On Non-uniform Security for Black-box Non-Interactive CCA Commitments

Rachit Garg* Dakshita Khurana† George Lu* Brent Waters*

August 9, 2022

Abstract

We obtain a black-box construction of non-interactive CCA commitments against non-uniform adversaries. This makes black-box use of an appropriate base commitment scheme for small tag spaces, variants of sub-exponential hinting PRG (Koppula and Waters, Crypto 2019) and variants of keyless sub-exponentially collision-resistant hash function with security against non-uniform adversaries (Bitansky, Kalai and Paneth, STOC 2018 and Bitansky and Lin, TCC 2018).

All prior works on non-interactive non-malleable or CCA commitments without setup first construct a "base" scheme for a relatively small identity/tag space, and then build a tag amplification compiler to obtain commitments for an exponential-sized space of identities. Prior black-box constructions either add multiple rounds of interaction (Goyal, Lee, Ostrovsky and Visconti, FOCS 2012) or only achieve security against uniform adversaries (Garg, Khurana, Lu and Waters, Eurocrypt 2021).

Our key technical contribution is a novel tag amplification compiler for CCA commitments that replaces the non-interactive proof of consistency required in prior work. Our construction satisfies the strongest known definition of non-malleability, i.e., CCA2 (chosen commitment attack) security. In addition to only making black-box use of the base scheme, our construction replaces sub-exponential NIWIs with sub-exponential hinting PRGs, which can be obtained based on assumptions such as (sub-exponential) CDH or LWE.

1 Introduction

Non-malleable commitments [DDN91] and their stronger counterparts CCA commitments [CLP10] are core cryptographic primitives that provide security in the presence of "man in the middle" attacks. They ensure that a man-in-the-middle adversary, that simultaneously participates in two or more protocol sessions, cannot use information obtained in one session to breach security in another. They also enable secure multi-party computation, coin flipping and auctions.

This work builds non-interactive CCA commitments, which involve just a single commit message from the committer. We focus on the (standard) notion of security against non-uniform adversaries, which necessitates that these commitments be perfectly binding and computationally hiding. For these commitments, the perfect binding requirement is that for any commitment string c generated maliciously with potentially an arbitrary amount of preprocessing, there do not

*UT Austin. Email:{rachg96, bwaters, gclu}@cs.utexas.edu
†UIUC. Email: dakshita@illinois.edu.

exist two openings to messages m and m' such that $m \neq m'$. The (computational) hiding property requires that for every pair of equal-length messages m and m', the distributions of commitments $\mathsf{com}(m)$ and $\mathsf{com}(m')$ are computationally indistinguishable.

The notion of CCA security for commitments is defined analogously to encryption schemes, except that the adversary is given access to a decommitment oracle. However, unlike the case of encryption, non-interactive commitments without setup do not allow for efficient decommitment given a trapdoor/secret key. In more detail, the hiding game is strengthened significantly to give the adversary oracle access to an *inefficient* decommitment/value function $\mathsf{CCA.Val}$ where on input a string c, $\mathsf{CCA.Val}(\mathsf{tag}, c)$ will return m if $\mathsf{CCA.Com}(\mathsf{tag}, m; r) \to c$ for some r. The adversary must first specify a challenge tag tag^*, along with messages m_0^*, m_1^*. It is then allowed oracle access to $\mathsf{CCA.Val}(\mathsf{tag}, \cdot)$ for every $\mathsf{tag} \neq \mathsf{tag}^*$, and can make an arbitrary (polynomial) number of queries before and after obtaining the challenge commitment. [1]

This CCA-based definition is the strongest known definition of non-malleability. In the non-interactive setting, the often-used definition of (concurrent) non-malleability with respect to commitment is a special case of this definition where the adversay is only allowed to make parallel oracle queries once it obtains the challenge commitment.

Prior Work on Non-Malleable Commitments. There have been several results [DDN91, Bar02, PR05, PR08, LPV, PPV08, LP09, Wee10, PW10, LP, Goy11, GLOV12, GRRV14, GPR16, COSV16, COSV17, Khu17, LPS17, KS17, GR19] that gradually reduced the round complexity and the cryptographic assumptions required to achieve non-malleable commitments. In the non-interactive setting, Pandey, Pass and Vaikuntanathan [PPV08] first obtained non-malleable commitments from a strong non-falsifiable assumption. A lower bound due to Pass [Pas13] demonstrated the difficulty of obtaining a non-interactive construction from standard assumptions.

Nevertheless, recent works of Lin, Pass and Soni [LPS17], Bitansky and Lin [BL18b], Kalai and Khurana [KK19], Garg et al. [GKLW21] and Khurana [Khu21] made progress towards improving these assumptions. These works proceed in two steps: the first step builds a "base" scheme supporting a small (typically, constant-sized) tag space and the second step converts commitments supporting a small tag space to commitments that support a much larger tag space.

Base Constructions. Three recent works [LPS17, BL18b, KK19] build non-interactive base schemes: non-malleable commitments for a tag space of size $c \log \log \kappa$ for a specific constant $c > 0$, based on various hardness assumptions. Specifically, Lin, Pass and Soni [LPS17] assume a sub-exponential variant of the hardness of time-lock puzzles, and Bitansky and Lin [BL18b] rely on sub-exponentially hard injective one-way functions that admit hardness amplification beyond negligible. Finally, Kalai and Khurana [KK19] assume classically sub-exponentially hard but quantum easy non-interactive commitments (which can be based, e.g., on sub-exponential hardness of DDH), and sub-exponentially quantum hard non-interactive commitments (which can be based, e.g., on sub-exponential hardness of LWE).

Tag Amplification. The second step, as discussed above, builds a tag amplficiation compiler that increases the tag space exponentially. Starting with non-malleable commitments for a tag space of

[1] The assumption that the commitment takes input a tag is without loss of generality when the tag space is exponential. As is standard with non-malleable commitments, tags can be generically removed by setting the tag as the verification key of a signature scheme, and signing the commitment string using the signing key.

size $c \log \log \kappa$ for a specific constant $c > 0$ (or sometimes even smaller), multiple applications of this compiler yield commitments for a tag space of size 2^κ.

This step, which is also the focus of the current work, typically involves encoding a single tag from a larger space into many tags from a smaller space, and then committing to a given message several times, once w.r.t. each small tag. In addition, an implicit/explict *proof of consistency* of these commitments is provided, and this proof is required to hide the committed message. Such a proof becomes challenging to implement in the non-interactive setting without setup.

Nevertheless, tag amplification was obtained in [LPS17] against *uniform* man-in-the-middle adversaries based on sub-exponential non-interactive witness indistinguishable (NIWI) proofs and keyless collision resistant hash functions against uniform adversaries. It was also obtained in [BL18b] against *non-uniform* man-in-the-middle adversaries based on sub-exponential non-interactive witness indistinguishable (NIWI) proofs and keyless collision resistant hash functions with a form of collision resistance even against *non-uniform* adversaries. Somewhat orthogonally, [Khu21] obtained tag amplification from sub-exponential indistinguishability obfuscation and sub-exponential one-way functions, while avoiding the need for keyless collision resistant hashing.

Black-box Tag Amplification. Recently, [GKLW21] developed the first tag amplification technique that only made *black-box use of the base commitment*. That work additionally assumed (black-box access to) hinting PRGs and keyless collision resistant hash functions against uniform adversaries. Hinting PRGs themselves admit constructions from the CDH and LWE assumptions. Besides being black-box , this was the first solution that *did not* rely on non-interactive witness indistinguishable (NIWI) proofs, which so far are only known based on the hardness of the decisional linear problem over bilinear maps [GOS12], or derandomization assumptions and trapdoor permutations [BOV07], or indistinguishability obfuscation and one-way functions [BP15]. However, GKLW only obtain security against uniform adversaries.

But non-uniform security is often necessary when using non-malleable commitments within a bigger protocol. For instance, round efficient secure multi-party computation protocols in the plain model [BHP17, ACJ17, HHPV18, BL18a, BGJ$^+$18, CCG$^+$21] against malicious adversaries usually include a step where participants commit to their inputs via a non-malleable/CCA commitment, in addition to providing a proof that the CCA commitment is consistent with other messages sent in the protocol. In low-interaction settings such as those of super-polynomial secure MPC in two or three [BGJ$^+$17] messages, these proofs of consistency are often simulated non-uniformly, which ends up necessitating the use of non-malleable commitments with security against non-uniform adversaries.

Our work addresses the following natural gap in our understanding of non-interactive non-malleable/CCA commitments.

Is it possible to obtain black-box non-interactive CCA commitments against non-uniform adversaries?

Our Results. This work provides a *black-box approach* to achieving non-interactive CCA commitments with security against *non-uniform adversaries*, by relying on keyless hash functions that satisfy collision-resistance against non-uniform adversaries, and by overcoming seemingly fundamental limitations from the prior work of [GKLW21]. In addition, our tag amplification technique achieves provable security without the need for NIWIs as in prior work [BL18b], and by instead

relying on a sub-exponentially secure variant of hinting PRGs, which can themselves be obtained from (sub-exponential) CDH or LWE just like their counterparts in [KW19].

2 Overview of Techniques

We now give an overview of our amplification technique, where the goal is to amplify a scheme for $O(N)$ tags to a scheme for 2^N tags, with computational cost that grows polynomially with N and the security parameter κ. This process can be applied iteratively $c+1$ times to a base NM commitment scheme that handles tags of size $\underbrace{\lg \lg \cdots \lg}_{c \text{ times}}(\kappa)$ for some constant c and results in a scheme that handles tags of size 2^κ.

Templates for Tag Amplification. To perform tag amplification, we will build on a tag encoding scheme that was first suggested by [DDN91]. They suggest a method of breaking a large tag T^j (say, in $[2^N]$) into N small tags $t_1^j, t_2^j, \ldots t_N^j$, each in $2N$, such that for two different large tags $T^1 \neq T^2$, there exists at least one index i such that $t_i^2 \notin \{t_1^1, t_2^1, \ldots t_N^1\}$. This is achieved by setting $t_i^j = i||T^j[i]$, where $T^j[i]$ denotes the i^{th} bit of T^j.

Given this tag amplification technique, we start by describing a template for non-interactive tag amplification suggested in [KS17, LPS17]. A CCA commitment scheme for tags in 2^N will generate a commitment to a message m as $\mathsf{CCA.Com}(1^\kappa, \mathsf{tag}, m; r) \to \mathsf{com}$. The string com is generated by first applying the DDN encoding to tag to obtain N tags $t_1, \ldots t_N$. Next, these (smaller) tags are used to generate commitments to m in the smaller tag scheme as $c_i = \mathsf{Small.Com}(1^\kappa, (t_i), \mathsf{msg} = m; r_i)$ for $i \in [N]$. The intuition for security is as follows: recall that the DDN encoding ensures that for two different large tags $T^1 \neq T^2$, there exists at least one index i such that $t_i^2 \notin \{t_1^1, t_2^1, \ldots t_N^1\}$. This (roughly) implies that the commitment generated by an adversary w.r.t. tag t_i^2 is *independent* of the challenge commitment string, as we desire. However, the commitments w.r.t. other tags t_j^2 could potentially depend on the challenge commitment, which is undesirable. To get around this issue, the templates in [KS17, LPS17][2] suggest that the committer attach a type of zero knowledge (ZK) proof that all commitments are to the same message m using the random coins as a witness. In the setting of non-interactive amplification, the ZK proof will need to be non-interactive. For technical reasons, it is in fact required to be ZK against adversaries running in time T, where T is the time required to brute-force break the underlying CCA scheme for small tags.

Since non-interactive ZK proofs do not exist without trusted setup, the techniques in [LPS17, KS17, BL18b, KK19] rely on weaker variants of ZK such as NIWIs, and [LPS17, KS17, BL18b] combine NIWIs with a trapdoor statement that an (inefficient) ZK simulator uses to simulate the ZK proof. At the same time, for soundness, we require that an adversary cannot use the trapdoor statement to cheat. This is challenging when the trapdoor statement is fixed independently of the statement being proven, because a *non-uniform* adversary can always hardwire the trapdoor and use this to provide convincing proofs of false statements.

Given this barrier, [LPS17] restricted themselves to achieving tag amplification against *uniform adversaries*, based on (sub-exponential) NIWIs and keyless collision-resistant hash functions against uniform adversaries. Subsequently [BL18b] developed a technique to obtain tag amplification against *non-uniform adversaries*, based on NIWIs and assuming the existence of keyless

[2] These are the non-interactive versions of templates previously suggested in [DDN91, LP09, Wee10].

collision-resistant hash functions that satisfy some form of security against non-uniform adversaries. Very roughly, they assume that no adversary with non-uniform advice of size S can find more than $\mathsf{poly}(S)$ collisions[3].

More recently, [GKLW21] developed a method for performing non-interactive tag amplification without NIWIs, and while only making *black-box use* of the underlying base commitment. However, the resulting scheme is secure only against *uniform adversaries*. On the other hand, the goal of this work is to achieve a *black-box construction* that avoids NIWIs and achieves security against *non-uniform adversaries*, under a similar keyless assumption as [BL18b]. To highlight the bottlenecks in the non-uniform setting, we give a brief overview of the technique of [GKLW21].

