mathsf{append}((\sigma(i_j \cdot v), i_j \cdot v, 1))$
6 : $n \leftarrow 0; c_1 \leftarrow 0; c_2 \leftarrow 0$
7 : $s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$
8 : **foreach** $j \in [k]$ **do**
9 : $\quad$ **if** $\sigma(i_j) = \perp$ **then**
10 : $\quad\quad \sigma(i_j) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1$
11 : $\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,3), \mathsf{O_v}(.,.,3)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$
12 : **foreach** $j \in [|\mathsf{T}|]$ **do**
13 : $\quad (\mathbf{s}, m, f) \leftarrow \mathsf{T}[j]$
14 : $\quad \sigma(m) \leftarrow \mathbf{s}$
15 : **foreach** $j \in [p]$ **do**
16 : $\quad$ **if** $\sigma(j) = \perp$ **then**
17 : $\quad \sigma(j) \leftarrow \mathsf{RLbls}[z_3]; z_3 \leftarrow z_3 + 1$
18 : **return** σ

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 2)$:

1 : $c_1 \leftarrow c_1 + 1$
2 : **if** $(\mathbf{a}, *, *) \notin \mathsf{T}$ **then**
3 : $\quad \mathsf{AddToTable}(\mathbf{a}, c_1, 1)$
4 : **if** $(\mathbf{b}, *, *) \notin \mathsf{T}$ **then**
5 : $\quad \mathsf{AddToTable}(\mathbf{b}, c_1, 2)$
6 : **if** $\exists \mathbf{c}' : (\mathbf{c}', v\mathsf{T}(\mathbf{a}), *) \in \mathsf{T}$ **then**
7 : $\quad \mathbf{c} \leftarrow \mathbf{c}'$
8 : **elseif** $\exists i : (c_1, i) \in \mathsf{Srps}_1$ **then**
9 : $\quad (\mathbf{c}, \mathsf{T}(\mathbf{c}), f) \leftarrow \mathsf{T}[i]$
10 : $\quad \mathsf{VarReduce}(\mathsf{T}, v\mathsf{T}(\mathbf{a}), \mathsf{T}(\mathbf{c}))$
11 : **else return** false
12 : **return** $(\mathbf{b} = \mathbf{c})$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 2)$:

1 : $c_1 \leftarrow c_1 + 1$
2 : **if** $(\mathbf{a}, *, *) \notin \mathsf{T}$ **then**
3 : $\quad \mathsf{AddToTable}(\mathbf{a}, c_1, 1)$
4 : **if** $(\mathbf{b}, *, *) \notin \mathsf{T}$ **then**
5 : $\quad \mathsf{AddToTable}(\mathbf{b}, c_1, 2)$
6 : **if** $\exists \mathbf{c}' : (\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ **then**
7 : $\quad \mathbf{c} \leftarrow \mathbf{c}'$
8 : **elseif** $\exists i : (c_1, i) \in \mathsf{Srps}_1$ **then**
9 : $\quad (\mathbf{c}, \mathsf{T}(\mathbf{c}), f) \leftarrow \mathsf{T}[i]$
10 : $\quad \mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), \mathsf{T}(\mathbf{c}))$
11 : **else**
12 : $\quad \mathbf{c} \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1$
13 : $\quad \mathsf{T}.\mathsf{append}(\mathbf{c}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), 1)$
14 : $\quad$ **if** $\exists j \in [k] : (c_1, 3, j) \in \mathsf{Inputs}$ **then**
15 : $\quad\quad \sigma(i_j) \leftarrow \mathbf{c}$
16 : $\quad\quad \mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), i_j)$
17 : $\quad$ **return** $\mathbf{c}$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 3)$:

1 : $c_2 \leftarrow c_2 + 1$
2 : **if** $(\mathbf{a}, *, *) \notin \mathsf{T}$ **then**
3 : $\quad \mathsf{T}.\mathsf{append}(\mathbf{a}, \mathsf{Vals}[z_2], 2); z_2 \leftarrow z_2 + 1$
4 : **if** $(\mathbf{b}, *, *) \notin \mathsf{T}$ **then**
5 : $\quad \mathsf{T}.\mathsf{append}(\mathbf{a}, \mathsf{Vals}[z_2], 2); z_2 \leftarrow z_2 + 1$
6 : **if** $\exists \mathbf{c}' : (\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ **then**
7 : $\quad \mathbf{c} \leftarrow \mathbf{c}'$
8 : **elseif** $\exists i : (c_2, i) \in \mathsf{Srps}_2$ **then**
9 : $\quad (\mathbf{c}, \mathsf{T}(\mathbf{c}), f) \leftarrow \mathsf{T}[i]$
10 : $\quad \mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), \mathsf{T}(\mathbf{c}))$
11 : **else**
12 : $\quad \mathbf{c} \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1$
13 : $\quad \mathsf{T}.\mathsf{append}(\mathbf{c}, C)$
14 : **return** $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 3)$:

1 : $c_2 \leftarrow c_2 + 1$
2 : **if** $(\mathbf{a}, *, *) \notin \mathsf{T}$ **then**
3 : $\quad \mathsf{T}.\mathsf{append}(\mathbf{a}, \mathsf{Vals}[z_2], 2); z_2 \leftarrow z_2 + 1$
4 : **if** $(\mathbf{b}, *, *) \notin \mathsf{T}$ **then**
5 : $\quad \mathsf{T}.\mathsf{append}(\mathbf{a}, \mathsf{Vals}[z_2], 2); z_2 \leftarrow z_2 + 1$
6 : **if** $\exists \mathbf{c}' : (\mathbf{c}', v\mathsf{T}(\mathbf{a}), *) \in \mathsf{T}$ **then**
7 : $\quad \mathbf{c} \leftarrow \mathbf{c}'$
8 : **elseif** $\exists i : (c_2, i) \in \mathsf{Srps}_2$ **then**
9 : $\quad (\mathbf{c}, \mathsf{T}(\mathbf{c}), f) \leftarrow \mathsf{T}[i]$
10 : $\quad \mathsf{VarReduce}(\mathsf{T}, v\mathsf{T}(\mathbf{a}), \mathsf{T}(\mathbf{c}))$
11 : **else return** false
12 : **return** $(\mathbf{b} = \mathbf{c})$

Procedure $\mathsf{AddToTable}(\boldsymbol{\ell}, c, b)$:

1 : **if** $\exists j \in [k] : (c, b, j) \in \mathsf{Inputs}$ **then** $\sigma(i_j) \leftarrow \boldsymbol{\ell}; \mathsf{T}.\mathsf{append}(\boldsymbol{\ell}, i_j, 0)$
2 : **if** $(c, b) \in \mathsf{Free}$ **then** $n \leftarrow n + 1; \mathsf{T}.\mathsf{append}(\boldsymbol{\ell}, X_n, 0)$
3 : **if** $(\boldsymbol{\ell}, *, *) \notin \mathsf{T}$ **then** $\mathsf{T}.\mathsf{append}(\boldsymbol{\ell}, \mathsf{Vals}[z_2], 2); z_2 \leftarrow z_2 + 1$

Fig. 19. The Decode function. The function $\mathsf{T}.\mathsf{append}(\mathsf{tuple})$ appends tuple into T. The first inserted tuple has index 1 and the index increases by 1 for every subsequent insertion. $\mathsf{T}[i]$ returns the tuple at index i in T. The function $\mathsf{T}.\mathsf{index}(\mathbf{a})$ returns the index of the tuple of the form $(\mathbf{a}, *, *)$ in T and the function $\mathsf{T}(\mathbf{a})$ returns pre-image expression A such that $(\mathbf{a}, A, *) \in \mathsf{T}$. The function $\mathsf{VarReduce}(\mathsf{T}, C, C')$ equates expressions C and C', expresses the variable with highest subscript in the equation (say X_j) in terms of other variables, and substitutes X_j throughout T.

$\mathcal{R}_1$ if it was an input to $\mathcal{R}_1$, queried by $\mathcal{R}_1$ or an answer to a previously made $\mathsf{Eval}(.,.,1)$ query and a label has been seen by $\mathcal{R}_2$ if it was an input to $\mathcal{R}_2$, queried by $\mathcal{R}_2$ or an answer to a previously made $\mathsf{Eval}(.,.,2)$ query. To begin with we note that the sets $\mathcal{Z}, \mathcal{X}, \mathcal{Y}_1, \mathcal{Y}_2$ are never updated after line 7 in Encode and hence are identical to what they would have been in $\mathbb{G}_{13}$ if the σ input to Encode was sampled.

The ordered first-in first-out (FIFO) list Lbls first contains the input labels of $\mathcal{R}_2$, then the labels that are answers to the Eval queries of $\mathcal{R}_2$ and had not been seen before by $\mathcal{R}_2$ (these labels are ordered in the order of the queries). Next, it contains labels that are input to $\mathcal{R}_1$ that were not queried by $\mathcal{R}_2$ during its run (ordered in the order of inputs of $\mathcal{R}_1$). Finally, it contains labels that are answers to the Eval queries of $\mathcal{R}_1$ during its second run that had not been seen by it before (during the second run) and are not among labels seen by $\mathcal{R}_2$ during its run (again ordered in the order of the queries).

The ordered (FIFO) list Vals first contains the pre-image of the labels (i.e. values in $\mathbb{Z}_p$) that were queried by $\mathcal{R}_2$ that it had not seen before (these pre-images are ordered in the order of the queries). It also contains the pre-image of the labels that queried by $\mathcal{R}_1$ during its second run that had not been seen by it before (during the second run) and are not among labels seen by $\mathcal{R}_2$ during its run (again ordered in the order of the queries).

The set Inputs is used to keep track of the labels that are inputs to $\mathcal{R}_1$ that $\mathcal{R}_2$ makes a query on or is an answer to $\mathcal{R}_2$'s Eval query. Each element of the set is a tuple that consists of the sequence number of the query, a flag indicating whether the label was the first or the second label of a query or the output of a query and the j such that the label is $\sigma(i_j)$.

The data structure T is an ordered list of tuples with each tuple consisting of a label, its pre-image expression (as defined in Section 4.4) and a flag which takes value $0, 1, 2$. The flag set to 1 indicates that the label in the tuple needs to be written to the list Lbls and the flag set to 2 indicates that the pre-image of the label in the tuple needs to be written to the list Vals. The tuples containing the input labels of $\mathcal{R}_2$ are first added to T with the pre-image of the labels being the pre-image expression. For every unique label seen by $\mathcal{R}_2$ a tuple containing the label is added to T when the label is queried or is an answer to a query. When $\mathcal{R}_2$ queries a label, there might already be a tuple containing the label- in that case, no new tuple containing the label is added to T. In case there was no tuple in T containing the queried label, the label might either have been **(a)** an input to $\mathcal{R}_1$ or **(b)** a label that was an answer to $\mathcal{R}_1$'s Eval query and had not been seen by $\mathcal{R}_1$ before the query or **(c)** neither of **(a)**, **(b)**. For case **(a)** and **(c)** the pre-image expression for the label is its actual pre-image. For case **(b)**, the pre-image expression of the label is a placeholder variable X_n (with n incremented every time for a new placeholder). In this case, a tuple that consists of the sequence number of the query, whether the label was the first or the second label of the query is added to the set Free. Other than the input labels and the labels queried by $\mathcal{R}_2$, the only labels seen by $\mathcal{R}_2$ are the answers to the Eval queries. Consider an Eval query by $\mathcal{R}_2$ on labels $\mathbf{a}, \mathbf{b}$. Let $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$. Suppose no tuple in T contains $\mathbf{c}$, in that case a tuple containing the label $\mathbf{c}$ and the pre-image expression the sum of the pre-image expressions of labels $\mathbf{a}$ and $\mathbf{b}$ is added to T. On the other hand if T contains $\mathbf{c}$ but its pre-image expression differs from the sum of the pre-image expressions of labels $\mathbf{a}$ and $\mathbf{b}$, then $\mathbf{c}$ is a surprise label as defined in Section 4.4. The two pre-image expressions are equated and the variable with highest subscript in the equation (say X_j) is expressed in terms of other variables, and then X_j is substituted throughout T (we name this the VarReduce procedure). A tuple that consists of the sequence number of the query, the index of the tuple containing the label $\mathbf{c}$ is added to the set Srps_1. As mentioned previously, a surprise label may arise even for a O_v query by $\mathcal{R}_2$. Suppose an O_v query was made on labels $\mathbf{a}, \mathbf{b}$. In this case $\mathbf{b}$ is a surprise label if there is a tuple containing $\mathbf{b}$ such that the pre-image expression is different from the pre-image expression of the $\mathbf{a}$ by v. Again, in this case in a similar fashion, one variable is substituted throughout T using the VarReduce procedure and a tuple that consists of the sequence number of the query, the index of the tuple containing the label $\mathbf{b}$ is added to the set Srps_1.