Black-box Tag Amplification. To begin, we note that the tag amplification technique sketched above is not black-box in the base commitment due to the use of variants of ZK. Recall that ZK is used to ensure consistency of adversarial commitments generated w.r.t. different small tags. In the CCA setting, this allows using a CCA decommitment oracle that opens a commitment under any one of the adversary's small tags, without the adversary noticing which one was opened. In other words, ZK is used to establish a system where the adversary cannot submit a commitment such that its opening will be different under oracle functions that open different commitments, which turns out to be crucial to achieving CCA security.

In [GKLW21], this system is established by means of a *hinting PRG* [KW19]. At a high level, the construction in [GKLW21] sets things up so that the CCA oracle that opens a commitment under one of the adversary's small tags will recover a candidate PRG seed s. This seed deterministically generates (a significant part of) the randomness used to create commitments with respect to *all* the adversary's small tags. The oracle uses this property to check for consistency by re-evaluating the underlying small-tag commitments, and checking them against the original. These checks intuitively serve as a substitute for ZK proofs, however they differ from ZK in that the checking algorithm sometimes allows partially malformed commitments to be opened to valid values. While creating such partially malformed commitments is actually easy for the adversary, the adversary is still unable to distinguish between oracles that open different small tag commitments.

The work [GKLW21] converts CCA commitments with $4N$ tags to CCA commitments with 2^N tags, assuming hinting PRGs and statistically equivocal commitments without setup, that satisfy binding against uniform adversaries. A hinting PRG satisfies the following property: for a uniformly random short seed s, expand $PRG(s) = z_0 z_1 z_2 \dots z_n$. Then compute matrix x by sampling uniformly random $v_1 v_2 \dots v_n$, and setting for all $i \in [n]$, $M_{s_i,i} = z_i$ and $M_{1-s_i,i} = v_i$. The requirement is that z_0, M generated using a uniformly random seed must be indistinguishable from a uniform random string.

Here, we actually note that prior works [KW19, GKLW21] can be made to work based on a hinting PRG that actually satisfies a weaker property: namely, that z_0, M obtained as described above should be indistinguishable from u, M where u is generated uniformly at random and M is generated as described above. Looking ahead, we will define a variant of a hinting PRG and will rely on the fact that this weaker property can be used instead.

Hinting PRGs were built based on CDH, LWE [KW19], as well as more efficient versions based on the ϕ-hiding and DBDHI assumptions [GVW20]. The required equivocal commitments can be

[3]Technically, they rely on a more general notion of incompressible problems, which is a collection of efficiently recognizable and sufficiently dense sets, one for each security parameter, for which no adversary with non-uniform description of polynomial size in S can find more than $K(S)$ elements in the set.

obtained from keyless collision resistant hash functions against uniform adversaries, based on the blueprint of [DPP93] and [HM96], and more recently [BKP18], in the keyless hash setting.

The [GKLW21] technique. We now provide a brief overview of the [GKLW21] technique, since their construction will serve as a starting point for our work.

Let $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ be a CCA commitment for $4N$ tags. Then [GKLW21] assume tags take identities of the form $(i, \beta, \gamma) \in [N] \times \{0,1\} \times \{0,1\}$ and that the $\mathsf{Small.Com}$ algorithm requires randomness of length $\ell(\kappa)$. Their transformation produces three algorithms, $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$. The $\mathsf{CCA.Com}$ algorithm on input a tag tag from the large tag space, an input message, and uniform randomness, first samples a seed s of size n for a hinting PRG. It uses the first co-ordinate z_0 (of the output of the hinting PRG on input s), as a one-time pad to mask the message m, resulting in string c. Next, it generates n equivocal commitments $\{\sigma_i\}_{i\in[n]}$, one to each bit of s. We will let y_i denote the opening of the i^{th} equivocal commitment (this includes the i^{th} bit s_i of s). Finally, it 'signals' each of the bits of s by generating commitments $\{c_{x,i,b}\}_{x\in[N],i\in[n],b\in\{0,1\}}$ using the small tag scheme. For every $i \in [n]$, the commitments $\{c_{x,i,0}\}_{x\in[N]}$ and $\{c_{x,i,1}\}_{x\in[N]}$ are generated as follows:

1. If $s_i = 0$
   (a) $c_{x,i,0} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 0), \mathsf{msg} = y_i; r_{x,i})$
   (b) $c_{x,i,1} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 1), \mathsf{msg} = y_i; \tilde{r}_{x,i})$
2. If $s_i = 1$
   (a) $c_{x,i,0} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 0), \mathsf{msg} = y_i; \tilde{r}_{x,i})$
   (b) $c_{x,i,1} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 1), \mathsf{msg} = y_i; r_{x,i})$

where all the $\tilde{r}_{x,i}$ values are uniformly random, whereas $r_{x,i}$ values correspond to the output of the hinting PRG on seed s. The output of $\mathsf{CCA.Com}$ is $\mathsf{tag}, c, \{\sigma_i\}_{i\in[n]}, \{c_{x,i,b}\}_{x\in[N],i\in[n],b\in\{0,1\}}$.

On an oracle query of the form $\mathsf{CCA.Val}(\mathsf{tag}, \mathsf{com})$, we must return the message committed in the string com, if one exists. To do this, we parse $\mathsf{com} = \mathsf{tag}, c, \{\sigma_i\}_{i\in[n]}, \{c_{x,i,b}\}_{x\in[N],i\in[n],b\in\{0,1\}}$, and then recover the values committed under small tags $(1, \mathsf{tag}_1, 0)$ and $(1, \mathsf{tag}_1, 1)$, which also helps recover the seed s of the hinting PRG. Next, we check that for every $i \in [n]$, the recovered values correspond to openings of the respective σ_i. We also compute hinting PRG(s), and use the resulting randomness to check that for all $x \in [N]$, the commitments that were supposed to use the outcome of the PRG were correctly constructed. If any of these checks fail, we know that the commitment string com cannot be a well-formed commitment to any message. Therefore, if any of the checks fail, the oracle outputs $\perp$. These checks are inspired by [KW19], and intuitively, ensure that it is computationally infeasible for an adversary to query the oracle on commitment strings that lead to different outcomes depending on which small tag was used. If all these checks pass, the $\mathsf{CCA.Val}$ algorithm uses c to recover and output m.

To prove that the resulting scheme is CCA secure against uniform adversaries, note that the set $\{(x, \mathsf{tag}_x)\}_{x\in[N]}$ is nothing but the DDN encoding of the tag tag. This means that for our particular method of generating the commitments $c_{x,i,b}$ described above, for each of the adversary's oracle queries, there will be an index $x' \in [N]$ such that the tags $(x', \mathsf{tag}_{x'}, 0)$ and $(x', \mathsf{tag}_{x'}, 1)$ used to generate $\{c_{x',i,b}\}_{i\in[n],b\in\{0,1\}}$ in that query will differ from *all small tags used to generate the challenge commitment*.

The first step towards proving security of the resulting commitment will be to define an alternative CCA.ValAlt algorithm, that instead of recovering the values committed under tags $(1, \mathsf{tag}_1, 0)$ and $(1, \mathsf{tag}_1, 1)$, recovers values committed under $(x', \mathsf{tag}_{x'}, 0)$ and $(x', \mathsf{tag}_{x'}, 1)$. The goal is to ensure that it is computationally infeasible for an adversary to query the oracle on commitment strings for which CCA.Val and CCA.ValAlt lead to different outcomes. In more detail, because of the checks performed by the valuation algorithms, it is possible to argue that any adversary that distinguishes CCA.Val from CCA.ValAlt must query the oracle with a commitment string that has following property: For some $i \in [n], x \in [N]$, $c_{x,i,0}$ and $c_{x,i,1}$ are small tag commitments to openings of the equivocal commitment to some bit b and $1 - b$ respectively. One can then brute-force extract these openings from $c_{x,i,0}$ and $c_{x,i,1}$ to contradict the binding property of the commitment against *uniform* sub-exponential adversaries.

This first step already becomes a bottleneck in the non-uniform setting: in general, an adversary with bounded polynomial advice can always sample an equivocal (non-interactive) commitment string together with an opening to 0 and another opening to 1.

The problem in the non-uniform case. As discussed above, the proof/construction in [GKLW21] falls apart in the very first step when considering a non-uniform adversary. In fact, such an adversary can attack the [GKLW21] scheme by non-uniformly sampling equivocal commitments $\{\widetilde{\sigma}_i\}_{i\in[n]}$ together with randomness $\{\widetilde{y}_{0,i}\}_{i\in[n]}$ and $\{\widetilde{y}_{1,i}\}_{i\in[n]}$ that can be used to open these commitments to both 0 and 1 respectively. Next, it can set the components $\{\widetilde{c}_{x,i,b}\}_{x\in[N],i\in[n],b\in\{0,1\}}$ as small-tag commitments to both types of openings. This allows the attacker to explicitly break CCA2 security, as we describe next.

Let $x' \in [N]$ be an index such that the tags $(x', \mathsf{tag}_{x'}, 0)$ and $(x', \mathsf{tag}_{x'}, 1)$ used to generate $\{c_{x',i,b}\}_{i\in[n],b\in\{0,1\}}$ in that query differ from *all small tags used to generate the challenge commitment*. On one hand, CCA2 security of the small-tag scheme will ensure that seed recovered from small-tag commitments $(x', \widetilde{\mathsf{tag}}_{x'}, 0)$ and $(x', \widetilde{\mathsf{tag}}_{x'}, 1)$ are independent of the seed in the challenge commitment. On the other hand, the actual committed value, which is defined via the seed recovered from $(1, \widetilde{\mathsf{tag}}_1, 0), (1, \widetilde{\mathsf{tag}}_1, 1)$ will exactly match the value in the challenge commitment, allowing this adversary to break CCA2 security. The equivocation described above would allow the adversary to ensure that all the hinting PRG checks pass, despite the use of different types of seeds in small tags $(1, \widetilde{\mathsf{tag}}_1, 0), (1, \widetilde{\mathsf{tag}}_1, 1)$ versus $(x', \widetilde{\mathsf{tag}}_{x'}, 0), (x', \widetilde{\mathsf{tag}}_{x'}, 1)$.

Towards a Solution. Now, one could hope to rely on some form of *non-uniform* security of keyless hash functions [BKP18, BL18b]. Prior works [BKP18, BL18b] have formulated and used the assumption that there exist keyless hash functions where any adversary with non-uniform advice of size S can only find $\mathsf{poly}(S)$ collisions. Inspired by a technique in [BL18b], we could hope to define a "bad" CCA2 query as one that contains openings to both a zero and a one for the equivocal commitment. Next, we could hope to limit the number of "bad" CCA2 queries that a non-uniform adversary will make to its decommitment oracle. As long as this set of "bad" queries is bounded and is just a function of the adversary's non-uniform advice, our challenger could also hope to non-uniformly obtain answers to such queries and use these instead of running the CCA.Val or CCA.ValAlt function.

Unfortunately, in the [GKLW21] protocol, even given just bounded (polynomial) non-uniform advice, an adversary will be able to equivocate *all of its* commitments and generate an *unbounded* number of bad queries. Moreover, because the hinting PRG is not injective, each bad query could

have multiple possible openings to different seeds. This indicates that the [GKLW21] protocol needs to be fundamentally modified to enable security against non-uniform attacks.

Our Approach. We begin by understanding how the [GKLW21] protocol can possibly be modified to disallow the attack described above.

- As described above, we want to force the adversary to "use up" bits of non-uniform advice for each new bad query that it makes. This will hopefully help limit the number of unique bad queries, and our reduction could then non-uniformly obtain answers to each of these queries.
- To allow the reduction to non-uniformly answer bad queries, we will aim to pair every possible bad query with a *unique* seed value that can be used to answer this bad query in place of running the $\mathsf{CCA.Val}$ or $\mathsf{CCA.ValAlt}$ function.

Limiting bad seeds instead of bad queries. The first bullet aims to *limit the number of bad queries*. While we will not be able to achieve this, we will achieve a slightly weaker property that will nevertheless suffice for our proof idea to go through. In more detail, we will tie every CCA2 query, and in particular the *equivocal commitment part of every CCA2 query* to an auxiliary input parameter. That is, in addition to message and randomness, each equivocal commitment will obtain as input an auxiliary parameter. There will be no hiding requirement on the auxiliary parameter; it will only serve to strengthen the binding property of the equivocal commitment. We will require that there exists a fixed polynomial $K(\cdot)$ such that any adversary with non-uniform advice of size S is unable to output $K(S)$ different pairs of auxiliary parameters and commitment strings, with valid openings for each pair to both a zero and a one. We will rely on keyless collision-resistant hash functions against non-uniform adversaries to build modified equivocal commitments with this guarantee. While this does not limit the number of bad queries that an adversary can make, it does limit the number of unique auxiliary input parameters that an adversary can use to generate CCA2 queries where it is able to open the equivocal commitments to both a zero and a one.

The goal of the second bullet is to allow a reduction to answer all bad queries by pairing every such query with a *unique seed* that can be used to non-uniformly answer this query in place of running the $\mathsf{CCA.Val}$ or $\mathsf{CCA.ValAlt}$ function. To get this idea to work, we must assign a "right" candidate seed to each bad query. As discussed above, in the [GKLW21] protocol, any adversary that can find two openings for the equivocal commitments could submit a bad query where multiple possible seed values match the output of the HPRG. To prevent this, we will explicitly force the HPRG to be injective. In more detail, we add what we call an"injective extension" to the HPRG. This is an additional algorithm $\mathsf{ExtEval}(s) \to \mathsf{r}_{\mathsf{ext}}$ that is an injective function on the HPRG seed s. The HPRG security requirement is also slightly modified to ensure that an adversary will not be able to distinguish the PRG output z from uniform given the hint matrix M (described above) and *additionally given* $\mathsf{r}_{\mathsf{ext}}$.

Now the CCA2 commitment will additionally consist of the value $\mathsf{r}_{\mathsf{ext}} = \mathsf{ExtEval}(s)$, and $\mathsf{CCA.Val}/\mathsf{CCA.ValAlt}$ will reject if for a recovered candidate seed s', $\mathsf{ExtEval}(s') \neq \mathsf{r}_{\mathsf{ext}}$. As a result, there will be at most a single seed s that will be "compatible" with any commitment string.

Going back to the construction of our CCA2 commitment, we will compute the modified equivocal commitments with auxiliary parameter set to $\mathsf{r}_{\mathsf{ext}}$, where recall that $\mathsf{r}_{\mathsf{ext}} = \mathsf{ExtEval}(s)$. At this point, we will be able to assign (at most) one unique 's' to each auxiliary parameter. Moreover,

by the (strengthened) binding property of equivocal commitments, any non-uniform attacker will be able to equivocate on at most a small number of auxiliary parameter values.

Analyzing Security. To prove CCA2 security of the resulting construction, we will proceed as follows. In the first hybrid (Game 1), we will switch to a challenger that depending on the adversary's non-uniform advice, stores a "cheat-sheet" consisting of all 'bad' $\mathsf{r}_{\mathsf{ext}}$ that the adversary can query on (with more than a certain inverse-polynomial probability), together with their inverses s under the injective algorithm $\mathsf{ExtEval}(\cdot)$. Our challenger will (1) rely on the cheat-sheet to answer any adversarial queries for which $\mathsf{r}_{\mathsf{ext}}$ lies on the cheat-sheet, and (2) use $\mathsf{CCA.Val}$ to decrypt only those queries for which $\mathsf{r}_{\mathsf{ext}}$ lies outside the cheat-sheet.

In the second hybrid (Game 2), the challenger will behave similarly as the previous hybrid, except using $\mathsf{CCA.ValAlt}$ to decrypt queries for which $\mathsf{r}_{\mathsf{ext}}$ lies outside the cheat-sheet. By the strong binding property of the equivocal commitment, the adversary is guaranteed to not equivocate on these queries (except with low probability). Therefore by the argument outlined in the proof of the [GKLW21] technique, the outputs of $\mathsf{CCA.Val}$ and $\mathsf{CCA.ValAlt}$ will be indistinguishable on these queries. The rest of the proof will follow similarly to [GKLW21]. There is one major hurdle in realizing this outline, as we discuss next.

Modifying the $\mathsf{CCA.Val}$ algorithm. The first hybrid (Game 1) described above will actually *not* be indistinguishable from the output of the actual CCA2 game. This is because a non-uniform adversary may generate equivocation queries for which $\mathsf{r}_{\mathsf{ext}}$ lies on the cheat-sheet and has an inverse (a hinting PRG seed), but the $\mathsf{CCA.Val}$ algorithm run by the CCA2 challenger may *not* be able to find this seed. To deal with this issue, we will change the $\mathsf{CCA.Val}$ algorithm so that it performs a brute-force search through all possible seeds to find the one (if any) that matches $\mathsf{r}_{\mathsf{ext}}$.

At first it appears that the rest of the proof should be easy once this is done. It should be possible to rely on security of the (1) auxiliary-input equivocal commitments and (2) hinting PRGs with injective extension, to show that the (updated) CCA2 game is indistinguishable from the first hybrid. However, while this is true, proving it turns out to be fairly tricky. To prove indistinguishability, we must design an efficient reduction B that has oracle access to an adversary A which distinguishes between the CCA2 game and the first hybrid. This reduction B should be able to use such an adversary to break security of equivocal commitments, by generating many more equivocal openings than its (non-uniform) advice would allow it to. The adversary A is a CCA2 adversary, which means it makes multiple (a-priori unbounded) calls to a $\mathsf{CCA.Val}$ oracle, and B must find a way to answer these queries. But recall that the oracle needs to perform a brute-force search through all possible seeds to find the one (if any) that matches $\mathsf{r}_{\mathsf{ext}}$ – simulating this process will make B inefficient. As such, B will need to maintain its own cheat-sheet to answer $\mathsf{CCA.Val}$ queries. Even with such a cheat-sheet, the proof is not straightforward: the set of most common equivocal queries in the CCA2 game may in general be different from the set of most common queries when B answers from its cheat-sheet.

Intermediate Cheat-Sheets. To make the proof go through, we will rely on a sequence of carefully defined intermediate cheat-sheets (that we will call lists from this point on). These will be defined inductively, and in the base case $\mathcal{L}^{(0)}$ will be empty. Let $Q = Q(\kappa)$ denote the total number of oracle calls that the attacker makes. For $j \in [1, Q]$, the j^{th} intermediate list, denoted by $\mathcal{L}^{(j)}$ will contain the $\mathsf{r}_{\mathsf{ext}}$ values and corresponding seeds for A's most common equivocal queries in its first

j oracle calls. Note that this does not suffice to fully define $\mathcal{L}^{(j)}$, since we also need to determine how the first $j-1$ oracle calls of A will be answered: in the definition of $\mathcal{L}^{(j)}$, the first j oracle calls will be answered using the CCA.ValAlt algorithm with access to the list $\mathcal{L}^{(j-1)}$. The final list $\mathcal{L}$ used by CCA.ValAlt in Game 1 will correspond exactly to $\mathcal{L} = \mathcal{L}^{(Q)}$. We show the following inductively for every j: when the first $j-1$ CCA.Val queries are answered using list $\mathcal{L}^{(j-1)}$, then it is possible to add new common equivocal queries and update the list to $\mathcal{L}^{(j)}$. This will eventually allow us to switch to the first hybrid described above, which uses CCA.ValAlt (plus the final list $\mathcal{L}^{(Q)}$).

We point the reader to Section 6.2.1 for a more detailed overview of this part of the proof. There we also discuss why for technical reasons, we require as building blocks for our equivocal commitment, keyless hash functions with specific parameters. In more detail, we require that an adversary with $S(\kappa)$ bits of advice cannot produce more than $S(\kappa) \cdot p(\kappa)$ pairs of "distinct collisions" for some a-priori fixed polynomial $p(\cdot)$, where "distinct collisions" means that no entry in any pair of collisions matches an entry in another pair. The assumption is described formally and analyzed in Section 4.1.

Completing the Analysis. After switching to CCA.ValAlt (plus the cheat-sheet), the next hybrid will sample equivocal commitments $\{\sigma_i\}_{i\in[n]}$, for the challenge commitment, together with randomness $\{y_{0,i}\}_{i\in[n]}$ and $\{y_{1,i}\}_{i\in[n]}$ that can be used to equivocally open these commitments to 0 and 1 respectively. Next, inspired by [KW19] the components $\{c^*_{x,i,b}\}_{x\in[N],i\in[n],b\in\{0,1\}}$ are modified in the challenge commitment to "drown" out information about s via noise, while relying on CCA2 security of the underlying small tag scheme to run the CCA.ValAlt function and recover values committed under $(x', \mathsf{tag}_{x'}, 0)$ and $(x', \mathsf{tag}_{x'}, 1)$. This step crucially makes use of the fact that the tags $(x', \mathsf{tag}_{x'}, 0)$ and $(x', \mathsf{tag}_{x'}, 1)$ differ from *all small tags used to generate the challenge commitment*. Finally, we rely on the security of the hinting PRG to switch to using uniform randomness everywhere.

Hinting PRGs with Injective Extension. We now describe how to achieve hinting PRGs with injective extension by modifying the constructions in [KW19]. Recall that we require hinting PRGs with injective extensions that satisfy a different security property than prior work: namely, for a uniformly random short seed s, expand $PRG(s) = z_0 z_1 z_2 \dots z_n$ and compute the injective output $\mathsf{r}_{\mathsf{ext}}$. Then compute matrix M by sampling uniformly random $v_1 v_2 \dots v_n$, and setting for all $i \in [n]$, $M_{s_i,i} = z_i$ and $M_{1-s_i,i} = v_i$. The requirement is that z_0 generated using a uniformly random seed must be indistinguishable from uniform, *even given M and given the output $\mathsf{r}_{\mathsf{ext}}$ of the injective extension*.

We build hinting PRGs with an injective extension by modularly combining the constructions in [KW19] with any leakage-resilient injective one-way function (LRIOWF). To enable this, we note that hinting PRG constructions in [KW19] from CDH and LWE have a "lossy" property, where PRG parameters can be generated in lossy mode in such a way that the output of the hinting PRG is simulatable given just a small amount of advice. We call the resulting abstraction a lossy hinting function. To achieve injectivity, we rely on a leakage resilient injective one-way function (LRIOWF) applied to the seed s of the lossy hinting function[4]. Finally, we generate the 'mask' z_0 of the hinting PRG as the Goldreich-Levin hardcore bits of the LRIOWF. To prove that z_0 is pseudorandom even in the presence of $\mathsf{r}_{\mathsf{ext}}$ and M, we will switch the lossy hinting function to

[4]For example, any sub-exponentially secure injective one-way function will suffice for our purposes.

lossy mode. In this mode the hinting function will only leak a few bits about the inverse s of the LRIOWF. We will then invoke the Goldreich-Levin theorem to argue that distinguishing the mask from uniform will require inverting the LRIOWF given just a few bits of leakage on s, which is impossible by assumption on the LRIOWF. This completes an overview of our techniques.

Comparison with Prior Work. We conclude with a comparison of our techniques against prior work that relies on keyless collision-resistant hash functions against non-uniform adversaries. While [BKP18] relies on this assumption to obtain 3-message zero-knowledge via substantially different techniques, [BL18b] applies this to a setting that is much closer to our work, that is, to achieving non-interactive non-malleable commitments. In more detail, [BL18b] use keyless hash functions against non-uniform adversaries to build a special type of 1-message zero-knowledge for NP with a *weak* soundness guarantee against non-uniform provers. They achieve this by building on the usual template for 1-message ZK, where a prover proves (via a NIWI) that either $x \in L$ or that the prover knows a trapdoor. The trapdoor, roughly, corresponds to a collision in a keyless hash function; and is derived as a function of the statement x. This ensures that a prover that can (non-uniformly) find a fixed set of non-uniform collisions will only be able to provide convincing proofs for a fixed set of statements. In their construction of non-malleable commitments, the use of NIWIs to prove a statement of the form "$x \in L$ or the prover knows a trapdoor" results in non-black-box use of the underlying base scheme.

Unlike [BL18b], we do not construct any variant of non-interactive ZK (or rely on assumptions like NIWI that imply non-interactive ZK). We develop a new template to directly achieve tag amplification for non-malleable commitments against non-uniform adversaries, without reliance on NIWIs. Our methodology to "tie" together the set of collisions an adversary can find with the number of commitments that an adversary can cheat on is entirely different from that of [BL18b].

3 Background

3.1 Non-uniform Security

We say that a cryptographic game is $\mathsf{T}(\cdot)$-non-uniform secure if for any Turing Machine in $\mathsf{poly}(\mathsf{T}(\kappa))$ time with $\mathsf{poly}(\kappa)$ non-uniform advice only has only negligible advantage in said game. We will refer to $\mathsf{poly}(\cdot)$-non-uniform secure schemes as achieving 'plain' non-uniform security.

In addition, we will say a cryptographic scheme is subexponentially secure against non-uniform adversaries if there exists some constant $c > 0$ such that the scheme is 2^{n^c}-non-uniform secure. When the constant c is explicitly required, we will say c-subexponentially secure.

3.2 CCA Commitments

We present our definition of CCA secure commitments [CLP10], which is derived from [GKLW21] with modifications made for defining security against non-uniform attackers. Intuitively, these are tagged commitments where a commitment to message m under tag tag and randomness r is created as $\mathsf{CCA.Com}(\mathsf{tag}, m; r) \rightarrow \mathsf{com}$. The scheme will be statistically binding, i.e., for all $\mathsf{tag}_0, \mathsf{tag}_1, r_0, r_1$ and $m_0 \neq m_1$ we have that $\mathsf{CCA.Com}(\mathsf{tag}_0, m_0; r_0) \neq \mathsf{CCA.Com}(\mathsf{tag}_1, m_1; r_1)$.

The hiding property is a strengthened CCA2-style definition where an attacker outputs a challenge tag tag^* along with messages m_0, m_1 and receives a challenge commitment com^* to either m_0

or m_1. The attacker's job is to guess the message that was committed to with oracle access to an (inefficient) value function $\mathsf{CCA.Val}$ where $\mathsf{CCA.Val}(\mathsf{com})$ will return m if $\mathsf{CCA.Com}(\mathsf{tag}, m; r) \to \mathsf{com}$ for some r. The attacker is allowed oracle access to $\mathsf{CCA.Val}(\cdot)$ for any $\mathsf{tag} \neq \mathsf{tag}^*$. In the non-interactive setting, the traditional notion of non-malleability (as seen in [BL18b, KK19], etc.) is simply a restriction of the CCA game where the adversary is only allowed to simultaneously submit a single set of decommitment queries. The proof of this is immediate and can be found in [BFMR18].