After $\mathcal{R}_2$ finishes running, tuples containing input labels of $\mathcal{R}_1$ that $\mathcal{R}_2$ did not see are added to T with the pre-image expression being the actual pre-images (this ensures that all labels input to $\mathcal{R}_1$ are in T). Next, when $\mathcal{R}_1$ is run, for every unique label seen by $\mathcal{R}_1$ by that is not contained in a tuple in T, a tuple containing the label is added to T in the way similar to while running $\mathcal{R}_2$ with the only major differences being **(a)** no placeholder variables are introduced in this run i.e. for input labels that were previously not in T, the pre-image expression is the pre-image of the label itself **(b)** surprises are kept track of in the set Srps_2 in this case.

After $\mathcal{R}_1$ is run for the second time, the sets Lbls, Vals are populated based on the entries in T. All the labels that are not in T are inserted into the list RLbls in lexicographical order of the pre-images. Finally if

$|\mathcal{Z}| \leqslant l$, the encoding D for some fixed σ for which $|\mathcal{Z}| \leqslant l$ is returned. Otherwise $\mathsf{Lbls}, \mathsf{RLbls}, \mathsf{Vals}, \mathsf{Srps}_1, \mathsf{Srps}_2, \mathsf{Inputs}, \mathsf{Free}$ along with ϕ, the state output by $\mathcal{R}_1$ is returned by Encode.

THE Decode PROCEDURE. The Decode procedure, as mentioned previously runs $\mathcal{R}_2$ and then $\mathcal{R}_1$. It starts running $\mathcal{R}_2$ using the state ϕ and the first $k+2$ labels in Lbls. It maintains the data structure T just like Encode. It inserts tuples containing the input labels of $\mathcal{R}_2$ into T with the pre-image expressions being the pre-images (which it gets from the input randomness). On an Eval or $\mathsf{O_v}$ query on labels $\mathbf{a}, \mathbf{b}$, it first checks if a tuple containing $\mathbf{a}$ is in T. If not it checks if a tuple containing the sequence number of the query, 1 and some $j \in [k]$ is present in Inputs- if that is the case, the pre-image expression of $\mathbf{a}$ is i_j. It then checks whether a tuple containing the sequence number of the query and 1 is present in Free- in that case the pre-image expression of $\mathbf{a}$ is a new placeholder variable X_n (n is incremented). Otherwise, it reads the next value of Vals and assigns it to the pre-image expression of $\mathbf{a}$. For the other input label $\mathbf{b}$, the same steps are followed.

In order to find the answer label for an Eval query, it first checks whether there is a tuple in Srps_1 with the first entry being the sequence number of the query and some i- if that is the case the label in the i^{th} tuple of T is returned as answer and the pre-image expression of the i^{th} tuple of T and the sum of the pre-image expressions of the two inputs are equated and one variable is substituted throughout T using the $\mathsf{VarReduce}$ procedure. Otherwise, it returns removes the top label from the list Lbls and returns it. It also adds a tuple containing the label to T. The $\mathsf{O_v}$ queries are answered in an analogous fashion. Also note that for every label queried by $\mathcal{R}_2$ and every label that is an answer to $\mathcal{R}_2$'s Eval query, the Decode procedure checks whether it is an input label to $\mathcal{R}_1$ using the Inputs set.

After running $\mathcal{R}_2$, some of the inputs to $\mathcal{R}_1$ might have been figured out using the set Inputs during $\mathcal{R}_2$'s query. The rest are obtained by removing the elements of the list Lbls and assigning them to unassigned inputs of $\mathcal{R}_1$. For each such label, a tuple containing the label is inserted to T. Then Decode runs $\mathcal{R}_1$ with the state ϕ and its input labels. The queries of $\mathcal{R}_1$ are answered in a similar fashion like the queries of $\mathcal{R}_2$.

After $\mathcal{R}_1$ has finished running, for every tuple in T, Decode assigns the labels to the pre-image expression (we shall argue that if the input to Decode was a valid encoding i.e. not D, then after the execution of $\mathcal{R}_1$ all the pre-image expressions in T are actual pre-images i.e. there are no more placeholders in T). Finally, the values in $\mathbb{Z}_p$ that have not been assigned images are assigned images using the labels in RLbls.

We say that the output of Encode is a valid encoding if the output was produced after the check $|\mathcal{Z}| \geqslant l$ succeeded in Encode. We claim that Decode produces the correct σ whenever it receives a valid encoding as input. The main observation here is that T is identically populated in $\mathsf{Encode}, \mathsf{Decode}$. From the pseudocode and description above of $\mathsf{Encode}, \mathsf{Decode}$, it is not very difficult to infer correctness. Nonetheless, we provide a proof sketch of correctness of decoding to Appendix B.

Next we shall relate the probability of Encode returns a valid encoding to the probability of $\mathcal{R}_1, \mathcal{R}_2$ winning $\mathbb{G}_{13}$. We have already argued that the outputs of $\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1), \mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)$ and the sets $\mathcal{Z}, \mathcal{Y}_1, \mathcal{Y}_2$ are identical in $\mathbb{G}_{13}$ and Encode and the outputs of $\mathsf{Eval}(.,.,3), \mathsf{O_v}(.,.,3)$ of Encode are identical to the outputs of $\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)$ of $\mathbb{G}_{13}$ respectively.

Therefore, $\mathbb{G}_{13} \Rightarrow \mathsf{true}$ implies that $|\mathcal{Z}| \geqslant l$ in Encode. So, for good σ's, the probability that Encode produces a valid encoding (i.e. does not output D) is ϵ.

Since, Encode produces a valid encoding whenever $|\mathcal{Z}| \geqslant l$, i.e. whenever $\mathbb{G}_{13}$ outputs true given that σ was sampled in $\mathbb{G}_{13}$ we have,

$$\Pr\left[\text{Decoding is correct}\right] \geqslant \epsilon\ . \tag{22}$$

At the end of execution of Encode, let $|\mathsf{Lbls}| = b_1, |\mathsf{RLbls}| = b_2, |\mathsf{Vals}| = o, |\mathsf{Free}| = f, |\mathsf{Inputs}| = i, |\mathsf{Srps}_1| = s_1, |\mathsf{Srps}_2| = s_2, |\mathsf{T}| = t$. Let the size of the internal state output by $\mathcal{R}_1$ be s. We make the following observations about the outputs of Encode.

- For every $i \in \mathbb{Z}_p$, exactly one of the following is true.
  - i is present in Vals.
  - $\sigma(i)$ is present in Lbls.
  - $\sigma(i)$ is present in RLbls.
  - $\sigma(i)$ was queried by $\mathcal{R}_2$ and an entry was added to Inputs.
  - $\sigma(i)$ was queried by $\mathcal{R}_2$ and an entry was added to Free.

Therefore, $|\mathsf{Lbls}| + |\mathsf{RLbls}| + |\mathsf{Vals}| + |\mathsf{Free}| + |\mathsf{Inputs}| = p$ i.e. $b_1 + b_2 + o + f + i = p$. Also note that every label that is not present in T is added to RLbls, so $b_2 = p - t$ and $b_1 = t - o - (i + f)$
- Since a placeholder X_i is introduced only when an entry is added to Free and at least one placeholder is removed from T whenever an entry is added to Srps_1 or Srps_2, we have $|\mathsf{Srps}_1| + |\mathsf{Srps}_2| \leqslant |\mathsf{Free}|$ i.e. $s_1 + s_2 \leqslant f$.
- $|\mathsf{Free}| + |\mathsf{Inputs}| \geqslant |\mathcal{Z}|$ because every entry in $\mathcal{Z}$ is in one of $\mathsf{Free}, \mathsf{Inputs}$. We also have that $|\mathcal{Z}| \geqslant l$ (for any encoding that is output, including D). It follows that $|\mathsf{Free}| + |\mathsf{Inputs}| \geqslant l$ i.e. $f + i \geqslant l$.

Claim. If σ is "good" then the size of the encoding space is at most

$$2^s p! \left(1 + \frac{6q}{p}\right)^{(2q-l)} \left(\frac{p}{8q^2(2k+2+3q)}\right)^{-l} .$$

Proof. Below, we establish upper bounds on the number of possibilities of various components present in the encoding.

- Each entry in Free is a tuple where the first element is in $[q]$ and the second element is in $[2]$. There are f entries in total. Therefore, the number of possibilities of Free is $(2q)^f$.
- Since each entry in Srps_1 consists of triple whose first element is in $[q]$, second element is in $[t]$ (an index of T). Therefore, the number of possibilities of Srps_1 is $(2tq)^{s_1}$.
- Since each entry in Srps_2 consists of triple whose first element is in $[q]$, second element is in $[t]$. Therefore, the number of possibilities of Srps_2 is $(2tq)^{s_2}$.
- Each entry of Inputs consists of a tuple whose first element is in $[q]$ and the second element is in $[3]$ and the third element is in $[k]$. Since there are i entries in total, there are at most $(3qk)^i$ possibilities.
- The list values has o entries. Since $2k+2$ values are already known to not be in Vals (as $1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdots v$ will never be in Vals) and all the entries are distinct and are in $\mathbb{Z}_p$, there are $\binom{p-2k-2}{o} o!$ possibilities.
- The list Lbls has $b_1 = t - o - (i + f)$ distinct labels each of which are in $\mathcal{L}$ (a set of size p). Hence, the total number of possibilities are at most $\binom{p}{t-o-(i+f)}(t - o - (i + f))!$.
- Since the labels being added to the list RLbls are the only ones absent in T, and T is fully populated in Decode before RLbls is used, it suffices to remember their lexicographical ordering. There are $p - t$ known distinct labels in RLbls. The number of possibilities are $(p - t)!$.
- The state output by $\mathcal{R}_1$ is s bits, hence the number of possibilities are 2^s.

The encoding space is at most the product of the number of possibilities of all the components. Hence, the size of the encoding space is upper bounded by

$$2^s (2q)^f (2tq)^{(s_1+s_2)} (3qk)^i \binom{p-2k-2}{o} o! \binom{p}{t-o-(i+f)} (t - o - (i + f))!(p - t)! .$$

Since $s_1 + s_2 \leqslant f$, and $4t > 3k$ (since $t \geqslant 2k + 2$), the size of the encoding space is at most

$$2^s (4tq^2)^{(f+i)} \binom{p-2k-2}{o} o! \binom{p}{t-o-(i+f)} (t - o - (i + f))!(p - t)! .$$

Since $o \leqslant 2q - |\mathcal{Z}| \leqslant 2q - l$ (as $\mathsf{Vals}, \mathcal{Z}$ are populated only for labels on which queries are made), we have

$$\begin{aligned}
\binom{p}{t-o-(i+f)} (t - o - (i + f))! \binom{p-2k-2}{o} o!(p - t)! &\leqslant p! \left(\frac{(p-2k-2)!}{(p-2k-2-(i+f))!} \frac{(p-t)!}{(p-t+o+i+f)!}\right) \\
&\leqslant p! \left(\frac{(p-2k-2)^o}{(p-t)^{o+i+f}}\right) \\
&\leqslant p! \left(\frac{p-2k-2}{p-t}\right)^{2q-l} (p-t)^{-(i+f)} .
\end{aligned}$$

Since $t \leqslant 2k + 2 + 3q$ (size of T can increase by at most 3 on every query), $i + f \geqslant l$, $o \leqslant 2q - l$, and $p - 2k - 2 - 3q \geqslant \frac{p}{2}$ (because $6q \leqslant p - 4k - 4$) we have the size of the encoding space is at most

$$2^s p! \left(1 + \frac{6q}{p}\right)^{(2q-l)} \left(\frac{p}{8q^2(2k + 2 + 3q)}\right)^{-l}.$$

This concludes the proof.

□

5 Conclusions

Despite a clear restriction of our result to straightline reductions, we believe the main contribution of this work is the introduction of novel techniques for proving lower bounds on the memory of reductions that will find wider applicability. In particular, we clearly departed from the framework of prior works [2,13] tailored at the usage of lower bounds for streaming algorithms, and provided the first lower bound for "algebraic" proofs in the public-key domain. The idea of a problem-specific proof of memory could be helpful elsewhere.

Of course, there are several open problems. It seems very hard to study the role of rewinding for such reductions. In particular, the natural approach is to resort to techniques from communication complexity (and their incarnation as streaming lower bounds), as they are amenable to the multi-pass case. The simple combinatorial nature of these lower bounds however is at odds with the heavily structured oracles we encounter in the generic group model. Another problem we failed to solve is to give an adversary $\mathcal{A}$ in our proof which uses little memory – we discuss a candidate in the body, but analyzing it seems to give us difficulties similar to those of rewinding.

This latter point makes a clear distinction, not discussed by prior works, between the *way* in which we prove memory-tightness (via reductions using small memory), and its most general interpretation, as defined in [2], which would allow the reduction to adapt its memory usage to that of $\mathcal{A}$.

Acknowledgements

We thank the anonymous reviewers of EUROCRYPT 2020 for helpful comments. This work was partially supported by NSF grants CNS-1930117 (CAREER), CNS-1926324, and by a Sloan Research Fellowship.

References

1. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001.
2. Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Memory-tight reductions. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 101–132. Springer, Heidelberg, August 2017.
3. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.
4. Rishiraj Bhattacharyya. Memory-tight reductions for practical key encapsulation mechanisms. In *PKC 2020*.
5. Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 415–447. Springer, Heidelberg, April / May 2018.
6. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
7. Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 649–665. Springer, Heidelberg, August 2010.

8. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.
9. 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, Heidelberg, December 2005.
10. 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, Heidelberg, February 2004.
11. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.
12. Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. http://eprint.iacr.org/2001/112.
13. Yuyu Wang, Takahiro Matsuda, Goichiro Hanaoka, and Keisuke Tanaka. Memory lower bounds of reductions revisited. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 61–90. Springer, Heidelberg, April / May 2018.

A Memory-Tight Reduction in the AGM

In the algebraic group model (AGM) [8], the adversary is assumed to only produce new group elements by applying the group operation to received group elements. In particular, for every group element X that the adversary outputs, it also sends a representation $\vec{x}$ of X with respect to the group elements it received as inputs. For example, let G be a group of order p. Suppose an adversary that outputs one group element received group elements U, V as inputs. It shall output $[X]_{\vec{x}}$ where X is a group element and $\vec{x} = x_1, x_2 \in \mathbb{Z}_p$ such that $X = U^{x_1} V^{x_2}$.

Before giving the memory-tight reduction to the discrete logarithm problem in the AGM, we shall first describe the discrete-logarithm problem, re-formulate the ODH assumption in the random oracle model and AGM, and formally introduce PRFs (pseudorandom functions)- a tool which shall be required for our memory-tight reduction.

Figure 20 formally defines the game for the discrete logarithm problem in a group G. The adversary is given as input a generator and a random element of a group G. The game outputs true if the adversary can successfully output the discrete log of the random group element with respect to the generator. The advantage of an adversary against the discrete logarithm problem in a group G is defined as

$$\mathsf{Adv}^{\mathsf{DL}}_{G}(\mathcal{A}) = \Pr\left[\mathbb{G}^{\mathsf{DL}}_{G}(\mathcal{A}) \Rightarrow \mathsf{true}\right] .$$

We have previously formalized the ODH assumption in the random oracle and the generic group models. Here, we give a formalization of this assumption in the random-oracle and algebraic group models. Let G be a group of order p. For a fixed $\mathsf{hLen} \in \mathbb{N}$, let Ω_{hLen} be the set of hash functions mapping $\{0,1\}^*$ to $\{0,1\}^{\mathsf{hLen}}$. In Figure 21, we formally define the games $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}_{G,\mathsf{hLen}}$, $\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}AGM}}_{G,\mathsf{hLen}}$. The difference here from the previous definition is that for every $\mathsf{H}, \mathsf{H_v}$ query the adversary makes, it also sends over a representation of the group element with respect to its input group elements i.e. g, U, V. (Also, compared to the previous definition, the generic group oracle is absent here because we are no more in the generic group model). The advantage of violating ODH is defined as

$$\mathsf{Adv}^{\mathsf{ODH\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) = \left|\Pr\left[\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right]\right| .$$

We now briefly introduce the notion of PRF security. Let $F : K \times D \to R$ be an efficiently computable keyed function. Let $\mathsf{RF}_{D,R}$ be a random function mapping elements of D to R. Consider the games in Figure 20. The advantage of an adversary $\mathcal{A}$ against the PRF security of F is defined as,

$$\mathsf{Adv}^{\mathsf{PRF}}_{F}(\mathcal{A}) = \left|\Pr\left[\mathbb{G}^{\mathsf{PRF\text{-}REAL}}_{F}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbb{G}^{\mathsf{PRF\text{-}RAND}}_{F}(\mathcal{A}) \Rightarrow 1\right]\right| .$$

Game $\mathbb{G}_G^{\mathsf{DL}}(\mathcal{A})$:	**Game** $\mathbb{G}_F^{\mathsf{PRF\text{-}REAL}}(\mathcal{A})$:	**Game** $\mathbb{G}_F^{\mathsf{PRF\text{-}RAND}}(\mathcal{A})$:
1 : $g \leftarrow\$\, G^*$	1 : $k \leftarrow\$\, K$	1 : $b \leftarrow \mathcal{A}^{\mathsf{RF}_{D,R}(\cdot)}$
2 : $h \leftarrow\$\, G$	2 : $b \leftarrow \mathcal{A}^{F(k,\cdot)}$	2 : **return** b
3 : $v \leftarrow \mathcal{A}(g,h)$	3 : **return** b	
4 : **return** $(h = g^v)$		

Fig. 20. Left: Game for the discrete logarithm problem in a group G of prime order p where $G^* = G\backslash\{0\}$ is the set of generators. **Right:**Games for PRF security

Game $\mathbb{G}_{G,\mathsf{hLen}}^{\mathsf{ODH\text{-}RAND\text{-}AGM}}$:	**Game** $\mathbb{G}_{G,\mathsf{hLen}}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}$:	**Oracle** $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$: // $X = g^{x_1}U^{x_2}V^{x_3}$
1 : $u \leftarrow\$\, \mathbb{Z}_p; U \leftarrow g^u$	1 : $u \leftarrow\$\, \mathbb{Z}_p; U \leftarrow g^u$	1 : **return** $\mathsf{H}(X)$
2 : $v \leftarrow\$\, \mathbb{Z}_p; V \leftarrow g^v$	2 : $v \leftarrow\$\, \mathbb{Z}_p; V \leftarrow g^v$	
3 : $\mathsf{H} \leftarrow\$\, \Omega_{\mathsf{hLen}}$	3 : $\mathsf{H} \leftarrow\$\, \Omega_{\mathsf{hLen}}$	
4 : $W \leftarrow \{0,1\}^{\mathsf{hLen}}$	4 : $W \leftarrow \mathsf{H}(g^{uv})$	
5 : $b \leftarrow \mathcal{A}^{\mathsf{H_v}(\cdot),\mathsf{H}(\cdot)}(g,U,V,W)$	5 : $b \leftarrow \mathcal{A}^{\mathsf{H_v}(\cdot),\mathsf{H}(\cdot)}(g,U,V,W)$	
6 : **return** b	6 : **return** b	
Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$: // $Y = g^{y_1}U^{y_2}V^{y_3}$		
1 : **if** $Y = U$ **then return** $\perp$		
2 : **return** $\mathsf{H}(Y^v)$		

Fig. 21. Games for ODH assumption in the AGM.

Memory-tight AGM reduction. In the following theorem (Theorem 2), we show that for all adversaries in the AGM against ODH, there exist adversaries against DL and PRF security of keyed functions that use additional memory that grows at most logarithmically in the number of queries of the ODH adversary i.e. the reductions are memory-tight.

Theorem 2. *Let $F_1 : K_{F_1} \times \mathbb{Z}_p^3 \to \{0,1\}^{\mathsf{hLen}}$, $F_2 : K_{F_2} \times G \to \{0,1\}^{\mathsf{hLen}}$ be keyed functions. For all ODH adversaries $\mathcal{A}$ in the Algebraic Group Model, making a total of q queries to $\mathsf{H}, \mathsf{H_v}$, there exist adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}, \mathcal{G}$ such that*

$$\mathsf{Adv}_{G,\mathsf{hLen}}^{\mathsf{ODH\text{-}AGM}}(\mathcal{A}) \leqslant \mathsf{Adv}_G^{\mathsf{DL}}(\mathcal{B}) + q^2\mathsf{Adv}_G^{\mathsf{DL}}(\mathcal{C}) + \mathsf{Adv}_G^{\mathsf{DL}}(\mathcal{D}) + \mathsf{Adv}_{F_1}^{\mathsf{PRF}}(\mathcal{E}) + q^2\mathsf{Adv}_{F_1}^{\mathsf{PRF}}(\mathcal{F}) + \mathsf{Adv}_{F_2}^{\mathsf{PRF}}(\mathcal{G}) \ .$$

Adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}, \mathcal{G}$ are nearly as efficient as $\mathcal{A}$ in terms of time complexity. Moreover, adversaries $\mathcal{B}, \mathcal{E}$ use at most $8\log p + \log|K_{F_1}| + 2\mathsf{hLen}$ bits of memory in addition to the memory used by $\mathcal{A}$, adversaries $\mathcal{C}, \mathcal{F}$ use at most $15\log p + \log|K_{F_1}| + 2\mathsf{hLen} + 4\log q$ bits of memory in addition to the memory used by $\mathcal{A}$, and adversaries $\mathcal{D}, \mathcal{G}$ use at most $8\log p + \log|K_{F_2}| + 2\mathsf{hLen}$ bits of memory in addition to the memory used by $\mathcal{A}$.

Proof. Before diving into the details of the proof, we shall provide some intuition. For a memory-tight reduction, the $\mathsf{H}, \mathsf{H_v}$ oracles have to be simulated with low memory. The reason it is non-trivial is because the adversary can make a $\mathsf{H_v}$ query on Y and a H query on $X = Y^v$, and the reduction must reply with the same answers for both queries. In the AGM, however, the adversary would need to send over $\vec{x} = x_1x_2x_3$ and $\vec{y} = y_1y_2y_3$, the representation of X and Y respectively i.e. $X = g^{x_1}U^{x_2}V^{x_3}$ and $Y = g^{y_1}U^{y_2}V^{y_3}$. We shall next outline a strategy for a reduction to the discrete logarithm problem, point out flaws in the strategy and fix them.

The reduction to the discrete logarithm problem gets as input (g, V). It would sample u from $\mathbb{Z}_p$ and sample W at random and run the ODH adversary $\mathcal{A}$. For every H query on X with representation $x_1x_2x_3$, the reduction would associate a polynomial $m_X = x_1 + ux_2 + x_3S$ in the variable S. Observe that the reduction is supposed to respond with $\mathsf{H}(g^{x_1+ux_2+x_3v})$ i.e. $\mathsf{H}(g^{m_X(v)})$. For every $\mathsf{H_v}$ query on Y with representation $y_1y_2y_3$, the reduction would associate a polynomial $m_{Y^v} = y_1S + uy_2S + y_3S^2$. Observe that the reduction is supposed to respond with $\mathsf{H}(g^{y_1v+uy_2v+y_3v^2})$ i.e. $\mathsf{H}(g^{m_{Y^v}(v)})$. Essentially the reduction could associate a

univariate polynomial of degree at most two for every query. Since a univariate polynomial of degree at most two has at most three coeffecients, the reduction could respond with the output of a PRF that takes three elements of $\mathbb{Z}_p$ as input and returns a value in $\{0,1\}^{\mathsf{hLen}}$. Finally when the adversary makes a H query on g^{uv} (this can be checked by the reduction by checking if the queried value equals V^u), the reduction could solve for v from the representation of g^{uv}, i.e. suppose the representation was $x_1x_2x_3$ then the reduction could output $(x_1 + ux_2)(x_3 - u)^{-1}$ if $x_3 \neq u$. (Also it is easy to show that the adversary cannot have any advantage against ODH if it does not make a H query on g^{uv}).

This strategy however has two flaws. The first flaw is that the adversary could potentially come up with different representations for the same group element. In this case the strategy we described would yield different answers to the two queries. An even more contrived case would be when the reduction makes H_v query on Y with representation $y_1y_2y_3$ and a H query on $X = Y^v$ with representation $x_1x_2x_3$ and not all of y_3, x_1, x_2 are zero and $y_1 + uy_2 \neq x_3$. In this case again our strategy would yield different answers. The second flaw is that when the the adversary makes a H query on g^{uv} with representation $x_1x_2x_3$ such that $x_3 = u$. We first will talk about how to handle the second flaw because it is simpler.

The second flaw can be done away with by giving a separate reduction to discrete logarithm that sets U to be its discrete logarithm instance input. This reduction would be simpler because it picks its own v and can simulate $\mathsf{H}, \mathsf{H}_\mathsf{v}$ queries by using a PRF that takes as input a group element and outputs a string in $\{0,1\}^{\mathsf{hLen}}$.

In order to handle the first flaw, we give another reduction to the discrete logarithm problem that answers queries similar to the first one but computes its output differently. This reduction randomly chooses two values in $\{1, \cdots, q\}$ where q is the total number of $\mathsf{H}, \mathsf{H}_\mathsf{v}$ queries the adversary makes and remembers the polynomial associated with these two indices. After running the adversary to completion the reduction equates the two remembered polynomial, finds solutions and checks if any of the solutions is the correct discrete logarithm. Note that if the adversary is indeed able to engineer a scenario that we described in the first flaw, then at least two of the q polynomials are distinct but equal when evaluated at v (because they are different representations of the same group element). So, with probability $\frac{1}{q^2}$ this reduction would choose the right polynomials if the scenario related to the first flaw happens. If it chooses the right polynomials, it can successfully compute the discrete logarithm. Note that for this reduction there is a multiplicative advantage loss of factor q^2. Now, we shall start with the formal proof.