We mention two distinct features of our definition.First, we explicitly denote the running time of the $\mathsf{CCA.Val}$ algorithm despite the fact that it is not polynomial time. Explicitly specifying the runtime of the $\mathsf{CCA.Val}$ oracle will help us in complexity leveraging when performing tag amplification. We will call the commitment scheme to be 2^{κ^v}-efficient, i.e. can run in time (polynomially in) 2^{κ^v} where $v \geq 1$ and the security of the scheme is considered for subexponential adversaries. This additional specification was not required in [GKLW21].

Second, (as in [GKLW21]) we require a recover from randomness property, which allows one to open the commitment given all the randomness used to generate said commitment. This can be achieved generically with no additional assumptions.

Remark 3.1. We note that by considering non-uniform attackers our definition actually becomes simpler than that of [GKLW21] where they considered security against a stronger than uniform adversary, which they labeled as e-computationally enabled security. Such an adversary can run any Turing Program that runs in time $\mathsf{poly}(2^{\kappa^e})$ and obtain it's output as a non-uniform advice. This notion helped them perform complexity leveraging and obtain a uniformly secure non-malleable commitment scheme. Since we consider security against non-uniform adversaries, which are allowed to obtain non-uniform advice that may take an arbitrary amount of time to compute, our presentation is simpler.

3.2.1 Definition

A CCA secure commitment is parameterized by a tag space of size $N = N(\kappa)$ where tags are in $[1, N]$ for message space $\mathcal{M} = \{0,1\}^{w(\kappa)}$ where $w(\cdot)$ is a polynomial function (for simplicity in notation we often skip the dependence on κ). It consists of three algorithms:

- $\mathsf{CCA.Com}(1^\kappa, \mathsf{tag}, m; r) \to \mathsf{com}$ is a randomized PPT algorithm that takes as input the security parameter κ, a tag $\mathsf{tag} \in [N]$, a message $m \in \{0,1\}^w$ and outputs a commitment com, including the tag com.tag. We denote the random coins explicitly as r.
- $\mathsf{CCA.Val}(\mathsf{com}) \to m \cup \perp$ is a deterministic inefficient algorithm that takes in a commitment com and outputs either a message $m \in \{0,1\}^w$ or a reject symbol $\perp$.
- $\mathsf{CCA.Recover}(\mathsf{com}, r) \to m$ is a deterministic algorithm which takes a commitment com and the randomness r used to generate com and outputs the underlying message m.

We now define the correctness, efficiency properties, as well as the security properties of perfect binding and message hiding.

Correctness

Definition 3.1. We say that our CCA secure commitment scheme is perfectly correct if the following holds. $\forall m \in \{0,1\}^w$, $\mathsf{tag} \in [N]$ and r we have that

$$\mathsf{CCA.Val}(\mathsf{CCA.Com}(1^\kappa, \mathsf{tag}, m; r)) = m.$$

Efficiency

Definition 3.2. We say that our CCA secure commitment scheme is $\mathsf{T}(\cdot)$-efficient, if $\mathsf{CCA.Com}, \mathsf{CCA.Recover}$ run in time $\mathsf{poly}(|m|, \kappa)$, while $\mathsf{CCA.Val}$ runs in time $\mathsf{poly}(|m|, \mathsf{T}(\kappa))$.[5]

Security

Binding.

Definition 3.3. We say that our CCA secure commitment is perfectly binding if $\forall c, \forall m_0, m_1 \in \{0,1\}^w$ s.t. $m_0 \neq m_1$ and $\mathsf{CCA.Val}(c) \in \{m_1, \perp\}$, there does not exist r such that

$$\mathsf{CCA.Recover}(c, r) = m_0$$

Moreover, for any c such that $\mathsf{CCA.Val}(c) = m_1 \neq \perp$, then there exists r such that $\mathsf{CCA.Recover}(c, r) = m_1$.

Weak Binding.

Definition 3.4. We say that our CCA secure commitment is perfectly binding if $\forall c, \forall m_0, m_1 \in \{0,1\}^w$ s.t. $m_0 \neq m_1$ and $\mathsf{CCA.Val}(c) \in \{m_1, \perp\}$, there does not exist r such that

$$\mathsf{CCA.Recover}(c, r) = m_0$$

CCA Hiding. We also define a CCA message hiding game between a challenger and an attacker. The game is parameterized by a security parameter κ.

1. The attacker sends a "challenge tag" $\mathsf{tag}^* \in [N]$.
2. The attacker makes a polynomial number of repeated commitment queries com. If $\mathsf{com.tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as
$$\mathsf{CCA.Val}(\mathsf{com}).$$
3. The attacker sends two messages $m_0, m_1 \in \{0,1\}^w$.
4. The challenger flips a coin $b \in \{0,1\}$ and sends $\mathsf{com}^* = \mathsf{CCA.Com}(\mathsf{tag}^*, m_b; r)$ for randomly chosen r.

[5] In order for the scheme to be secure, the runtime of the $\mathsf{CCA.Val}$ oracle should be bigger than the runtime of the subexponential adversary. We will imagine runtime of the $\mathsf{CCA.Val}$ oracle to be 2^{κ^v} where $v > 1$.

5. The attacker again makes a polynomial number of repeated queries of commitment com. If $\mathsf{com.tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

$$\mathsf{CCA.Val}(\mathsf{com}).$$

6. The attacker finally outputs a guess b'.

We define the attacker's advantage in the game to be $\Pr[b' = b] - \frac{1}{2}$ where the probability is over all the attacker and challenger's coins.

Definition 3.5. A CCA secure commitment scheme scheme given by algorithms $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be $\mathsf{T}(\cdot)$-CCA secure if for any $\mathsf{T}(\cdot)$-non-uniform adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that the attacker's advantage in the game is $\mathsf{negl}(\kappa)$.

We also define another notion of security which we call "same tag" computation enabled secure for a weaker class of adversaries who only submit challenge queries that all have the same tag.

Definition 3.6. A CCA secure commitment scheme scheme given by algorithms $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be "same tag" $\mathsf{T}(\cdot)$-CCA secure if for any $\mathsf{T}(\cdot)$-non-uniform adversary $\mathcal{A}$ which generates queries such that all commitment queries submitted by $\mathcal{A}$ are on the same tag, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the attacker's advantage in the game is $\mathsf{negl}(\kappa)$.

Recovery From Randomness

Definition 3.7. We say that our CCA secure commitment scheme can be recovered from randomness if the following holds. For all $m \in \{0,1\}^w$, $\mathsf{tag} \in [N]$, and r we have that

$$\mathsf{CCA.Recover}(\mathsf{CCA.Com}(1^\kappa, \mathsf{tag}, m; r), r) = m.$$

4 Setupless Equivocal Commitments against Non-Uniform Adversaries

Equivocal commitments are commitments introduced by DiCrescenzo et al [CIO98] that have two computationally indistinguishable modes of setup. In the normal mode the setup outputs public parameters such that the commitment is statistically binding. In the alternate mode, the setup outputs public parameters and a trapdoor which can output commitments that open to both 0 and 1.

A setupless equivocal commitment sceme doesn't have a trusted setup algorithm. Instead we have an inefficient equivocation algorithm that can output commitments to both 0 and 1. The security of the scheme is guaranteed for adversaries that run in less than the equivocation time. A setupless equivocal commitment scheme, secure against uniform adversaries can be constructed from any setupless statistical hiding, computationally binding commitment scheme [GKLW21]. These can be built using a strong extractor and a keyless collision resistant hash function ([DPP93, HM96, BKP18]). But for non-uniform adversaries, it is easy to hardwire collisions for the setupless collision resistant hash function and hence break binding security of the scheme.

In order to achieve non-uniform security, Bitansky et al [BKP18], suggested a multi-collision resistance assumption that essentially claims that hardwiring collisions is the best that an adversary can do. Informally, the K strong multi-collision resistant property states that any non-uniform

adversary with advice advice can not output more than $\mathsf{K}(|\mathsf{advice}|)$ many collisions (assume that K blows up the length). This assumption was used by Bitansky et al [BKP18] to create statistically hiding commitments with a special binding against non-uniform adversaries.

We introduce a modified notion called "Setupless Equivocal Commitment with Auxiliary Input" that builds on these prior work, assumptions and takes in an auxiliary input $\mathsf{aux} \in \{0,1\}^*$ additionally and commits to a bit b and aux. The inefficient equivocation algorithm can take in any aux and output a commitment that can be open to both 0 and 1. We hide b (aux can not be hidden) while guaranteeing computational binding against non-uniform adversaries. We show that a similar construction showed by [BKP18] using multi-collision resistant hash functions and a strong extractor also gives this notion.

4.1 Distinct Strong Keyless Multi-Collision Resistance

The definition from [BKP18, BL18b] states that a non-uniform attacker with advice string advice cannot output more than $\mathsf{K}(\kappa, |\mathsf{advice}|)$ collisions (one can think of K as a polynomial that grows the advice length, [BL18b] say this could, for instance, be a quadratic polynomial). We further weaken the definition so that the adversary is required to output all distinct elements in its pairs of collisions, i.e. letting $\mathbf{X} = \left(X_1^{(0)}, X_1^{(1)}, \ldots, X_K^{(0)}, X_K^{(1)}\right)$, we require that there do not exist any $i, j \in [K]^2$, $b, c \in \{0,1\}^2$ such that $X_i^{(b)} = X_j^{(c)}$. We call this modified notion distinct strong multi-collision resistance. Formally,

Definition 4.1 ((T, K)-Distinct Strong Multi-Collision Resistance)**.** Let $\mathsf{T} = \mathsf{T}(\cdot)$ and $\mathsf{K} = \mathsf{K}(\cdot,\cdot)$ be functions of the security parameter κ. A keyless hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ is (T, K) distinct strong multi-collision resistant if there is a negligible function negl such that for every polynomial size non-uniform adversary $\mathcal{A}$ that runs in time $\mathsf{poly}(\mathsf{T})$ and is given advice advice of length $\mathsf{poly}(\kappa)$, for every security parameter κ, for $\mathsf{T} = \mathsf{T}(\kappa)$ and $K = \mathsf{K}(\kappa, |\mathsf{advice}|)$,

$$\Pr\left[\left(X_1^{(0)}, X_1^{(1)}, \ldots, X_K^{(0)}, X_K^{(1)}\right) \leftarrow \mathcal{A}(1^\kappa) : \begin{array}{c} \forall (i,b) \neq (j,c) \in [K] \times \{0,1\}, X_i^{(b)} \neq X_j^{(c)} \\ \forall i \in [K], \mathsf{H.Hash}(1^\kappa, X_0^{(i)}) = \mathsf{H.Hash}(1^\kappa, X_1^{(i)}) \end{array}\right] \leq \mathsf{negl}(\kappa).$$

While this is not part of our definition, for applications we will require that the number of collisions remain linear in the size of advice, i.e., there is a fixed polynomial $p(\cdot)$ such that $\mathsf{K}(\kappa, |\mathsf{advice}|) \leq p(\kappa) \cdot |\mathsf{advice}|$. In Appendix B, we show that our assumption, namely (T, K)-distinct strong multi-collision resistance holds in the auxiliary-input random oracle model [Unr07] with $p(\kappa)$ as small as 1, i.e. $\mathsf{K}(\kappa, |\mathsf{advice}|) \leq |\mathsf{advice}|$.

4.2 Setupless Equivocal Commitment with Auxillary Input

An auxiliary input equivocal commitment scheme AuxEquiv without setup consists of the algorithms:

$\mathsf{AuxEquiv.Com}(1^\kappa, \mathsf{aux}, b) \to (c, d)$ is a randomized PPT algorithm that takes in a bit $b \in \{0,1\}$, some auxiliary information $\mathsf{aux} \in \{0,1\}^*$ and security parameter $\kappa \in \mathbb{N}$ and outputs a commitment c, decommitment string d.

$\mathsf{AuxEquiv.Decom}(\mathsf{aux}, c, d) \to \{0, 1, \perp\}$ is a deterministic polytime algorithm that takes in the commitment c along with the auxiliary information aux and it's opening d and reveals the bit that it was committed to or $\perp$ to indicate failure.

$\mathsf{AuxEquiv.Equivocate}(1^\kappa, \mathsf{aux}) \to (c, d_0, d_1)$ is an (inefficient) randomized algorithm that takes in the security parameter and some auxiliary information aux and outputs a commitment string c and decommitment strings to both 0 and 1.