First off, we define two games $\mathbb{G}_0, \mathbb{G}_1$ in Figure 22. Observe that $\mathbb{G}_0$ perfectly simulates $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}_{G,\mathsf{hLen}}$ to $\mathcal{A}$. The only difference in $\mathbb{G}_0$ from $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}_{G,\mathsf{hLen}}$ is that the random oracle H is lazily sampled in $\mathbb{G}_0$ and there is some extra bookkeeping. The lazy sampling and bookkeeping does not affect the view of $\mathcal{A}$ in any way, and the output of $\mathbb{G}_0$ is identical to that of $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}_{G,\mathsf{hLen}}$ when interacting with $\mathcal{A}$. Hence we have that

$$\Pr\left[\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbb{G}_0 \Rightarrow 1\right] .$$

Similarly, $\mathbb{G}_1$ perfectly simulates $\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}AGM}}_{G,\mathsf{hLen}}$ to $\mathcal{A}$ and the output of $\mathbb{G}_1$ is identical to that of $\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}AGM}}_{G,\mathsf{hLen}}$ when interacting with $\mathcal{A}$. Thus

$$\Pr\left[\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] = \Pr\left[\mathbb{G}_1 \Rightarrow 1\right] .$$

Therefore we have that

$$\mathsf{Adv}^{\mathsf{ODH\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) = \left|\Pr\left[\mathbb{G}_0 \Rightarrow 1\right] - \Pr\left[\mathbb{G}_1 \Rightarrow 1\right]\right| . \tag{23}$$

Now, observe that $\mathbb{G}_0, \mathbb{G}_1$ are identical if the flag ASKA is not set to true in both of them. Using the Fundamental Lemma of Game Playing, we have

$$\left|\Pr\left[\mathbb{G}_0 \Rightarrow 1\right] - \Pr\left[\mathbb{G}_1 \Rightarrow 1\right]\right| \leqslant \Pr\left[\mathsf{ASKA} = \mathsf{true} \text{ in } \mathbb{G}_1\right] .$$

Therefore it follows from (23) that

$$\mathsf{Adv}^{\mathsf{ODH\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) \leqslant \Pr\left[\mathsf{ASKA} = \mathsf{true} \text{ in } \mathbb{G}_1\right] . \tag{24}$$

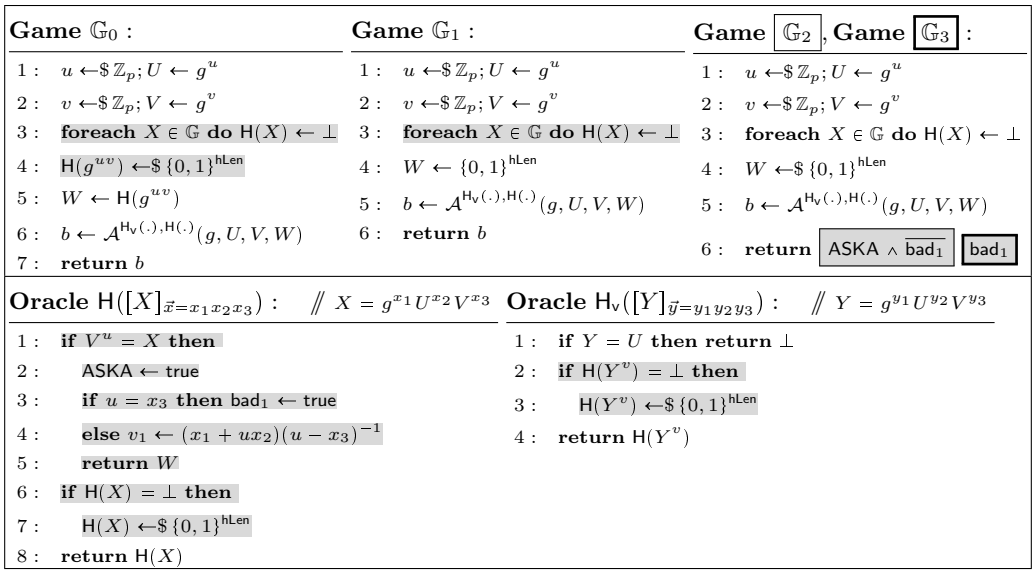

Fig. 22. Games $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$. The differences of $\mathbb{G}_0$ from $\mathbb{G}^{\mathsf{ODH\text{-}REAL\text{-}AGM}}_{G,\mathsf{hLen}}$ have been highlighted. The differences of $\mathbb{G}_1$ from $\mathbb{G}^{\mathsf{ODH\text{-}RAND\text{-}AGM}}_{G,\mathsf{hLen}}$ have been highlighted. In $\mathbb{G}_2$, $\mathbb{G}_3$ the code in the thinner box is present only in $\mathbb{G}_2$ and the code in the thicker box is present only in $\mathbb{G}_3$. The differences in $\mathbb{G}_2, \mathbb{G}_3$ from $\mathbb{G}_1$ have been highlighted. The $\mathsf{H}, \mathsf{H_v}$ oracles have not been specified for $\mathbb{G}_2, \mathbb{G}_3$ because they are identical as that in $\mathbb{G}_1$.

Next, we introduce two games $\mathbb{G}_2$, $\mathbb{G}_3$ in Figure 22. The code in the thinner box is present only in $\mathbb{G}_2$ and the code in the thicker box is present only in $\mathbb{G}_3$. We shall use this convention throughout this proof. Note that, since $\mathbb{G}_2$ is identical to $\mathbb{G}_1$ except for the return value and returns $\mathsf{ASKA} \wedge \overline{\mathsf{bad}_1}$, we have that

$$\Pr\left[\mathsf{ASKA} \wedge \overline{\mathsf{bad}_1} = \mathsf{true} \text{ in } \mathbb{G}_1\right] = \Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] .$$

Similarly since $\mathbb{G}_3$ is identical to $\mathbb{G}_1$ except for the return value and returns bad_1, we have that

$$\Pr\left[\mathsf{bad}_1 = \mathsf{true} \text{ in } \mathbb{G}_1\right] = \Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] .$$

Since

$$\Pr\left[\mathsf{ASKA} = \mathsf{true} \text{ in } \mathbb{G}_1\right] \leqslant \Pr\left[\mathsf{ASKA} \wedge \overline{\mathsf{bad}_1} = \mathsf{true} \text{ in } \mathbb{G}_1\right] + \Pr\left[\mathsf{bad}_1 = \mathsf{true} \text{ in } \mathbb{G}_1\right] ,$$

it follows that

$$\Pr\left[\mathsf{ASKA} = \mathsf{true} \text{ in } \mathbb{G}_1\right] = \Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] + \Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] .$$

Combining with (24) we get

$$\mathsf{Adv}^{\mathsf{ODH\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) \leqslant \Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] + \Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] . \tag{25}$$

Next we introduce $\mathbb{G}_4$ in Figure 23 that is identical to $\mathbb{G}_2$ with some additional bookkeeping. It additionally defines a mapping m on points where H is defined that maps group elements to polynomials in $\mathbb{Z}_p[S]$. All group elements X on which to H query was made is mapped to $x_1 + ux_2 + Sx_3$ where (x_1, x_2, x_3) is the representation of X. Group elements Y^v such that a $\mathsf{H_v}$ query was made on Y is mapped to $y_1S + uy_2S + y_3S^2$ where (y_1, y_2, y_3) is the representation of Y. The mapping of a group element X is denoted by m_X. Observe that for all X if m_X is defined then $X = g^{\mathsf{m}_X(v)}$. Since, $\mathbb{G}_4$ involves only additional bookkeeping compared to $\mathbb{G}_1$ and the additional bookkeeping in no way affects the flag ASKA

$$\Pr\left[\mathbb{G}_2 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_4 \Rightarrow \mathsf{true}\right] . \tag{26}$$

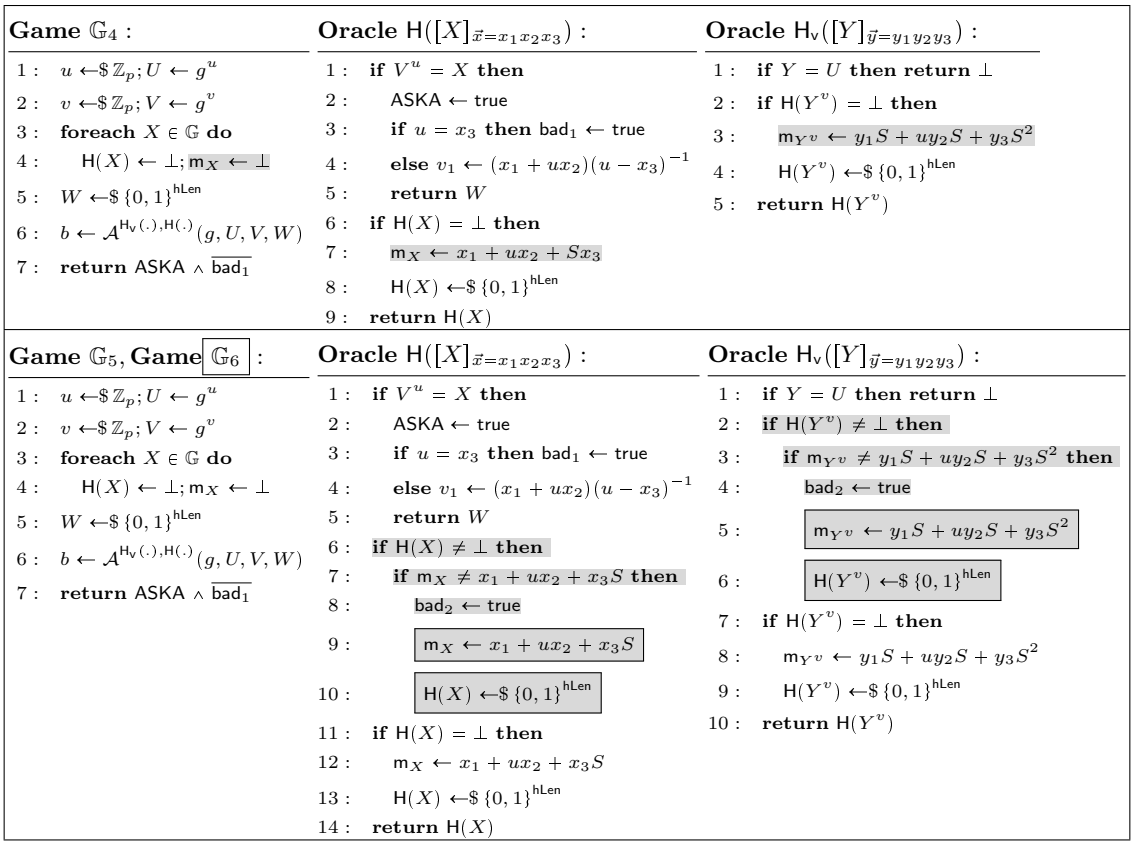

Fig. 23. Games $\mathbb{G}_4, \mathbb{G}_5, \mathbb{G}_6$. The differences of $\mathbb{G}_4$ from $\mathbb{G}_2$ have been highlighted. The differences of $\mathbb{G}_5, \mathbb{G}_6$ from $\mathbb{G}_4$ have been highlighted. In $\mathbb{G}_5$, $\mathbb{G}_6$ the code in the box is present only in $\mathbb{G}_6$.

Game $\mathbb{G}_5$ in Figure 23 introduces a bad_2 event compared to $\mathbb{G}_4$. The bad_2 event happens during a H query on X with representation x_1, x_2, x_3 if m_X is previously defined but is not equal to $x_1 + ux_2 + x_3S$. In case this event happens m_X is reassigned to $x_1 + ux_2 + x_3S$. The bad_2 event happens during a $\mathsf{H_v}$ query on Y with representation y_1, y_2, y_3 if m_{Y^v} is previously defined but is not equal to $y_1S + uy_2S + y_3S^2$. In case this event happens m_{Y^v} is reassigned to $y_1S + uy_2S + y_3S^2$. Observe that even though m_X is reassigned, it is still true that if m_X is defined then $X = g^{\mathsf{m}_X(v)}$ (same for Y^v). This bad_2 event does not affect anything in $\mathbb{G}_5$, hence

$$\Pr\left[\mathbb{G}_4 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_5 \Rightarrow \mathsf{true}\right] . \tag{27}$$

Next, we introduce game $\mathbb{G}_6$ in Figure 23. It differs from $\mathbb{G}_5$ only when bad_2 is set to true. Whenever bad_2 happens in $\mathbb{G}_6$ during a H query on X, $\mathsf{H}(X)$ is re-sampled and whenever bad_2 happens in $\mathbb{G}_6$ during a $\mathsf{H_v}$ query on Y, $\mathsf{H}(Y^v)$ is re-sampled. Note that $\mathbb{G}_5, \mathbb{G}_6$ are identical if the bad_2 flag is not set in either of them. Using the Fundamental Lemma of Game Playing we have

$$\Pr\left[\mathbb{G}_5 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_6 \Rightarrow \mathsf{true}\right] + \Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_6\right] . \tag{28}$$

Observe that in $\mathbb{G}_6$, for all group elements X, $\mathsf{H}(X)$ is sampled uniformly at random from $\{0,1\}^{\mathsf{hLen}}$ if and only if m_X is updated in the previous step. Moreover, since $g^{\mathsf{m}_X(v)} = X$, $\mathsf{m}_{X_1} \neq \mathsf{m}_{X_2}$ if $X_1 \neq X_2$. In $\mathbb{G}_7$, therefore instead of randomly sampling the value of $\mathsf{H}(X)$, we can assign it the output of a random function that takes as input the coefficients of m_X and returns a value in $\{0,1\}^{\mathsf{hLen}}$. Let $\mathsf{RF}_{\mathbb{Z}_p^3,\{0,1\}^{\mathsf{hLen}}}$ be a random function mapping $\mathbb{Z}_p^3$ to $\{0,1\}^{\mathsf{hLen}}$. So, $\mathbb{G}_7$ remains identical to $\mathbb{G}_6$ and we have the following.

$$\Pr\left[\mathbb{G}_6 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathbb{G}_7 \Rightarrow \mathsf{true}\right] . \tag{29}$$

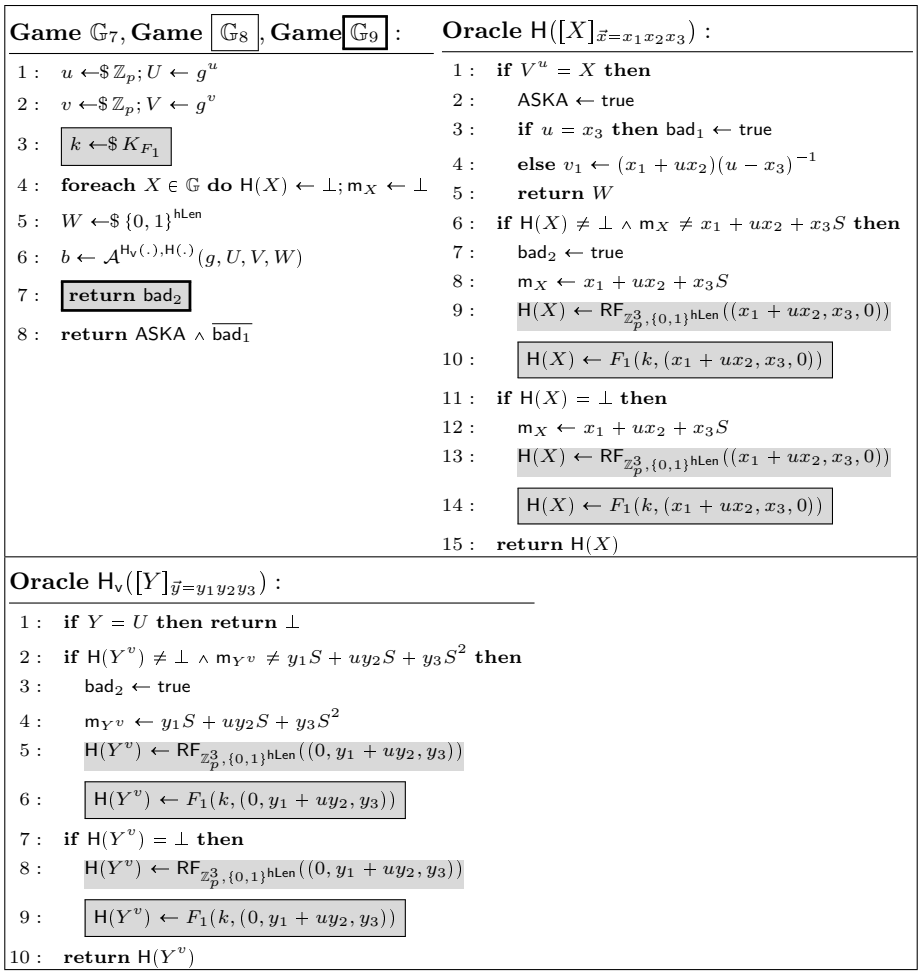

Fig. 24. Games $\mathbb{G}_7, \mathbb{G}_8, \mathbb{G}_9$.The differences of these games from $\mathbb{G}_6$ have been highlighted. The code in the thinner box is present only in $\mathbb{G}_8$ and the code in the thicker box is present only in $\mathbb{G}_9$.

$$\Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_6\right] = \Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_7\right] . \tag{30}$$

Next, we replace the random function $\mathsf{RF}_{\mathbb{Z}_p^3,\{0,1\}^{\mathsf{hLen}}}$ with a keyed function $F_1 : \mathbb{Z}_p^3 \times K_{F_1} \rightarrow \{0,1\}^{\mathsf{hLen}}$ in $\mathbb{G}_8$ (Figure 24). Consider adversary $\mathcal{E}$ in Figure 25. It is easy to see that $\mathcal{E}$ simulates $\mathbb{G}_7$ to $\mathcal{A}$ when interacting with $\mathbb{G}_{F_1}^{\mathsf{PRF\text{-}RAND}}$ i.e. when it can query $O = \mathsf{RF}_{\mathbb{Z}_p^3,\{0,1\}^{\mathsf{hLen}}}(.)$ and returns 1 if and only $\mathbb{G}_7$ returns true. It simulates $\mathbb{G}_8$ to $\mathcal{A}$ when interacting with $\mathbb{G}_{F_1}^{\mathsf{PRF\text{-}REAL}}$ i.e. when it can query $O = F_1(k, .)$ and returns 1 if and only $\mathbb{G}_8$ returns true. Therefore,

$$\Pr\left[\mathbb{G}_7 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_8 \Rightarrow \mathsf{true}\right] + \mathsf{Adv}_{F_1}^{\mathsf{PRF}}(\mathcal{E}) . \tag{31}$$

It can be verified from the pseudocode that adversary $\mathcal{E}$ uses at most $8 \log p + \log |K_{F_1}| + 2\mathsf{hLen}$ bits of memory in addition to the memory used by $\mathcal{A}$ and is nearly as efficient as $\mathcal{A}$.

We can simplify $\mathbb{G}_8$ and re-write it as shown in Figure 26. We next introduce $\mathbb{G}_9$ in Figure 24 that is identical to $\mathbb{G}_7$ except that it returns the value of bad_2. So we have

$$\Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_7\right] = \Pr\left[\mathbb{G}_9 \Rightarrow \mathsf{true}\right] . \tag{32}$$

Next we introduce $\mathbb{G}_{10}$ which is identical to $\mathbb{G}_8$ except that it returns the boolean value $(v_1 = v)$ instead of $\mathsf{ASKA} \wedge \overline{\mathsf{bad}_1}$. Suppose the query for which the flag ASKA is set in $\mathbb{G}_{10}$ is on $[X]_{\vec{x}=x_1x_2x_3}$. Since ASKA is set

Adversary $\mathcal{E}^O$:// $O : \mathbb{Z}_p^3 \to \{0,1\}^{\mathsf{hLen}}$

$u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
$v \leftarrow\$ \mathbb{Z}_p; V \leftarrow g^v$
$W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.)}(g, U, V, W)$
if $\mathsf{ASKA} \wedge \overline{\mathsf{bad}_1} = \mathsf{true}$ **then return** 1
return 0

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

if $V^u = X$ **then**
  $\mathsf{ASKA} \leftarrow \mathsf{true}$
  if $u = x_3$ **then** $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  return W
return $O((x_1 + ux_2, x_3, 0))$

Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$:

if $Y = U$ **then return** $\perp$
return $O((0, y_1 + uy_2, y_3))$

Adversary $\mathcal{F}^O$:// $O : \mathbb{Z}_p^3 \to \{0,1\}^{\mathsf{hLen}}$

$u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
$v \leftarrow\$ \mathbb{Z}_p; V \leftarrow g^v$
$q_1 \leftarrow\$ [q]; q_2 \leftarrow\$ [q] \backslash q_1$
$\mathsf{p}_1 \leftarrow \perp, \mathsf{p}_2 \leftarrow \perp$
$k \leftarrow\$ K_{F_1}; v_2 \leftarrow 0$
$W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
$q' \leftarrow 0$
$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.)}(g, U, V, W)$
$v_{21}, v_{22} \leftarrow$ solutions obtained from equating $\mathsf{p}_1, \mathsf{p}_2$
if $g^{v_{21}} = V$ **then** $v_2 \leftarrow v_{21}$
else $v_2 \leftarrow v_{22}$
return $(v_2 = v)$

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

$q' \leftarrow q' + 1$
if $V^u = X$ **then**
  return W
if $q' = q_1$ **then** $\mathsf{p}_1 \leftarrow x_1 + ux_2 + x_3S$
if $q' = q_2$ **then** $\mathsf{p}_2 \leftarrow x_1 + ux_2 + x_3S$
return $O((x_1 + ux_2, x_3, 0))$

Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$:

$q' \leftarrow q' + 1$
if $Y = U$ **then return** $\perp$
if $q' = q_1$ **then** $\mathsf{p}_1 \leftarrow y_1S + uy_2S + y_3S^2$
if $q' = q_2$ **then** $\mathsf{p}_2 \leftarrow y_1S + uy_2S + y_3S^2$
return $O((0, y_1 + uy_2, y_3))$

Fig. 25. Adversaries $\mathcal{E}$, $\mathcal{F}$. It is easy to see that $\mathcal{E}$ simulates $\mathbb{G}_7$ to $\mathcal{A}$ when interacting with $\mathbb{G}_F^{\mathsf{PRF\text{-}RAND}}$ and simulates $\mathbb{G}_8$ to $\mathcal{A}$ when interacting with $\mathbb{G}_F^{\mathsf{PRF\text{-}REAL}}$. Adversary $\mathcal{F}$ simulates $\mathbb{G}_{11}$ to $\mathcal{A}$ when interacting with $\mathbb{G}_F^{\mathsf{PRF\text{-}RAND}}$ and simulates $\mathbb{G}_{12}$ to $\mathcal{A}$ when interacting with $\mathbb{G}_F^{\mathsf{PRF\text{-}REAL}}$.

for this query, $X = g^{uv}$. If bad_1 is not set to true in $\mathbb{G}_{10}$, then v_1 is indeed equal to v. Therefore

$$\Pr\left[\mathbb{G}_8 \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{10} \Rightarrow \mathsf{true}\right] . \tag{33}$$

Now, consider adversary $\mathcal{B}$ against the discrete logarithm problem in Figure 27. Observe that it simulates $\mathbb{G}_{10}$ perfectly to $\mathcal{A}$ and outputs v_1. Since $\mathbb{G}_{10}$ outputs true if and only if v_1 is the discrete logarithm of V, it follows that

$$\Pr\left[\mathbb{G}_{10} \Rightarrow \mathsf{true}\right] = \mathsf{Adv}_G^{\mathsf{DL}}(\mathcal{B}) . \tag{34}$$

Observe that $\mathcal{B}$ is nearly as efficient as $\mathcal{A}$. It requires memory to store U, V, W, u, k, v_1, the current query and the return value. These can be stored in at most $8 \log p + \log |K_{F_1}| + 2\mathsf{hLen}$ bits.