Definition 4.2. Correctness - We say an equivocal commitment scheme is perfectly correct if for all $b \in \{0,1\}, \mathsf{aux} \in \{0,1\}^*$,

$$\Pr\left[\begin{matrix}(c,d) \leftarrow \mathsf{AuxEquiv.Com}(1^\kappa, \mathsf{aux}, b) \\ b' \leftarrow \mathsf{AuxEquiv.Decom}(\mathsf{aux}, c, d) \\ b' = b\end{matrix}\right] = 1$$

Definition 4.3. Efficiency - We say an equivocal commitment scheme is efficient if $\mathsf{AuxEquiv.Com}$ and $\mathsf{AuxEquiv.Decom}$ run in $\mathsf{poly}(\kappa, |\mathsf{aux}|)$ time, and $\mathsf{AuxEquiv.Equivocate}$ runs in time $\mathsf{poly}(2^\kappa, |\mathsf{aux}|)$.

We now define the binding and equivocal properties.

Definition 4.4. An equivocal commitment without setup scheme is said to be $(\mathsf{T}(\cdot), \mathsf{K}(\cdot))$ binding secure if for any *non-uniform* adversary $\mathcal{A}$ running in time $\mathsf{poly}(\mathsf{T}(\kappa))$ for some polynomial and given an advice $\mathsf{advice}(\kappa)$ (for simplicity, denoted as advice) of length $\mathsf{poly}(\kappa)$ and a setting of $K = \mathsf{K}(|\mathsf{advice}|, \kappa)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that,

$$\Pr\left[\begin{matrix}\Big((\mathsf{aux}^{(1)}, c^{(1)}, d_0^{(1)}, d_1^{(1)}), \ldots, \\ (\mathsf{aux}^{(K)}, c^{(K)}, d_0^{(K)}, d_1^{(K)})\Big) \leftarrow \mathcal{A}(1^\kappa)\end{matrix} : \begin{matrix}\forall i \in [K], \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_0^{(i)}) = 0, \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_1^{(i)}) = 1 \\ \forall i \neq j \in [K], \mathsf{aux}^{(i)} \neq \mathsf{aux}^{(j)}\end{matrix}\right] \leq \mathsf{negl}(\kappa).$$

Definition 4.5. We say that a scheme is equivocal if for all $b \in \{0,1\}, \mathsf{aux} \in \{0,1\}^*$ the statistical difference between the following two distributions is negligible in κ.

- $\mathcal{D}_0 = (\mathsf{aux}, c, d)$ where $\mathsf{AuxEquiv.Com}(1^\kappa, \mathsf{aux}, b) \to (c, d)$.
- $\mathcal{D}_1 = (\mathsf{aux}, c, d_b)$ where $\mathsf{AuxEquiv.Equivocate}(1^\kappa, \mathsf{aux}) \to (c, d_0, d_1)$.

4.3 Construction

We construct auxiliary-input equivocal commitments assuming a keyless hash function that is distinct strong multi-collision resistant and a strong extractor. This is based on constructions introduced and presented in [DPP93, HM96, BKP18]. Let the keyless hash function be $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$. A $(\kappa, \mathsf{negl}(\kappa)$ strong extractor SExt (Appendix A) that takes a seed of κ bits and an input of 3κ bits and outputs a single bit, $\mathsf{SExt} : \{0,1\}^\kappa \times \{0,1\}^{3\kappa} \to \{0,1\}$.

$\mathsf{AuxEquiv.Com}(1^\kappa, \mathsf{aux}, b) \to (c, d)$.
Sample a seed $g \leftarrow \{0,1\}^\kappa$. Choose $v \leftarrow \{0,1\}^{3\kappa}$. Compute $w = b \oplus \mathsf{SExt}(g, v)$. Compute $h = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v))$. Compute $c = (g, w, h)$ and $d = v$.

$\mathsf{AuxEquiv.Decom}(\mathsf{aux}, c, d) \to \{0, 1, \perp\}$

Parse c as (g, w, h). Check if $h = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, d))$, output $\perp$ if fails. Output $w \oplus \mathsf{SExt}(g, d)$.

$\mathsf{AuxEquiv.Equivocate}(1^\kappa, \mathsf{aux}) \to (c, d_0, d_1)$

Sample a seed $g \leftarrow \{0,1\}^\kappa$ for a SExt. Sample $w \leftarrow \{0,1\}$. Sample $t \xleftarrow{R} \{0,1\}^{3\kappa}$.
Define $\mathcal{V}_t = \{v : \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))\}$. Partition $\mathcal{V}_t = \mathcal{V}_t^0 \cup \mathcal{V}_t^1$ where $\mathcal{V}_t^i = \{v : v \in \mathcal{V}_t \wedge \mathsf{SExt}(g, v) = i\}$, output $\perp$ if either $\mathcal{V}_t^0$ or $\mathcal{V}_t^1$ are $\emptyset$.
Sample $v_0 \xleftarrow{R} \mathcal{V}_t^w, v_1 \xleftarrow{R} \mathcal{V}_t^{w \oplus 1}$. Output $\perp$ if no such v_0 or v_1 exist. $h \leftarrow \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))$.
Output $((g, w, h), v_0, v_1)$.

We defer the analysis of this construction and a proof of the following lemma to the appendix (Appendix C).

Lemma 4.1. If $\mathsf{H}(\cdot)$ is a $(\mathsf{T}(\cdot), \mathsf{K}(\cdot,\cdot))$ distinct strong multi-collision resistant keyless hash function against non-uniform adversaries and SExt is a $(k, \epsilon) = (\kappa, \mathsf{negl}(\kappa))$ Strong Seeded extractor, then the construction above is a correct and efficient equivocal commitment scheme (Definition 4.3), and is $(\mathsf{T}(\cdot), \mathsf{K}(\cdot,\cdot))$-binding secure (Definition 4.4).

4.4 Amplification

Lemma 4.2. If there exists a $(\mathsf{T}(\cdot), \mathsf{K}(\cdot,\cdot)))$-binding equivocal commitment scheme, then for any polynomial $p(\cdot)$, there exists a $(\mathsf{T}(\cdot), \mathsf{K}(\cdot,\cdot))/p(\kappa))$-binding equivocal commitment scheme.

Proof. Let Small.AuxEquiv.Com, Small.AuxEquiv.Decom, Small.AuxEquiv.Equivocate be a $(\mathsf{T}(\cdot), \mathsf{K}(\cdot))$-binding equivocal commitment scheme. Consider a $p(\cdot)-$parallel repetition of Small.AuxEquiv

$\mathsf{AuxEquiv.Com}(1^\kappa, \mathsf{aux}, b) \to (c, d)$.

For $i \in [p(\kappa)]$, run $(c_i, d_i) \leftarrow \mathsf{Small.AuxEquiv.Com}(1^\kappa, (\mathsf{aux}, i), b)$. Output $(c = \{c_i\}, d = \{d_i\})$

$\mathsf{AuxEquiv.Decom}(\mathsf{aux}, c, d) \to \{0, 1, \perp\}$

If $\exists b \in \{0,1\} : \forall i in [p(\kappa)]$, $\mathsf{AuxEquiv.Decom}((\mathsf{aux}, i), c_i, d_i) = b$, output b. Otherwise output $\perp$.

$\mathsf{AuxEquiv.Equivocate}(1^\kappa, \mathsf{aux}) \to (c, d_0, d_1)$

For $i \in [p(\kappa)]$, run $(c_i, d_{0,i}, d_{1,i}) \leftarrow \mathsf{Small.AuxEquiv.Equivocate}(1^\kappa, (\mathsf{aux}, i))$. Output $(c = \{c_i\}, d_0 = \{d_{0,i}\}, d_1 = \{d_{1,i}\})$

We defer the analysis of this construction to the appendix (Appendix C).

□

Corollary 4.1. Assume there exists a polynomial $p = p(\kappa)$ such that there exists a $(\mathsf{T}(\cdot), p(\cdot)|\mathsf{advice}|)$ distinct strong collision resistant hash function satisfying Definition 4.1, then for every polynomial $\mathsf{poly}(\cdot)$, there exists a $(\mathsf{T}(\cdot), \frac{|\mathsf{advice}|}{\mathsf{poly}(\kappa)})$-binding equivocal commitment scheme.

Proof. Fix the polynomial $p(\cdot)$ and the distinct strong collision resistant hash function that is guaranteed by the assumption. By lemma 4.1, there exists a correct and efficienct equivocal commitment that is $(\mathsf{T}(\cdot), p(\cdot)|\mathsf{advice}|)$-binding. Fix any polynomial $\mathsf{poly}(\cdot)$. Then by invoking lemma 4.2 on the polynomial $\mathsf{poly}(\cdot)p(\cdot)$, we have that there exists a $(\mathsf{T}(\cdot), \frac{|\mathsf{advice}|}{\mathsf{poly}(\kappa)})$-binding equivocal commitment scheme. □

5 Hinting PRGs with injective extension

A hinting pseudorandom generator as introduced by Koppula and Waters[KW19] is a pseudorandom generator with an enhanced security property. In this security game blocks that are output from the PRG are interspersed with random blocks where the placement is according to the seed of the PRG.

In this section we introduce a variant of Hinting PRGS that we call Hinting PRGs with injective extension. Our variant follows along the lines of the original, but with two critical modifications. The first is that we slightly relax the security game. On a seed s of length n bits, the hinting PRG outputs length $n+1$ blocks each consisting of ℓ bits. Informally, our security guarantee is that the adversary cannot distinguish between the following two distributions, each consisting of $(2n+1)$ blocks. In both distributions, all blocks but the first are generated identically: these output as a $2 \times n$ matrix where for all $i \in [n]$ the $(s_i, i)^{th}$ entry is set according to the $(i+1)^{th}$ block of the PRG evaluation, while the $(1-s_i, i)^{th}$ entry is a uniformly random string. In the first distribution, the first ℓ-bit block is set as the first block of the PRG evaluation, and in the second distribution, the first ℓ-bit block is set uniformly at random.

This relaxed security definition differs from the original security definition in which the second distribution consists of all random blocks. It is fairly easy to observe that our relaxed notion also suffices for performing the CCA transformation of [KW19] and will also suffice for our purposes. The primary reason for relaxing the security definition, is that it makes it easier to realize our second modification.

We additionally define an injective extension for the hinting PRG, where we require that the Hinting PRG evaluation algorithm additionally outputs a separate block that is injective with respect to the seed. To ensure injectivity we will define an algorithm that checks the Hinting PRG public parameters and outputs 0 if the public parameters were sampled so that the extended block might not be an injective function of the seed. That is there could be two seeds that output the same extended block. If the check function outputs 1, the extended block will be an injective function of the seed. The hinting PRG scheme consists of the following algorithms,

$\mathsf{Setup}(1^\kappa, 1^\ell)$: The setup algorithm takes as input the security parameter κ, and length parameter ℓ, and outputs public parameters pp and input length $n = n(\kappa, \ell)$

$\mathsf{Eval}\,(\mathsf{pp}, s \in \{0,1\}^n, i \in [n] \cup \{0\})$: The evaluation algorithm takes as input the public parameters pp, an n bit string s, an index $i \in [n] \cup \{0\}$ and outputs an ℓ bit string y.

$\mathsf{ExtEval}\,(\mathsf{pp}, s \in \{0,1\}^n)$: The extended evaluation algorithm takes as input the public parameters pp, an n bit string s and outputs a string of length $m = m(\kappa, \ell)$.

$\mathsf{CheckParams}\,(\mathsf{pp}, n)$: The algorithm takes as input the public parameters pp, the seed input length n and checks them to see if the function sampled is injective or not. It outputs $\{0,1\}$ accordingly.

Definition 5.1. A hinting PRG scheme is said to be non-uniform $T(\cdot)$-secure if for any polynomial $\ell(\cdot)$ and any adversary $\mathcal{A}$ running in time $\mathsf{poly}(T(\kappa))$ and $\mathsf{poly}(\kappa)$ advice, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds:

$$\left| \Pr \left[\beta \leftarrow A\left(\mathsf{pp}, \left(r_0^\beta, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n], b\in\{0,1\}}\right)\right) : \begin{array}{c} (\mathsf{pp}, n) \leftarrow \mathsf{Setup}(1^\kappa, 1^{\ell(\kappa)}), s \leftarrow \{0,1\}^n, \beta \leftarrow \{0,1\}, \\ r_0^0 = \mathsf{Eval}(\mathsf{pp}, s, 0), r_{\mathsf{ext}} = \mathsf{ExtEval}(\mathsf{pp}, s), r_0^1 \leftarrow \{0,1\}^\ell, \\ r_{i,s_i} = \mathsf{Eval}(\mathsf{pp}, s, i), r_{i,\overline{s_i}} \leftarrow \{0,1\}^\ell \; \forall\, i \in [n] \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\kappa)$$

Definition 5.2. A hinting PRG scheme is said to be extended injectively if for any security parameter $\kappa \in \mathbb{N}$, any polynomial $\ell(\cdot)$ and any $\mathsf{pp} \in \{0,1\}^*$ the following holds,

$$\Pr\left[\begin{array}{c} \exists s_1 \neq s_2 \in \{0,1\}^n, \\ \mathsf{ExtEval}(\mathsf{pp}, s_1) = \mathsf{ExtEval}(\mathsf{pp}, s_2) \end{array} : \begin{array}{c} n \in \mathbb{N} \\ \mathsf{CheckParams}(\mathsf{pp}, n) = 1 \end{array}\right] = 0.$$

Definition 5.3. A hinting PRG scheme is setup such that it outputs injective parameters if for any security parameter $\kappa \in \mathbb{N}$, any polynomial $\ell(\cdot)$ the following holds,

$$\Pr\left[\mathsf{CheckParams}(\mathsf{pp}, n) = 0 : (\mathsf{pp}, n) \leftarrow \mathsf{Setup}(1^\kappa, 1^{\ell(\kappa)})\right] = 0.$$

Definition 5.4. A hinting PRG scheme is succinct if the length of the seed n, public parameters and injective extension are independent of the block length parameter ℓ.