Next we introduce $\mathbb{G}_{11}$ which is similar to $\mathbb{G}_9$ with a few modifications. First, it keeps a counter for the total number of queries that $\mathcal{A}$ makes. It randomly picks two distinct numbers q_1 and q_2 from 1 through q where q is the total number of queries $\mathcal{A}$ makes to $\mathsf{H}, \mathsf{H_v}$. It stores the polynomial defined for the q_1^{st} query in p_1 and the polynomial defined for the q_2^{nd} query in p_2. After the execution of $\mathcal{A}$, it equates $\mathsf{p}_1, \mathsf{p}_2$ to obtain solutions v_{21}, v_{22}. If v_{21} is the discrete logarithm of V, it assigns v_{21} to v_2 and otherwise assigns v_{22} to v_2. (Note that if the equation has only one solution v_{22} is assigned 0 by default). It then returns the boolean

Game $\mathbb{G}_8$:

$u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
$v \leftarrow\$ \mathbb{Z}_p; V \leftarrow g^v$
$k \leftarrow\$ K_{F_1}$
foreach $X \in \mathbb{G}$ **do** $\mathsf{H}(X) \leftarrow \perp; \mathsf{m}_X \leftarrow \perp$
$W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.)}(g, U, V, W)$
return $\mathsf{ASKA} \wedge \overline{\mathsf{bad}_1}$

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

if $V^u = X$ **then**
  $\mathsf{ASKA} \leftarrow \mathsf{true}$
  if $u = x_3$ **then** $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  else $v_1 \leftarrow (x_1 + ux_2)(u - x_3)^{-1}$
  return W
if $\mathsf{H}(X) \neq \perp \wedge \mathsf{m}_X \neq x_1 + ux_2 + x_3S$ **then**
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{m}_X \leftarrow x_1 + ux_2 + x_3S$
$\mathsf{H}(X) \leftarrow F_1(k, (x_1 + ux_2, x_3, 0))$
return $F_1(k, (x_1 + ux_2, x_3, 0))$

Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$:

if $Y = U$ **then return** $\perp$
if $\mathsf{H}(Y^v) \neq \perp \wedge \mathsf{m}_{Y^v} \neq y_1S + uy_2S + y_3S^2$ **then**
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{m}_{Y^v} \leftarrow y_1S + uy_2S + y_3S^2$
$\mathsf{H}(Y^v) \leftarrow F_1(k, (0, y_1 + uy_2, y_3))$
return $F_1(k, (0, y_1 + uy_2, y_3))$

Fig. 26. Game $\mathbb{G}_8$ simplified. Game $\mathbb{G}_8$ has just been re-written in an equivalent form by removing some redundant code.

value ($v_2 = v$). So, $\mathbb{G}_{11}$ has some additional book-keeping compared to $\mathbb{G}_9$ and differs only in the return value. In particular, we have

$$\Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_9\right] = \Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_{11}\right] ,$$

i.e.

$$\Pr\left[\mathbb{G}_9 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_{11}\right] .$$

Now if the flag bad_2 is set to true in $\mathbb{G}_{11}$, it means that either for some X with representation $x_1x_2x_3$ on which a H query was made m_X had previously been defined but was different from $x_1 + ux_2 + Sx_3$ or for some Y with representation $y_1y_2y_3$ on which a $\mathsf{H_v}$ query was made m_{Y^v} had previously been defined but was different from $y_1S + uy_2S + y_3S^2$. Since $\mathsf{m}_{g^x}(v) = x$ by the definition of m, if bad_2 is set to true in $\mathbb{G}_{11}$, there are at least 2 queries by $\mathcal{A}$ such that the polynomial defined to answer the two queries were distinct but gave the same result when evaluated at v. Since $\mathsf{p}_1, \mathsf{p}_2$ are chosen randomly from the q polynomials defined to answer the queries, with probability at least $\frac{1}{q^2}$, $\mathsf{p}_1, \mathsf{p}_2$ are the two distinct but gave the same result when evaluated at v. Now, since $\mathsf{p}_1, \mathsf{p}_2$ can be of degree at most 2, equating them will yield at most two solutions v_{21}, v_{22}. It follows that $v_2 = v$ if the right polynomials $\mathsf{p}_1, \mathsf{p}_2$ are picked, which happens with probability $\frac{1}{q^2}$. So

$$\Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right] \geqslant \frac{1}{q^2} \Pr\left[\mathsf{bad}_2 = \mathsf{true} \text{ in } \mathbb{G}_{11}\right] .$$

Hence we have

$$\Pr\left[\mathbb{G}_9 \Rightarrow \mathsf{true}\right] \leqslant q^2 \Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right] . \tag{35}$$

Next, we replace the random function $\mathsf{RF}_{\mathbb{Z}_p^3,\{0,1\}^{\mathsf{hLen}}}$ with a keyed function $F_1 : \mathbb{Z}_p^3 \times K_{F_1} \rightarrow \{0,1\}^{\mathsf{hLen}}$ in $\mathbb{G}_{12}$ (Figure 28). Consider adversary $\mathcal{F}$ in Figure 25. It is easy to see that $\mathcal{F}$ simulates $\mathbb{G}_{11}$ to $\mathcal{A}$ when interacting with $\mathbb{G}_{F_1}^{\mathsf{PRF\text{-}RAND}}$ i.e. when it can query $O = \mathsf{RF}_{\mathbb{Z}_p^3,\{0,1\}^{\mathsf{hLen}}}(.)$ and returns 1 if and only $\mathbb{G}_{11}$ returns true. It simulates $\mathbb{G}_{12}$ to $\mathcal{A}$ when interacting with $\mathbb{G}_{F_1}^{\mathsf{PRF\text{-}REAL}}$ i.e. when it can query $O = F_1(k, .)$ and returns 1 if and only $\mathbb{G}_{12}$ returns true. Therefore,

$$\Pr\left[\mathbb{G}_{11} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{12} \Rightarrow \mathsf{true}\right] + \mathsf{Adv}_{F_1}^{\mathsf{PRF}}(\mathcal{F}) . \tag{36}$$

Game $\mathbb{G}_{10}$:

$u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
$v \leftarrow\$ \mathbb{Z}_p; V \leftarrow g^v$
$k \leftarrow\$ K_{F_1}; v_1 \leftarrow 0$
foreach $X \in \mathbb{G}$ **do** $\mathsf{H}(X) \leftarrow \perp; \mathsf{m}_X \leftarrow \perp$
$W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.)}(g, U, V, W)$
return $(v_1 = v)$

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

if $V^u = X$ **then**
  $\mathsf{ASKA} \leftarrow \mathsf{true}$
  if $u = x_3$ **then** $\mathsf{bad}_1 \leftarrow \mathsf{true}$
  else $v_1 \leftarrow (x_1 + ux_2)(u - x_3)^{-1}$
  return W
if $\mathsf{H}(X) \neq \perp \wedge \mathsf{m}_X \neq x_1 + ux_2 + x_3S$ **then**
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{m}_X \leftarrow x_1 + ux_2 + x_3S$
$\mathsf{H}(X) \leftarrow F_1(k, (x_1 + ux_2, x_3, 0))$
return $F_1(k, (x_1 + ux_2, x_3, 0))$

Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$:

if $Y = U$ **then return** $\perp$
if $\mathsf{H}(Y^v) \neq \perp \wedge \mathsf{m}_{Y^v} \neq y_1S + uy_2S + y_3S^2$ **then**
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{m}_{Y^v} \leftarrow y_1S + uy_2S + y_3S^2$
$\mathsf{H}(Y^v) \leftarrow F_1(k, (0, y_1 + uy_2, y_3))$
return $F_1(k, (0, y_1 + uy_2, y_3))$

Adversary $\mathcal{B}(g, V)$:

$u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
$k \leftarrow\$ K_{F_1}; v_1 \leftarrow 0$
$W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.)}(g, U, V, W)$
return v_1

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

if $V^u = X$ **then**
  $v_1 \leftarrow (x_1 + ux_2)(u - x_3)^{-1}$
  return W
return $F_1(k, (x_1 + ux_2, x_3, 0))$

Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$:

if $Y = U$ **then return** $\perp$
return $F_1(k, (0, y_1 + uy_2, y_3))$

Fig. 27. Game $\mathbb{G}_{10}$, reduction $\mathcal{B}$. The differences of $\mathbb{G}_{10}$ from $\mathbb{G}_8$ have been highlighted.

It can be verified from the pseudocode that adversary $\mathcal{F}$ uses at most $15 \log p + \log |K_{F_1}| + 2\mathsf{hLen} + 4 \log q$ bits of memory in addition to the memory used by $\mathcal{A}$ and is nearly as efficient as $\mathcal{A}$.

Now, consider adversary $\mathcal{C}$ against the discrete logarithm problem in Figure 27. Observe that it simulates $\mathbb{G}_{12}$ perfectly to $\mathcal{A}$ and outputs v_2. Since $\mathbb{G}_{12}$ outputs true if and only if v_2 is the discrete logarithm of V, it follows that

$$\Pr[\mathbb{G}_{12} \Rightarrow \mathsf{true}] = \mathsf{Adv}^{\mathsf{DL}}_G(\mathcal{C}) \,. \tag{37}$$

Observe that $\mathcal{C}$ is nearly as efficient as $\mathcal{A}$. It requires memory to store $U, V, W, u, k, , v_{21}, v_{22}, V, q, q_1, q_2, q', \mathsf{p}_1, \mathsf{p}_2$, the current query and the return value. These can be stored in at most $15 \log p + \log |K_{F_1}| + 2\mathsf{hLen} + 4 \log q$ bits.

Combining (26) to (37) we have

$$\Pr[\mathbb{G}_2 \Rightarrow \mathsf{true}] \leqslant \mathsf{Adv}^{\mathsf{DL}}_G(\mathcal{B}) + q^2\mathsf{Adv}^{\mathsf{DL}}_G(\mathcal{C}) + \mathsf{Adv}^{\mathsf{PRF}}_{F_1}(\mathcal{E}) + q^2\mathsf{Adv}^{\mathsf{PRF}}_{F_1}(\mathcal{F}) \,. \tag{38}$$

Now that we have upper bounded $\Pr[\mathbb{G}_2 \Rightarrow \mathsf{true}]$, we need to upper bound $\Pr[\mathbb{G}_3 \Rightarrow \mathsf{true}]$ to get an upper bound on $\mathsf{Adv}^{\mathsf{ODH\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A})$. Here, again we shall construct an adversary against the discrete logarithm problem.

We introduce $\mathbb{G}_{13}$ in Figure 29 next. It is easy to verify that it is identical to $\mathbb{G}_3$ ($\mathbb{G}_{13}$ replaces the random oracle H with a random function $\mathsf{RF}_{G,\{0,1\}^{\mathsf{hLen}}}$ and changes a couple of equality checks to a different but equivalent form). Hence we have

$$\Pr[\mathbb{G}_{13} \Rightarrow \mathsf{true}] = \Pr[\mathbb{G}_3 \Rightarrow \mathsf{true}] \,. \tag{39}$$

Game $\mathbb{G}_{11}$, Game $\mathbb{G}_{12}$:

$u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
$v \leftarrow\$ \mathbb{Z}_p; V \leftarrow g^v$
$q_1 \leftarrow\$ [q]; q_2 \leftarrow\$ [q] \backslash q_1$
$\mathsf{p}_1 \leftarrow \perp, \mathsf{p}_2 \leftarrow \perp$
$k \leftarrow\$ K_{F_1}; v_2 \leftarrow 0$
foreach $X \in \mathbb{G}$ **do** $\mathsf{H}(X) \leftarrow \perp; \mathsf{m}_X \leftarrow \perp$
$W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
$q' \leftarrow 0$
$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.)}(g, U, V, W)$
$v_{21}, v_{22} \leftarrow$ solutions obtained from equating $\mathsf{p}_1, \mathsf{p}_2$
if $g^{v_{21}} = V$ **then** $v_2 \leftarrow v_{21}$
else $v_2 \leftarrow v_{22}$
return $(v_2 = v)$

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

$q' \leftarrow q' + 1$
if $V^u = X$ **then**
  return W
if $\mathsf{H}(X) \neq \perp \wedge \mathsf{m}_X \neq x_1 + ux_2 + x_3S$ **then**
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{m}_X \leftarrow x_1 + ux_2 + x_3S$
if $q' = q_1$ **then** $\mathsf{p}_1 \leftarrow x_1 + ux_2 + x_3S$
if $q' = q_2$ **then** $\mathsf{p}_2 \leftarrow x_1 + ux_2 + x_3S$
$\mathsf{H}(X) \leftarrow \mathsf{RF}_{\mathbb{Z}_p^3,\{0,1\}^{\mathsf{hLen}}}((x_1 + ux_2, x_3, 0))$
$\mathsf{H}(X) \leftarrow F_1(k, (x_1 + ux_2, x_3, 0))$
return $\mathsf{H}(X)$

Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$:

$q' \leftarrow q' + 1$
if $Y = U$ **then return** $\perp$
if $\mathsf{H}(Y^v) \neq \perp \wedge \mathsf{m}_{Y^v} \neq y_1S + uy_2S + y_3S^2$ **then**
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{m}_{Y^v} \leftarrow y_1S + uy_2S + y_3S^2$
if $q' = q_1$ **then** $\mathsf{p}_1 \leftarrow y_1S + uy_2S + y_3S^2$
if $q' = q_2$ **then** $\mathsf{p}_2 \leftarrow y_1S + uy_2S + y_3S^2$
$\mathsf{H}(Y^v) \leftarrow \mathsf{RF}_{\mathbb{Z}_p^3,\{0,1\}^{\mathsf{hLen}}}((0, y_1 + uy_2, y_3))$
$\mathsf{H}(Y^v) \leftarrow F_1(k, (0, y_1 + uy_2, y_3))$
return $\mathsf{H}(Y^v)$

Adversary $\mathcal{C}(g, V)$:

$u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
$q_1 \leftarrow\$ [q]; q_2 \leftarrow\$ [q]$
$\mathsf{p}_1 \leftarrow \perp, \mathsf{p}_2 \leftarrow \perp$
$W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
$q' \leftarrow 0$
$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.)}(g, U, V, W)$
$v_{21}, v_{22} \leftarrow$ solutions obtained from equating $\mathsf{p}_1, \mathsf{p}_2$
if $g^{v_{21}} = V$ **then** $v_2 \leftarrow v_{21}$
else $v_2 \leftarrow v_{22}$
return v_2

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

$q' \leftarrow q' + 1$
if $V^u = X$ **then**
  return W
if $q' = q_1$ **then** $\mathsf{p}_1 \leftarrow x_1 + ux_2 + x_3S$
if $q' = q_2$ **then** $\mathsf{p}_2 \leftarrow x_1 + ux_2 + x_3S$
return $F_1(k, (x_1 + ux_2, x_3, 0))$

Oracle $\mathsf{H_v}([Y]_{\vec{y}=y_1y_2y_3})$:

$q' \leftarrow q' + 1$
if $Y = U$ **then return** $\perp$
if $q' = q_1$ **then** $\mathsf{p}_1 \leftarrow y_1S + uy_2S + y_3S^2$
if $q' = q_2$ **then** $\mathsf{p}_2 \leftarrow y_1S + uy_2S + y_3S^2$
return $F_1(k, (0, y_1 + uy_2, y_3))$

Fig. 28. Game $\mathbb{G}_{11}$, adversary $\mathcal{C}$. The differences of $\mathbb{G}_{11}$ from $\mathbb{G}_9$ have been highlighted.

Next, we introduce game $\mathbb{G}_{14}$ where the only change from $\mathbb{G}_{13}$ is that the random function is replaced by a keyed function $F_2 : K_{F_2} \times G \rightarrow \{0,1\}^{\mathsf{hLen}}$.

Consider adversary $\mathcal{G}$ in Figure 29. It is easy to see that $\mathcal{G}$ simulates $\mathbb{G}_{13}$ to $\mathcal{A}$ when interacting with $\mathbb{G}_{F_2}^{\mathsf{PRF\text{-}RAND}}$ i.e. when it can query $O = \mathsf{RF}_{G,\{0,1\}^{\mathsf{hLen}}}(.)$, and returns 1 if and only $\mathbb{G}_{13}$ returns true. It simulates

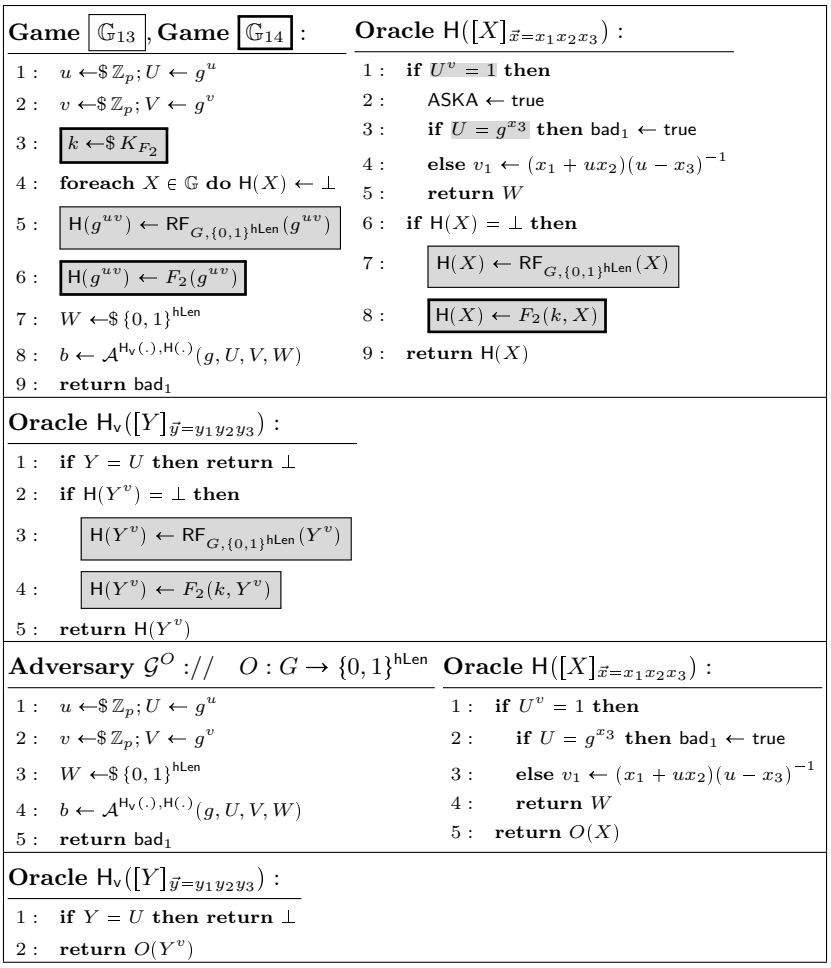

Fig. 29. Games $\mathbb{G}_{13}$, $\mathbb{G}_{14}$, adversary $\mathcal{G}$. The differences of $\mathbb{G}_{13}, \mathbb{G}_{14}$ from $\mathbb{G}_3$ have been highlighted. In $\mathbb{G}_{13}$, $\mathbb{G}_{14}$ the code in the thinner box is present only in $\mathbb{G}_{13}$ and the code in the thicker box is present only in $\mathbb{G}_{14}$.

$\mathbb{G}_{14}$ to $\mathcal{A}$ when interacting with $\mathbb{G}_{F_2}^{\mathsf{PRF\text{-}REAL}}$ i.e. when it can query $O = F_2(k, .)$, and returns 1 if and only $\mathbb{G}_{14}$ returns true. Therefore,

$$\Pr\left[\mathbb{G}_{13} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{14} \Rightarrow \mathsf{true}\right] + \mathsf{Adv}_{F_2}^{\mathsf{PRF}}(\mathcal{G}) \ . \tag{40}$$

It can be verified from the pseudocode that adversary $\mathcal{G}$ uses at most $8 \log p + \log |K_{F_2}| + 2\mathsf{hLen}$ bits of memory in addition to the memory used by $\mathcal{A}$ and is nearly as efficient as $\mathcal{A}$.

Next, we introduce game $\mathbb{G}_{15}$ that is identical to $\mathbb{G}_{14}$ in all respects except that it returns $(u_1 = u)$ instead of bad_1. It is easy to see that

$$\Pr\left[\mathbb{G}_{14} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathbb{G}_{15} \Rightarrow \mathsf{true}\right] \ . \tag{41}$$

Now, consider the adversary $\mathcal{D}$ against the discrete logarithm problem in Figure 30. Observe that it simulates $\mathbb{G}_{15}$ perfectly to $\mathcal{A}$ and outputs u_1. Since $\mathbb{G}_{15}$ outputs true if and only if u_1 is the discrete logarithm of U, it follows that

$$\Pr\left[\mathbb{G}_{15} \Rightarrow \mathsf{true}\right] = \mathsf{Adv}_{G}^{\mathsf{DL}}(\mathcal{D}) \ . \tag{42}$$

Observe that $\mathcal{D}$ is nearly as efficient as $\mathcal{A}$. It requires memory to store U, V, W, u, k, u_1, the current query and the return value. These can be stored in at most $8 \log p + \log |K_{F_2}| + 2\mathsf{hLen}$ bits. Combining (39) to (42) we get

$$\Pr\left[\mathbb{G}_3 \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Adv}_{G}^{\mathsf{DL}}(\mathcal{D}) + \mathsf{Adv}_{F_2}^{\mathsf{PRF}}(\mathcal{G}) \ .$$

Game $\mathbb{G}_{15}$:

1 : $u \leftarrow\$ \mathbb{Z}_p; U \leftarrow g^u$
2 : $v \leftarrow\$ \mathbb{Z}_p; V \leftarrow g^v$
3 : $u_1 \leftarrow 0; k \leftarrow\$ K_{F_2}$
4 : **foreach** $X \in \mathbb{G}$ **do** $\mathsf{H}(X) \leftarrow \perp$
5 : $W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
6 : $W \leftarrow \mathsf{H}(g^{uv})$
7 : $b \leftarrow \mathcal{A}^{\mathsf{H}_V(\cdot),\mathsf{H}(\cdot)}(g, U, V, W)$
8 : **return** $(u_1 = u)$

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

1 : **if** $U^v = 1$ **then**
2 : **if** $U = g^{x_3}$ **then**
3 : $\mathsf{bad}_1 \leftarrow \mathsf{true}$
4 : $u_1 \leftarrow x_3$
5 : **return** W
6 : **return** $F_2(k, X)$

Oracle $\mathsf{H}_V([Y]_{\vec{y}=y_1y_2y_3})$:

1 : **if** $Y = U$ **then return** $\perp$
2 : **return** $F_2(k, Y^v)$

Adversary $\mathcal{D}(g, U)$:

1 : $v \leftarrow\$ \mathbb{Z}_p; V \leftarrow g^v$
2 : $k \leftarrow\$ K_{F_2}$
3 : $W \leftarrow\$ \{0,1\}^{\mathsf{hLen}}$
4 : $b \leftarrow \mathcal{A}^{\mathsf{H}_V(\cdot),\mathsf{H}(\cdot)}(g, U, V, W)$
5 : **return** u_1

Oracle $\mathsf{H}([X]_{\vec{x}=x_1x_2x_3})$:

1 : **if** $U^v = 1$ **then**
2 : **if** $U = g^{x_3}$ **then**
3 : $u_1 \leftarrow x_3$
4 : **return** W
5 : **return** $F_2(k, X)$

Oracle $\mathsf{H}_V([Y]_{\vec{y}=y_1y_2y_3})$:

1 : **if** $Y = U$ **then return** $\perp$
2 : **return** $F_2(k, Y^v)$

Fig. 30. Game $\mathbb{G}_{15}$, adversary $\mathcal{D}$. The differences of $\mathbb{G}_{15}$ from $\mathbb{G}_{14}$ have been highlighted.

Combining this with (25) and (38) we have

$$\mathsf{Adv}^{\mathsf{ODH\text{-}AGM}}_{G,\mathsf{hLen}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{DL}}_{G}(\mathcal{B}) + q^2\mathsf{Adv}^{\mathsf{DL}}_{G}(\mathcal{C}) + \mathsf{Adv}^{\mathsf{DL}}_{G}(\mathcal{D}) + \mathsf{Adv}^{\mathsf{PRF}}_{F_1}(\mathcal{E}) + q^2\mathsf{Adv}^{\mathsf{PRF}}_{F_1}(\mathcal{F}) + \mathsf{Adv}^{\mathsf{PRF}}_{F_2}(\mathcal{G}) \,.$$

□

B Proof Sketch of Correctness of Decoding in Section 4.5

We shall argue that Decode produces the correct σ whenever it receives a valid encoding as input. So we assume throughout that the input to Decode is a valid encoding.

We shall show that T is identically populated in Encode, Decode and the answers to the queries of $\mathcal{R}_2, \mathcal{R}_1$ are identical in Encode, Decode. This would mean that the runs of $\mathcal{R}_2, \mathcal{R}_1$ would be identical in Encode, Decode since answers to every query is identical in Encode, Decode. In Encode, after running $\mathcal{R}_1$, note that T does not have any placeholder variables remaining because all the elements were $\mathcal{Z}$ were answers to $\mathcal{R}_1$ and would have had the placeholder variables for their pre-images removed by the time $\mathcal{R}_1$ finishes. So, all the tuples of T are triples of labels, pre-images and the flag after running $\mathcal{R}_1$. Since Encode returns a valid encoding (i.e. the check $|\mathcal{Z}| \geqslant l$ succeeds), the labels in RLbls are the ones absent in T appended in the lexicographical order of their pre-images. Now we shall use the fact that T is identical in Decode after running $\mathcal{R}_1$ i.e. ll the tuples of T are triples of labels, pre-images and the flag after running $\mathcal{R}_1$. Note that in Decode, the labels in T are correctly assigned to their pre-images and all unassigned pre-images in $\mathbb{Z}_p$ are assigned the labels in RLbls sequentially i.e. for all $x \in \mathbb{Z}_p$, $\sigma(x)$ is recovered correctly. Therefore Decode produces the correct σ. So all we need to show now is that T is identically populated in Encode, Decode and answers to every query is identical in Encode, Decode.

T IS IDENTICALLY POPULATED AND QUERIES ARE ANSWERED IDENTICALLY IN Encode, Decode. In Encode, since T is created after $\mathcal{R}_1$ is run for the first time, T is not affected by the first run of $\mathcal{R}_1$. Now, observe that the first $k+2$ tuples appended to T are $(\sigma(1), 1, 1), (\sigma(v), v, 1), (\sigma(i_1 \cdot v), i_1 \cdot v, 1), \cdots, (\sigma(i_k \cdot v), i_k \cdot v, 1)$ in

Encode. It follows that $\sigma(1), \sigma(v), \cdots, \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)$ are the first $k+2$ labels appended to Lbls. This means that in Decode, $\sigma(1), \sigma(v)$, $\sigma(i_j \cdot v)$'s are correctly assigned using the first $k+2$ labels in Lbls and the first $k+2$ tuples in T are the same as that in Encode. This means the first $k+2$ tuples are identically populated in T in Encode, Decode. Next, we sketch a proof that T is identically populated in Encode, Decode on every query made by $\mathcal{R}_2, \mathcal{R}_1$. We shall also show that the answers to the queries are identical in Encode, Decode. Note that T is modified only by the append and VarReduce operations. We shall be using the phrase "a label $\boldsymbol{\ell}$ is not in T" to mean $(\boldsymbol{\ell}, *, *) \notin \mathsf{T}$ throughout this proof sketch.