5.1 Definitions

5.1.1 Lossy hinting functions

To construct our injectively extended hinting PRG's, we define a new abstraction of lossy hinting functions, which capture some seperate properties of many known hinting PRG constructions. These can be constructed using both Diffie-Hellman and LWE assumptions (Sections D.2, D.3, D.4).

$\mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r_{Setup}})$: The setup is a randomized algorithm takes as input the security parameter κ, and length parameter ℓ, a seed length n, and outputs public parameters pp.

$\mathsf{LossyHint.Eval}\,(\mathsf{pp}, s \in \{0,1\}^n, i \in [n])$: The evaluation is a deterministic algorithm takes as input the public parameters pp, an n bit string s, an index $i \in [n]$ and outputs an ℓ bit string y.

$\mathsf{LossyHint.Sim}_1(\mathsf{r_{Setup}}, \mathsf{pp}, s)$: The simulator is a deterministic algorithm which takes the setup randomness and seed and outputs a short binary string hint.

$\mathsf{LossyHint.Sim}_2(\mathsf{r_{Setup}}, \mathsf{LossyHint.hint}, \mathsf{pp}, i \in [n], b \in \{0,1\})$: The second simulator is a deterministic algorithm which takes in the short hint string from $\mathsf{LossyHint.Sim}_1$ to simulate and recreate the output of $\mathsf{LossyHint.Eval}$ or a random string.

Definition 5.5. A lossy hinting function is non-uniform $T(\cdot)$-secure if for all $\mathsf{poly}(T(\kappa))$ time adversaries $\mathcal{A}$ with $\mathsf{poly}(\kappa)$ non-uniform advice, for all $\ell, n \in \mathsf{poly}(\kappa)$, there exists a negligible function negl such that the following holds

$$\left|\Pr\left[\beta \leftarrow A\left(\mathsf{pp}, \left(s, \left\{r^\beta_{i,b}\right\}_{i\in[n], b\in\{0,1\}}\right)\right) : \begin{array}{c} \mathsf{r_{Setup}} \xleftarrow{R} \{0,1\}^*, s \xleftarrow{R} \{0,1\}^n, \beta \xleftarrow{R} \{0,1\} \\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, 1^{\ell(\kappa)}, 1^n; \mathsf{r_{Setup}}), \mathsf{hint} \leftarrow \mathsf{Sim}_1(\mathsf{r_{Setup}}, \mathsf{pp}, s) \\ r^0_{i,s_i} = \mathsf{Eval}(\mathsf{pp}, s, i), r^0_{i,\overline{s_i}} \leftarrow \{0,1\}^\ell \; \forall\, i \in [n] \\ r^1_{i,b} = \mathsf{Sim}_2(\mathsf{r_{Setup}}, \mathsf{hint}, \mathsf{pp}, i, b) \; \forall i \in [n] b \in \{0,1\} \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\kappa)$$

We observe here the security definition differs from the traditional hinting PRG definition in a couple of ways. Perhaps most notably, the seed s itself is given to the adversary, but rather than attempting to distinguish the whole distribution from random, the adversary is tasked with distinguishing the traditional hinting PRG output (where function blocks are interleaved with truly random blocks depending on the seed bit) with a simulation, which does not have access to the seed but instead a much shorter 'hint' of it.

Definition 5.6. A lossy hinting function scheme is $\rho(\cdot)$-retaining if $\forall\kappa, \forall n, \ell \in \mathsf{poly}(\kappa)\ \rho(\kappa) \geq |\mathsf{hint}|$.

5.1.2 Leakage Resilient One Way Functions

Definition 5.7. A function family (defined by a randomized algorithm) $\mathcal{F}(\kappa) = \{f : \{0,1\}^\kappa \to \mathcal{R}_f\}$ is one way against a class of adversaries $\mathcal{A}_\kappa$ if for all $\mathcal{A} \in \mathcal{A}_\kappa$, there exists a negligible function negl such that

$$\Pr\begin{bmatrix} f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^\kappa) \\ x \xleftarrow{R} \{0,1\}^\kappa \\ x' \leftarrow \mathcal{A}(1^\kappa, f, f(x)) \\ f(x') = f(x) \end{bmatrix} \leq \mathsf{negl}(\kappa)$$

Definition 5.8. A one way function family $\mathcal{F}(\kappa) = \{f : \{0,1\}^\kappa \to \mathcal{R}_f\}$ is $\ell(\cdot)$-leakage resilient if for all $g : \{0,1\}^\kappa \to \{0,1\}^\ell$, the function family $\mathcal{F}' = \{f'(x) = f(x)||g(x)\}$ is one way.

Lemma 5.1. Let $\mathcal{F}$ be an (injective) subexponentially secure one way function family secure against time $\mathsf{poly}(2^{\kappa^c})$ adversaries. Then $\mathcal{F}$ is also $\kappa^c(\cdot)$ leakage-resilient (injective) one way function family secure against time $\mathsf{poly}(2^{\kappa^c})$ adversaries.

Proof. Let $\mathcal{A}$ be an adversary against a $\kappa^c(\cdot)$ leakage-resilient one way function family $\mathcal{F}$. Then consider the following adversary

$$\mathcal{A}'(f \in \mathcal{F}, y \in \mathcal{R})$$

- For every string $s \in \{0,1\}^{\kappa^c(\kappa)}$
  - Run $x \leftarrow \mathcal{A}(f, y)$ with s as the output to the leakage function
  - If $f(x) = y$, output x, otherwise, continue.

Since $\mathcal{A}$ runs in time $\mathsf{poly}(2^{\kappa^c})$ time, we can see that $\mathcal{A}'$ runs $\mathcal{A}$ a total of 2^{κ^c} times, so runs in time $2^{n^c} \cdot \mathsf{poly}(2^{\kappa^c}) \in \mathsf{poly}(2^{\kappa^c})$. Furthermore, since $\mathcal{A}'$ runs $\mathcal{A}$ on all strings s, this includes s which is equal to the output of the leakage function. Since we can easily verify the success of $\mathcal{A}$ by evaluating f, this lower bounds the success probability of $\mathcal{A}'$ with that of $\mathcal{A}$.

□

5.1.3 Goldreich Levin Hardcore bits

Theorem 5.1. [GL89] There exists function $b \leftarrow \mathsf{hcb}(x; r)$ such that for any one way function family $\mathcal{F}$, $(f, f(x), r, \mathsf{hcb}(x; r)) \approx_c (f, f(x), r, b')$, where $b' \xleftarrow{R} \{0,1\}$, $f \leftarrow \mathsf{Gen}_{\mathcal{F}}$, $x \xleftarrow{R} \{0,1\}^n$ and $r \xleftarrow{R} \{0,1\}^n$.

5.2 Construction

Using subexponentially secure lossy hinting functions and subexponentially secure leakage resilient injective one way functions, we can construct (subexponentially secure) hinting PRG's with injective extension. Since the lossy hinting function has most of the components of a hinting PRG, we primarily need to construct the additional zeroth block of the hinting PRG and the injective extension. We do this by taking the seed of the lossy hinting function and outputting the leakage resilient injective OWF applied to it as the injective extension. To get the zeroth block, we simply output ℓ independently generated GL hardcore bits of the OWF. We leverage the leakage resilience of the OWF in addition to the lossiness of the PRG to prove that the GL hardcore bits appear pseudorandom even given the rest of the lossy hinting function as output.

$\mathsf{HPRG.Setup}(1^\kappa, 1^\ell)$
Compute $n = (\rho(\kappa)+\ell)^{1/\delta}$. Let $\mathsf{LossyHint.pp} \leftarrow \mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n)$. Let $f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^n)$. Sample $r_1, \dots r_\ell \xleftarrow{R} (\{0,1\}^n)^\ell$. Output $\mathsf{HPRG.pp} = (\mathsf{LossyHint.pp}, f, r_1, \dots r_\ell)$ and n.

$\mathsf{HPRG.Eval}\,(\mathsf{HPRG.pp} = (\mathsf{LossyHint.pp}, n, f, r_1, \dots r_\ell), s \in \{0,1\}^n, i \in [n] \cup \{0\})$
If $i = 0$; for $j \in [\ell]$ set $b_j \leftarrow \mathsf{hcb}(s, r_j)$. Output $b_1 b_2 \dots b_\ell$.
Else, output $\mathsf{LossyHint.Eval}(\mathsf{LossyHint.pp}, s, i)$.

$\mathsf{HPRG.ExtEval}\,(\mathsf{pp}, s \in \{0,1\}^n)$. Output $f(s)$.

$\mathsf{HPRG.CheckParams}\,(\mathsf{HPRG.pp}, n)$. Return true.

We present the security analysis in the appendix (Appendix D.1).

5.2.1 Achieving Succinctness

Taken as written above, the hinting PRG scheme does not necessarily satisfy Definition 5.4. However, we can generically transform any hinting PRG scheme into one which satisfies this property by simply extending the output length with plain (non-hinting) PRGs

Lemma 5.2. Let HPRG be a 2^{κ^δ}-secure hinting PRG and let $\mathsf{PRG}(s, 1^\ell)$ be a 2^{κ^ϵ}-secure PRG with variable output length ℓ bits. Then the following is a succinct 2^{κ^δ}-secure hinting PRG.

$S.\mathsf{HPRG.Setup}(1^\kappa, 1^\ell)$
Compute $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$, output $((\mathsf{pp}, \ell), n)$.

$S.\mathsf{HPRG.Eval}\,((\mathsf{pp}, \ell), s, i)$
Compute $\mathsf{PRG}(\mathsf{HPRG.Eval}(\mathsf{pp}, s, i), 1^\ell)$

$S.\mathsf{HPRG.ExtEval}\,((\mathsf{pp}, \ell), s \in \{0,1\}^n)$. Output $\mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$.

$S.\mathsf{HPRG.CheckParams}\,((\mathsf{pp}, \ell), n)$. Return $\mathsf{HPRG.CheckParams}(\mathsf{pp}, n)$.

We defer the security analysis to the appendix (Appendix D.5).

6 Tag Amplification

We discuss how to amplify a non-uniform subexponentially secure CCA scheme for $N' = 4N$ tags to a scheme with 2^N tags. We will perform the amplification using non uniform subexponentially secure primitives AuxEquiv (Section 4), extended hinting PRG (Section 5). The amplification algorithm runs in time polynomial in N and the runtime of the primitives involved, thus N should always stay polynomial in the security parameter for the amplification to be an efficient algorithm.

Let the hinting PRG scheme $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{ExtEval}, \mathsf{CheckParams})$ be a succinct $T = 2^{\kappa^\gamma}$ secure for some constant $\gamma \in (0,1)$. Let AuxEquiv be $T = 2^{\kappa^\delta}$-binding secure and statistically hiding where $\delta \in (0,1)$. Let $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ be a 2^{κ^c}-subexponentially secure, weak binding, 2^{κ^v}-efficient CCA commitment scheme for $N'(\kappa) = N' = 4N$ tags where $c < 1$ and $v \geq 1$ for message length $u(\kappa)$[6]. We will assume tags take identities of the form $(i, \beta, \Gamma) \in [N] \times \{0,1\} \times \{0,1\}$ and that the Small.Com algorithm take in random coins of length $\ell(\kappa)$.

Let m be the message input to the commitment algorithm and length be denoted by $|m|$. Let $n' = n'(\kappa)$ be the length of the seed plus public parameters plus injective extension of the hinting PRG scheme when invoked on security parameter $\kappa'' = \kappa^{\frac{v}{\delta\gamma}}$. Since the scheme is succinct, n' is a function of only κ'' (and hence κ) and not the block length, which we will specify later. By Lemma 4.2, we will use a $(2^{\kappa^\delta}, \frac{|\mathsf{advice}|}{2 \cdot n'})$-binding secure commitment scheme AuxEquiv, and let $|y|$ refer to the length of the decommitment strings of said scheme. Finally, we run Small.Com on messages of size $|y|$, and let ℓ be the size of randomness used by Small.Com on said input size. We set the block size of our hinting PRG scheme to be the maximum of $|m|, N \cdot \ell$. For ease of notation we assume that $\mathsf{HPRG.Eval}(\mathsf{pp}, s, 0) \in \{0,1\}^{|m|}$ and $\forall i \in [n]$, $\mathsf{HPRG.Eval}(\mathsf{pp}, s, i) \in \{0,1\}^{\ell \cdot N}$, i.e. we ignore any extra bits output by the HPRG.Eval algorithm. Let $\Theta(\kappa^{\tilde{v}})$ denote the length of the seed n in relation to the security parameter.

Our transformation will produce three algorithms, $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ which we prove non-uniform 2^{κ^c}-subexponentially secure and $2^{\kappa^{v'}}$-efficient where $v' = \frac{v \cdot \tilde{v}}{\delta \cdot \gamma}$. The construction will call AuxEquiv on security parameter $\kappa' = \kappa^{\frac{v}{\delta}}$, HPRG on security parameter $\kappa'' = \kappa^{\frac{v}{\delta \cdot \gamma}}$ and Small on security parameter κ.

The different parameters will help us perform complexity leveraging. For simplicity, we assume that the message space of Small, $u(\kappa)$ is equal to the length of the decommitment string of the equivocal commitment called on κ'. We will ensure this property is satisfied in Section 7 when we recursively amplify the tags. The CCA.Val procedure in our transformation will be an inefficient algorithm that brute forces through each hinting PRG seed and run in time 2^n where $n = \Theta(\kappa''^{\tilde{v}})$. Thus our transformation will increase the runtime of CCA.Val from Small.Val that runs in time 2^{κ^v} to $2^{\kappa^{v'}}$.

Additionally, we will also present a fourth non-uniform algorithm CCA.ValAlt, which is only used in the proof and depends on the non-uniform advice it gets. In our proof we will first change how we answer an adversary's decommitment queries by using CCA.ValAlt to answer instead of CCA.Val. Since the queries made to the CCA.Val oracles differ in at least one position from tag^*, CCA.ValAlt will crucially rely on the security of Small.Com at this position by making calls to Small.Val to help in decommitment.

$\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}) \to m \cup \perp$ is a deterministic inefficient algorithm that takes in tag^*, a

[6] Recall from Definition 3.2 that a 2^{κ^v}-efficient scheme with $v \geq 1$ implies that the runtime of Small.Val is polynomial in 2^{κ^v}.

commitment com and a non-uniform advice list $\mathcal{L}$ and outputs either a message $m \in \{0,1\}^w$ or a reject symbol $\perp$. It will be used solely as an instrument in proving the scheme secure and not exported as part of the interface.

CCA.FindSeed(aux)

Inputs: String aux $=$ (HPRG.pp, aux$'$)
Output: $\tilde{s} \in \{0,1\}^n \cup \perp$

- Parse aux as (HPRG.pp, aux$'$)
- Iterate through all $\tilde{s} \in \{0,1\}^n$
  - If aux$' =$ HPRG.ExtEval(HPRG.pp, $\tilde{s}$), return s.
- Return $\perp$

Figure 1: Routine CCA.FindSeed

CCA.Check($\tilde{s}$, com)

Inputs: Seed candidate $\tilde{s} = \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$
Commitment com $= \left(\mathsf{tag}, \mathsf{aux}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]}\right)$
Output: $\{0,1\}$

- For $i \in [n]$
  1. Compute $(r_{1,i}, r_{2,i}, \ldots, r_{N,i}) =$ HPRG.Eval(HPRG.pp, $\tilde{s}, i$)
  2. For $x \in [N]$
     (a) Let $\tilde{y}_i =$ Small.Recover$(c_{x,i,\tilde{s}_i}, r_{x,i})$. If $\tilde{y}_i = \perp$, output 0
     (b) If $c_{x,i,\tilde{s}_i} \neq$ Small.Com$(1^\kappa, (x, \mathsf{tag}_x, \tilde{s}_i), \tilde{y}_i; r_{x,i})$, output 0.
     (c) If $\tilde{s}_i \neq$ AuxEquiv.Decom(aux, $\sigma_i, \tilde{y}_i$), output 0.
- Parse aux as (HPRG.pp, aux$'$).
- If HPRG.CheckParams(HPRG.pp, n) $= 0$, output 0.
- If aux$' \neq$ HPRG.ExtEval(HPRG.pp, s) output 0.
- If all the above checks have passed, output 1.

Figure 2: Routine CCA.Check

We now describe our transformation.

Transformation Amplify(Small $=$ (Small.Com, Small.Val, Small.Recover), HPRG, AuxEquiv, $w(\kappa), v'$) $\rightarrow$ NM $=$ (CCA.Com, CCA.Val, CCA.Recover) :

CCA.Com$(1^\kappa, \mathsf{tag}, m \in \{0,1\}^{w(\kappa)}; r) \rightarrow$ com

CCA.FindAlt$(x', \mathsf{com}, \mathcal{L})$

Inputs: Index $x' \in [N]$

Commitment $\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]}\right)$

Polynomial Size Non-Uniform Advice List $\mathcal{L}$

Output: $\tilde{s} \in \{0,1\}^n$

- If for some $\tilde{s} \in \{0,1\}^n$, $(\mathsf{com.aux}, \tilde{s}) \in \mathcal{L}$, where $\tilde{s}$ is the seed recorded from the advice. Output $\tilde{s}$.
- Else if com.aux is not recorded in $\mathcal{L}$,
  - For each $i \in [n]$
    1. Let $\tilde{y}_i = \mathsf{Small.Val}(c_{x',i,0})$
    2. Set $\tilde{z}_i = \mathsf{AuxEquiv.Decom}(\mathsf{aux}, \sigma_i, \tilde{y}_i)$. If $\tilde{z}_i = \perp$, set $\tilde{s}_i = 1$. Else, set $\tilde{s}_i = \tilde{z}_i$.
  - Output $\tilde{s} = \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$.

Figure 3: Routine CCA.FindAlt

CCA.Equiv(com)

Inputs: Commitment $\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux} = (\mathsf{HPRG.pp}, \mathsf{aux}'), c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]}\right)$

Output: Equivocation $(\mathsf{aux}, c, d_0, d_1) \cup \perp$

- For all $i \in [n], x \in [N], b \in \{0,1\}$,
  - If, $\mathsf{AuxEquiv.Decom}(\mathsf{Small.Val}(\mathsf{com}.c_{x,i,b})) = 0$, and $\mathsf{AuxEquiv.Decom}(\mathsf{Small.Val}(\mathsf{com}.c_{x,i,\bar{b}})) = 1$.
    Return $(\mathsf{aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x,i,b}), \mathsf{Small.Val}(\mathsf{com}.c_{x,i,\bar{b}}))$
- Return $\perp$

Figure 4: Routine CCA.Equiv

1. Compute $\kappa' = \kappa^{\frac{v}{\delta}}$. Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.[7]
2. Sample $(\mathsf{HPRG.pp}, n) \leftarrow \mathsf{HPRG.Setup}(1^{\kappa''}, 1^{\max(|m|, N\cdot\ell)})$.
3. Sample $s = s_1 \ldots s_n \xleftarrow{R} \{0,1\}^n$ as the seed of the extended hinting PRG.
4. Set $\mathsf{aux} = (\mathsf{HPRG.pp}, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s))$.
5. For all $i \in [n]$ run $\mathsf{AuxEquiv.Com}(1^{\kappa'}, \mathsf{aux}, s_i) \rightarrow (\sigma_i, y_i)$.
6. Let for $x \in [N], i \in [n]$, $r_{x,i}, \tilde{r}_{x,i} \in \{0,1\}^\ell$ be defined as follows:
7. For $i \in [n]$
   (a) Compute $(r_{1,i}, r_{2,i}, \ldots, r_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i)$

[7]The variables δ and γ are known from the security guarantees of AuxEquiv, HPRG respectively.

$\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^*; r_\mathcal{A}, r_\mathcal{C})$

Inputs: Security Parameter 1^κ
Adversary Algorithm $\mathcal{A}$ with advice $\mathsf{advice}_\mathcal{A}$
Query Count j
List of $(\mathsf{aux}, \mathsf{seed})$ pairs $\mathcal{L}^*$
Adversary and Challenger Random coins $r_\mathcal{A}, r_\mathcal{C}$
Output: List of "equivocating" aux values $\mathcal{E} = \{(\mathsf{aux}, c, d_0, d_1)\}$

- Initialize $\mathcal{A}$ with coins $r_\mathcal{A}$ and $\mathcal{C}$ with coins $r_\mathcal{C}$.
- Initialize $\mathcal{E} = \emptyset$
- If $j = 0$ abort and return $\mathcal{E}$
- $\mathcal{A}$ sends a "challenge tag" tag^*.
- **Pre Challenge Phase:** $\mathcal{A}$ makes a polynomial number of repeated commitment queries com. If $\mathsf{com.tag} = \mathsf{tag}^*$, $\mathcal{C}$ responds with $\perp$. For every query (with query number $\eta \in [\mathsf{Q}]$) made by $\mathcal{A}$.
  - If $\mathsf{CCA.Equiv}(\mathsf{com}) \neq \perp$ and $(\mathsf{com.aux}, \_, \_, \_) \notin \mathcal{E}$, add $\mathsf{CCA.Equiv}(\mathsf{com})$ to $\mathcal{E}$[a]
  - If $\eta = j$, abort and return $\mathcal{E}$
  - Respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}^*)$.
- $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^w$.
- $\mathcal{C}$ flips $b \in \{0,1\}$ and sends $\mathsf{com}^* = \mathsf{CCA.Com}(\mathsf{tag}^*, m_b; r)$ for randomly chosen r.
- **Post Challenge Phase:** Respond exactly as **Pre Challenge Phase**.
- Return $\mathcal{E}$ if experiment has not yet aborted.

[a] For notational convenience and clarity, we leave elements of a tuple we do not later refer to as _ when pattern matching.

Figure 5: Routine CCA.Exp, which returns a list of "equivocating" aux values

(b) Sample $(\tilde{r}_{1,i}, \tilde{r}_{2,i}, \ldots, \tilde{r}_{N,i}) \xleftarrow{R} \{0,1\}^{N \cdot \ell}$

8. Compute $c = m \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$
9. For $i \in [n]$, $x \in [N]$
   (a) If $s_i = 0$
      i. $c_{x,i,0} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 0), \mathsf{msg} = y_i; r_{x,i})$
      ii. $c_{x,i,1} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 1), \mathsf{msg} = y_i; \tilde{r}_{x,i})$
   (b) If $s_i = 1$
      i. $c_{x,i,0} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 0), \mathsf{msg} = y_i; \tilde{r}_{x,i})$
      ii. $c_{x,i,1} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 1), \mathsf{msg} = y_i; r_{x,i})$
10. Output $\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N]})_{i \in [n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

$\mathsf{CCA.AdviceList}(1^\kappa, \mathcal{A})$

Inputs: Security Parameter 1^κ, Adversary Algorithm $\mathcal{A}$ with advice $\mathsf{advice}_{\mathcal{A}}$

Output: Polynomial Size Non-Uniform Advice Lists $\mathcal{L}^{(0)}, \mathcal{L}^{(1)}, \ldots, \mathcal{L} = \mathcal{L}^{(\mathsf{Q})}$

- Let $\mathcal{A}$ make maximum $\mathsf{Q} = \mathsf{Q}(\kappa)$ queries during the query phases to the challenger. We will set $\mathcal{L}^{(0)} = \emptyset$.
- From construction in Section 6, let n denote the hinting PRG seed length and $n' - n$ denote the length of aux (which means n' is equal to the length of the public parameters plus the length of the injective extension plus the length of seed).
- For $j \in [\mathsf{Q}]$, we iteratively define $\mathcal{L}^{(j)}$ from $\mathcal{L}^{(j-1)}$ as follows.
  - Let $r_{\mathcal{A}}, r_{\mathcal{C}}$ denote the random coins of $\mathcal{A}$ and $\mathcal{C}$ respectively.
  - For every $\mathsf{aux} \in \{0,1\}^{n'-n}$, let
$$p^j_{\mathsf{aux}} = \Pr_{r_{\mathcal{A}}, r_{\mathcal{C}}}\left[(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)}; r_{\mathcal{A}}, r_{\mathcal{C}})\right]$$
  - Sort all $\mathsf{aux} \in \{0,1\}^{n'-n}$ according to the descending order of p^j_{aux} (ties broken lexicographically). Let $K = \frac{|\mathsf{advice}_{\mathcal{A}}|}{n'}$.
  - Set $\mathcal{L}^{(j)} = \{(\mathsf{aux}_i, \mathsf{CCA.FindSeed}(\mathsf{aux}_i))\}_{i \in [K]}$ as the top $K = \frac{|\mathsf{advice}_{\mathcal{A}}|}{n'}$ values of aux for which p^j_{aux} is the greatest (ties can be broken lexicographically).
- Denote $\mathcal{L}$ as $\mathcal{L}^{(\mathsf{Q})}$. Output the computed lists $\mathcal{L}^{(0)}, \mathcal{L}^{(1)}, \ldots, \mathcal{L} = \mathcal{L}^{(\mathsf{Q})}$.

Figure 6: Routine CCA.AdviceList

$\mathsf{CCA.Val}(\mathsf{com}) \to m \cup \perp$

1. Set $\tilde{s} = \mathsf{CCA.FindSeed}(\mathsf{com.aux})$.
2. If $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 0$ output $\perp$.
3. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, 0)$.

$\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}) \to m \cup \perp$

1. If $\mathsf{com.tag} = \mathsf{tag}^*$, output $\perp$.
2. Let x^* be the smallest index where the bits of $\mathsf{tag}^*, \mathsf{com.tag}$ differ.
3. Set $\tilde{s} = \mathsf{CCA.FindAlt}(x^*, \mathsf{com}, \mathcal{L})$.
4. If $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 0$ output $\perp$.
5. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, 0)$.