We shall first show that T is identical in Encode, Decode after an $\mathsf{Eval}(.,.,2)$ query if it was identical in Encode, Decode before the query and that the answer to the query is identical in Encode, Decode. We can prove a similar statement about $\mathsf{O_v}(.,.,2)$ queries analogously. We shall then show T is identical in Encode, Decode after before $\mathcal{R}_1$ is run in both of them for the last time. Then we shall that T is identical in Encode, Decode after an $\mathsf{Eval}(.,.,3)$ or $\mathsf{O_v}(.,.,3)$ query if it was identical in Encode, Decode before the query and that the answers to the queries are identical in Encode, Decode. This would conclude our proof that T is identically populated in Encode, Decode and that the queries of $\mathcal{R}_2$ and $\mathcal{R}_1$ are answered identically.

Assuming that T is identical in Encode, Decode before an $\mathsf{Eval}(.,.,2)$ query, we show that T is identical after the query in Encode, Decode and the answer to the query is identical in Encode, Decode. In Encode, when an $\mathsf{Eval}(.,.,2)$ query (assume the query sequence number is c_1) is made by $\mathcal{R}_2$, AddToTable is invoked on the input labels if they are not present in T (the input label not being in $\mathcal{Y}_2$ is equivalent to it not being in T). Suppose $\boldsymbol{\ell}$ is such an input label that is not present in T in Encode- this also means that $\boldsymbol{\ell}$ is not present in T in Decode from our assumption. Therefore, $\mathsf{AddToTable}(\boldsymbol{\ell}, c, b)$ would be invoked in both Encode, Decode. There are three possibilities.

1. In Encode, $\boldsymbol{\ell} \in \mathcal{X}$ and for some $j \in [k]$, $\boldsymbol{\ell} = \sigma(i_j)$. In this case, observe that (c, b, j) is added to Inputs. Since, this is the only point where tuples of the form $(*, 1, *), (*, 2, *)$ are added to Inputs in Encode, the equivalent condition in Decode is that there exists some $j \in [k]$ such that $(c, b, j) \in$ Inputs. Note that the same tuple is appended to T in Encode, Decode when these equivalent conditions are true.
2. In Encode, $\boldsymbol{\ell} \in \mathcal{X}$ and for no $j \in [k]$, $\boldsymbol{\ell} = \sigma(i_j)$. In this case, observe that (c, b) is added to Free. Since, this is the only point where tuples are added to Free in Encode, the equivalent condition in Decode is $(c, b) \in$ Free. Again, note that the same tuple is appended to T in Encode, Decode when these equivalent conditions are true.
3. In Encode, neither of the previous two conditions are true. In Decode, note that when AddToTable was invoked, $\boldsymbol{\ell}$ was not present in T and if either of the preceding two conditions were true, then a tuple containing $\boldsymbol{\ell}$ was appended to T. Therefore, $\boldsymbol{\ell}$ is not in T if and only if neither of the preceding conditions are true. Since the two preceding conditions are equivalent in Encode, Decode, the complement of the disjunction of the two conditions are also equivalent in Encode, Decode. Since the tuple $(\boldsymbol{\ell}, \sigma^{-1}(\boldsymbol{\ell}), 2)$ is added to T in Encode, it is easy to see that $\sigma^{-1}(\boldsymbol{\ell})$ is the element popped out to the list Vals. Therefore, again the same tuple is appended to T in Encode, Decode.

Therefore, after the invocation of AddToTable function, T is identical in Encode, Decode.

Note that the condition that $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ in Encode implies that there does not exist $\mathbf{c}'$ such that $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$. Also a tuple of the form $(c_1, *)$ is added to Srps_1 in Encode if and only if $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$. Therefore, the condition $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ is equivalent to the condition that there does not exist $\mathbf{c}'$ such that $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ and there is some i such that $(c_1, i) \in \mathsf{Srps}_1$ in Decode. Since T is identical up until this point, the operation VarReduce is identical in Encode, Decode.

The only other point when tuples are appended to T in Encode on an $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 2)$ query is when a tuple containing $\mathbf{c}$ (the answer of the Eval query) is added. Note that a tuple containing $\mathbf{c}$ is added to T in Encode if $\mathbf{c}$ is not in T. If $\mathbf{c}$ is not in T, then $(*, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \notin \mathsf{T}$ which in turn implies that no tuple of the form of $(c_1, *)$ was added to Srps_1. Therefore the equivalent condition in Decode is that for no $\mathbf{c}'$, $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ and for no i, $(c_1, i) \in \mathsf{Srps}_1$. It is easy to see that $\mathbf{c}$ is the label popped from the list Lbls in Decode when the condition is true. Therefore, again, the same tuple is appended to T in Encode, Decode. Also, the same label is the answer to the Eval query in Encode, Decode. Again, the condition $\mathbf{c}$ is not in T and

for some $j \in [k]$, $\mathbf{c} = \sigma(i_j)$ in Encode is equivalent to the condition that for no $\mathbf{c}'$, $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ and for no i, $(c_1, i) \in \mathsf{Srps}_1$ and for some $j \in [k]$, $(c_1, 3, j) \in \mathsf{Inputs}$ because a tuple of the form $(c_1, 3, *)$ is added to Inputs if and only if $\mathbf{c}$ is not in T and for some $j \in [k] : \mathbf{c} = \sigma(i_j)$ in Encode. Therefore, the $\mathsf{VarReduce}$ operation is identical in $\mathsf{Encode}, \mathsf{Decode}$. Also, in this case the same label $\sigma(i_j)$ is returned in $\mathsf{Encode}, \mathsf{Decode}$. In Encode, the condition $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) = \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ is equivalent to the condition there exists $\mathbf{c}'$ such that $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$. Since, T is identical up until this point in $\mathsf{Encode}, \mathsf{Decode}$, the condition $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) = \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ in Encode is equivalent condition $(\mathbf{c}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ in Decode. Note that for this equivalent condition, no changes are made to T in $\mathsf{Encode}, \mathsf{Decode}$ and the same label is returned in both procedures.

Thus, we have argued that the changes to T and the answer is identical in Encode, Decode on an $\mathsf{Eval}(.,.,2)$ query. Therefore, if T was identical in $\mathsf{Encode}, \mathsf{Decode}$ before an $\mathsf{Eval}(.,.,2)$ query, it remains identical after the query too and the answer of the $\mathsf{Eval}(.,.,2)$ query is identical in $\mathsf{Encode}, \mathsf{Decode}$.

Using very similar arguments we can prove that if T was identical in Encode, Decode before an $\mathsf{O_v}(.,.,2)$ query, it remains identical after the query too and the answer of the $\mathsf{O_v}(.,.,2)$ query is identical in $\mathsf{Encode}, \mathsf{Decode}$. We omit the proof since it is very similar to the previous one.

Since T was identical before running $\mathcal{R}_2$ in $\mathsf{Encode}, \mathsf{Decode}$, it remains identical after running $\mathcal{R}_2$ since the changes made on each $\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)$ query are identical in $\mathsf{Encode}, \mathsf{Decode}$ and all the answers to queries of $\mathcal{R}_2$ are identical in $\mathsf{Encode}, \mathsf{Decode}$. In Encode, the condition $\sigma(i_j)$ is not in T is equivalent to the condition $\sigma(i_j) = \perp$ in Decode because whenever a tuple of containing $\sigma(i_j)$ was appended to T in Decode, the value of $\sigma(i_j)$ was set and for $\sigma(i_j)$'s not in T, the value of $\sigma(i_j)$ is $\perp$. Therefore, T remains identical in $\mathsf{Encode}, \mathsf{Decode}$ till before running $\mathcal{R}_1$ in $\mathsf{Encode}, \mathsf{Decode}$.

Just like we argued that T remains identical after running $\mathcal{R}_2$ and all the answers to queries of $\mathcal{R}_2$ are identical in $\mathsf{Encode}, \mathsf{Decode}$, we can use very similar arguments to prove the T remains identical after running $\mathcal{R}_1$ in $\mathsf{Encode}, \mathsf{Decode}$ and all the answers to queries of $\mathcal{R}_1$ are identical in Encode, Decode. We omit the proof since it is very similar to the proof before.

There are no other changes to T after running $\mathcal{R}_1$ in $\mathsf{Encode}, \mathsf{Decode}$. Therefore, T is identical in $\mathsf{Encode}, \mathsf{Decode}$ and all the answers to queries of $\mathcal{R}_2, \mathcal{R}_1$ are identical in $\mathsf{Encode}, \mathsf{Decode}$.

C Rewinding: Conjecture and Obstacles

We briefly discuss the barriers in extending our result to consider rewinding.

A REDUCTION. First off, we informally describe a reduction $\mathcal{R}$ that *does* rewind the adversary $\mathcal{A}$ and answers all the $\mathsf{H}, \mathsf{H_v}$ queries using $O(k \log k + \log p)$ bits of memory where k is the total number of H and $\mathsf{H_v}$ query $\mathcal{A}$ makes. To simulate the random oracle in a memory-efficient way, the reduction $\mathcal{R}$ uses a PRF with key k_f whose domain is $\mathcal{L}$ and range is $\{0,1\}^{\mathsf{hLen}}$ and a list L. It keeps a counter on the number of queries by $\mathcal{A}$ and increments it by 1 on each H and $\mathsf{H_v}$ query. The reduction $\mathcal{R}$ samples a key k_f (assume k_f can be expressed in r bits) for the PRF and initializes L to empty before running $\mathcal{A}$.

For the first query made by $\mathcal{A}$ (H or $\mathsf{H_v}$), $\mathcal{R}$ responds with $\mathsf{PRF}(k_f, 1)$ and appends 1 to L. If the i^{th} query by $\mathcal{A}$ is a $\mathsf{H_v}$ query on $\mathbf{a}$, $\mathcal{R}$ records $\mathbf{a}$, rewinds $\mathcal{A}$ to its beginning, and checks if $\mathcal{A}$ had made a $\mathsf{H_v}$ query on $\mathbf{a}$ before the i^{th} query or if $\mathcal{A}$ had made a H query on $\mathbf{b}$ such that $\mathsf{O_v}(\mathbf{a}, \mathbf{b}) = 1$ before the i^{th} query. If $\mathcal{A}$ does find either of these to be true on the j^{th} query, $\mathcal{R}$ appends $L[j]$ to L. Similarly, if the i^{th} query by $\mathcal{A}$ is a H query on $\mathbf{a}$, $\mathcal{R}$ records $\mathbf{a}$, rewinds $\mathcal{A}$ to its beginning, and checks if $\mathcal{A}$ had made a H query on $\mathbf{a}$ before the i^{th} query or if $\mathcal{A}$ had made a $\mathsf{H_v}$ query on $\mathbf{b}$ such that $\mathsf{O_v}(\mathbf{b}, \mathbf{a}) = 1$ before the i^{th} query. If $\mathcal{A}$ does find either of these to be true on the j^{th} query, $\mathcal{R}$ appends $L[j]$ to L. While rewinding, $\mathcal{R}$ answers the t^{th} query ($t < i$) by $\mathsf{PRF}(k_f, L[t])$. Clearly, using this strategy, $\mathcal{R}$ simulates the answers to the queries of $\mathcal{A}$ correctly. Note that k_f can be expressed in r bits, L can be expressed in at most $k \log k$ bits, $\mathbf{a}$ can be expressed in at most $\log p$ bits and the query sequence number can be expressed in at most $\log k$ bits. Therefore, $\mathcal{R}$ uses at most $r + k \log k + \log p + \log k = O(k \log k + \log p)$ bits of memory.

REMARKS. Note that even if $\mathcal{R}$ rewinds $\mathcal{A}$, $\mathcal{R}$ needs to answer the queries of $\mathcal{A}$ from its beginning to the point where $\mathcal{R}$ rewound it, again. Therefore, it seems unavoidable that $\mathcal{R}$ needs to remember some information about each query of $\mathcal{A}$. Since there are k queries, this information would take at least $\log k$ bits for each

query (since answer to each query could be unique) and $k \log k$ bits in total. Hence, we conjecture a lower bound of $\Omega(k \log k)$ bits even when $\mathcal{R}$ rewinds $\mathcal{A}$.

We are not aware of any techniques that could help us establish this bound, and this appears a much harder problem than the straightline case.