$\mathsf{CCA.Recover}(\mathsf{com}, r) \to m \cup \perp$

1. From r, parse the seed s of the Hinting PRG.
2. If $\mathsf{CCA.Check}(s, \mathsf{com}) = 0$, output $\perp$.

3. From com, parse the commitment component c and the public parameter HPRG.pp.
4. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$.

Efficiency

Claim 6.1. If $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ is 2^{κ^v}-efficient CCA commitment scheme as per Definition 3.2 with tag space $N(\kappa) \in \mathsf{poly}(\kappa)$, $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is an efficient equivocal commitment scheme as per Definition 4.3, v, v' are constants where $v \geq 1$, then $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is $2^{\kappa^{v'}}$-efficient CCA commitment scheme. Moreover, there exists a *non-uniform* algorithm CCA.ValAlt that runs in time $\mathsf{poly}(|m|, 2^{\kappa^v})$.

Proof.

- CCA.Com calls Small.Com $2 \cdot n \cdot N$ times on the output of $\mathsf{Equiv.Com}(1^{\kappa'}, \cdot)$ in addition to some other poly-time computation. By Definition 3.2, Small.Com is $\mathsf{poly}(|m|, \kappa)$. Since Equiv.Com runs in time $\mathsf{poly}(\kappa')$ by Definition 4.3, this bounds the message of Small.Com with $\mathsf{poly}(\kappa)$. Along with the fact that n is bounded by the security parameter, and N is bounded by the tag space which we assume is $\mathsf{poly}(\kappa)$, this is overall polynomial bounded in κ.
- CCA.Recover does a single $\oplus$, in addition to invoking CCA.Check which computes Small.Recover, Small.Com and AuxEquiv.Decom a total of $n \cdot N$ times and some additional polynomial-time computation. Since com, n, N and r are all bounded by $\mathsf{poly}(\kappa)$ by the runtime of CCA.Com, CCA.Recover runs in $\mathsf{poly}(|m|, \kappa)$ as well.
- CCA.Val checks every possible seed in $\{0,1\}^n$, i.e. runs in time 2^n where n is in $\mathsf{poly}(\kappa'') = \Theta(\kappa''^{\tilde{v}}) = \Theta(\kappa^{v'})$. Thus the oracle is $2^{\kappa^{v'}}$-efficient from Definition 3.2.
- Additionally, CCA.ValAlt non-uniformly calls CCA.FindAlt that performs a check on a polynomial size list $\mathcal{L}$ and calls Small.Val, $n = \mathsf{poly}(\kappa)$ times. As Small.Val runs in time 2^{κ^v}, thus the runtime is $\mathsf{poly}(|m|, 2^{\kappa^v})$ where $v \geq 1$.

□

Correctness

Claim 6.2. If $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ is a correct CCA commitment scheme as per Definition 3.1 and $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom})$ is a correct equivocal commitment scheme as per Definition 4.2, hinting PRG satisfies the properties Definition 5.3, Definition 5.2 then $(\mathsf{CCA.Com}, \mathsf{CCA.Decom}, \mathsf{CCA.Val})$ is a correct CCA commitment scheme.

Proof. By the fact that the hinting PRG sets up parameters that are injective from Definition 5.3, Definition 5.2 implies that the same seed is output by CCA.FindSeed as was constructed by CCA.Com.

Note that if base scheme is correct, then $\forall i \in [n], x \in [N], b \in \{0,1\}$,

$$\mathsf{Small.Val}(\mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, b), y_i; r)) = y_i.$$

Also from correctness of equivocal scheme, $\forall i \in [n]$, and $\mathsf{aux} \in \{0,1\}^*$,

$$\mathsf{Equiv.Decom}(\mathsf{aux}, \mathsf{Equiv.Com}(1^{\kappa'}, \mathsf{aux}, s_i)) = s_i.$$

On input s, $\forall i \in [n], x \in [N]$, correctly sets the randomness along c_{x,i,s_i} and $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) = m$. We can observe that the scheme is correct. Observe that our hinting PRG block length needs to be long enough to use as Small.Com randomness, where Small.Com is used to commit decommitment strings from AuxEquiv. However, somewhat circularly, we require AuxEquiv to be $K(|\mathsf{advice}|) = \frac{|\mathsf{advice}|}{2n'}$ binding, where n' includes the seed and public parameters of the hinting PRG. Thus, the succinctness property of our hinting PRG is critical in removing this dependence of seed (and public parameter) length on block length. □

Binding Note that CCA.Val checks all possible candidate seeds, and if for a seed s, CCA.Check passes, it outputs $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$ otherwise outputs $\perp$. Similarly $\mathsf{CCA.Recover}(\mathsf{com}, s)$ given a candidate seeed s outputs $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$ if CCA.Check passes otherwise outputs $\perp$. By perfect injectivity of the injective HPRG extension, there is at most one candidate seed for which CCA.Check passes. This implies that if CCA.Check passes, there is a unique seed, and therefore, $\forall c, \forall m_0, m_1 \in \{0,1\}^w$ s.t. $m_0 \neq m_1$, there does not exist r such that

$$\mathsf{CCA.Recover}(\mathsf{com}, r) = m_0, \mathsf{CCA.Val}(\mathsf{com}) \in \{m_1, \perp\}$$

Moreover, if $\mathsf{CCA.Val}(\mathsf{com}) = m_1 \neq \perp$, there must exist a candidate seed s for which CCA.Check passes, and thus $\mathsf{CCA.Recover}(\mathsf{com}, s)$ must equal m_1.

Recovery from Randomness It is easy to see that the above scheme also satisfies the recovery from randomness property by correctness of CCA.Check, and as

$$c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) = m \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) = m.$$

6.1 Proof of Security

We now prove security by showing a sequence of games that prove that we can transform a 2^{κ^c}-subexponentially CCA secure 2^{κ^v}-efficient scheme on a small tag space to a 2^{κ^c}-subexponentially CCA secure $2^{\kappa^{v'}}$-efficient scheme on a larger tag space.

The argument proceeds through a sequence of games where we first change how we answer the decommitment queries to the adversary by using CCA.ValAlt instead of CCA.Val. We perform the change such that we generate a list $\mathcal{L}$ of most frequently used non-uniform queries using the procedure CCA.AdviceList. This list (dependent on the adversary's algorithm and advice) is given to CCA.ValAlt as advice. We argue that the adversary's behavior cannot change depending on this modification by invoking the non-uniform security of the equivocal commitment scheme. We utilize two main lemmas that rely on the injectivity of HPRG.ExtEval and the security of AuxEquiv to show that the probability an adversary equivocates on a query outside this list is negligible. Our final lemma will show that it is not possible for the adversary to submit a query where it doesn't equivocate but can notice a change between the two oracles.

After we have switched to CCA.ValAlt, we can rely on the non-uniform security of the base CCA secure scheme and add equivocations to hide the information about the hinting PRG seed. Finally, we will leverage on the hinting PRG security to remove the message that is committed by changing the evaluation $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0)$ at block 0 to random. Note that the reduction in this step needs to run CCA.ValAlt that contains the Small.Val procedure and invoke the security of the hinting PRG. The hinting PRG is $2^{\kappa''^{\gamma}} = 2^{\kappa^{\frac{v}{\delta}}}$-subexponentially secure while the Small.Val

procedure needs to run in time 2^{κ^v}. Since $\delta \in (0,1)$, the reduction goes through here. If we didn't have complexity leveraging between Small.Val and CCA.Val (which is brute forcing through the hinting PRG seeds), this reduction would not go through.

Theorem 6.1. Let (Small.Com, Small.Val, Small.Recover) be a 2^{κ^c}-subexponentially secure, 2^{κ^v}-efficient CCA commitment scheme for $N'(\kappa) = N' = 4N \in \mathsf{poly}(\kappa)$ tags and message space $u(\kappa) \in \mathsf{poly}(\kappa)$ where $c < 1$ and $v \geq 1$. Let (Setup, Eval, ExtEval, CheckParams) be a $T = 2^{\kappa^\gamma}$ secure hinting PRG with injective extension for some constant $\gamma \in (0,1)$ and (AuxEquiv.Com, AuxEquiv.Decom, AuxEquiv.Equivocate) be $T = 2^{\kappa^\delta}$-binding secure and statistically hiding setupless equivocal commitment where $\delta \in (0,1)$. Then the above commitment scheme (CCA.Com, CCA.Val) is a non-uniform 2^{κ^c}-subexponentially CCA secure for 2^N tags that is $2^{\kappa^{v'}}$-efficient and commits to messages of length $w(\kappa) \in \mathsf{poly}(\kappa)$ where $v' = \frac{v \cdot \tilde{v}}{\delta \cdot \gamma}$ and $\Theta(\kappa^{\tilde{v}})$ denote the length of the seed in relation to the security parameter κ.

Proof. We first define our sequence of games. Then for each adjacent set of games we prove that the advantage of any non-uniform attacker $\mathcal{A}$ that runs in time $\mathsf{poly}(2^{\kappa^c})$ must be negligibly close.

Game 0. This is the original message hiding game between a challenger and a non-uniform attacker for 2^{κ^c}-subexponentially secure adversaries. The game is parameterized by a security parameter κ.

1. The attacker sends a "challenge tag" $\mathsf{tag}^* \in \{0,1\}^N$.
2. **Pre Challenge Phase:** The attacker makes repeated commitment queries

$$\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N]})_{i \in [n]}\right).$$

   If $\mathsf{tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

$$\mathsf{CCA.Val}(\mathsf{com}).$$

3. **Challenge Phase**
   (a) The attacker sends two messages $m_0^*, m_1^* \in \{0,1\}^w$
   (b) **Part 1:**
   - Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
   - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Sample $(\mathsf{HPRG.pp}^*, n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(|m|, N \cdot \ell)})$.
   - Sample $s^* = s_1^* \dots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
   - Set $\mathsf{aux}^* = (\mathsf{HPRG.pp}^*, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}^*, s^*))$.
   - For all $i \in [n]$ run $\mathsf{AuxEquiv.Com}(1^{\kappa'}, \mathsf{aux}^*, s_i^*) \to (\sigma_i^*, y_i^*)$.
   - Let $r_{x,i}^*, \tilde{r}_{x,i}^* \in \{0,1\}^\ell$ be defined as follows:
   - For $i \in [n]$
     i. Compute $(r_{1,i}^*, r_{2,i}^*, \dots, r_{N,i}^*) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
     ii. Sample $(\tilde{r}_{1,i}^*, \tilde{r}_{2,i}^*, \dots, \tilde{r}_{N,i}^*) \xleftarrow{R} \{0,1\}^{N \cdot \ell}$

(c) **Part 2:**

- It chooses a bit $b \in \{0,1\}$ and sets $c^* = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m_b^*$.
- For $i \in [n]$, $x \in [N]$
  - i. If $s_i^* = 0$
    - A. $c_{x,i,0}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 0), y_i^*; r_{x,i}^*)$
    - B. $c_{x,i,1}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 1), y_i^*; \tilde{r}_{x,i}^*)$
  - ii. If $s_i^* = 1$
    - A. $c_{x,i,0}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 0), y_i^*; \tilde{r}_{x,i}^*)$
    - B. $c_{x,i,1}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 1), y_i^*; r_{x,i}^*)$
- Finally, it sends $\mathsf{com}^* = \left(\mathsf{tag}^*, \mathsf{aux}^*, c^*, (\sigma_i^*, \left(c_{x,i,0}^*, c_{x,i,1}^*\right)_{x\in[N]})_{i\in[n]}\right)$ as the commitment. All of the randomness is used as the decommitment string.

4. **Post Challenge Phase:** The attacker again makes commitment queries com. If $\mathsf{tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

$$\mathsf{CCA.Val}(\mathsf{com}).$$

5. The attacker finally outputs a guess b'.

Game 1. This is same as Game 0, except that during the **Pre Challenge Phase** and **Post Challenge Phase**, challenger using CCA.ValAlt to answer queries. Let $\mathcal{A}$ be an adversary with non-uniform advice that tries to guess the difference between the two games. The Challenger uses $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$ to return queries where $\mathcal{L}$ is generated through the procedure CCA.AdviceList.

1. **Non-uniform Computation:** The challenger generates the list $\mathcal{L}^{(0)}, \mathcal{L}^{(1)}, \ldots, \mathcal{L} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, (\mathcal{A}, \mathsf{advice}))$ by interacting with the attacker $\mathcal{A}$. It uses $\mathcal{L}$ to answer adversaries queries.
2. The attacker sends a "challenge tag" $\mathsf{tag}^* \in \{0,1\}^N$.
3. **Pre Challenge Phase:** The attacker makes repeated queries commitments com. If $\mathsf{com.tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

$$\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}).$$

4. **Challenge Phase** ....
5. **Post Challenge Phase:** Same as **Pre Challenge Phase**.
6. The attacker finally outputs a guess b'.

Game 2. In this game in **Part 1** the (σ_i^*, y_i^*) are now generated from the AuxEquiv.Equivocate algorithm instead of the AuxEquiv.Com algorithm.

- Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
- Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
- Sample $(\mathsf{HPRG.pp}^*, n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(|m|, N\cdot\ell)})$.
- Sample $s^* = s_1^* \dots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
- Set $\mathsf{aux}^* = (\mathsf{HPRG.pp}^*, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}^*, s^*))$.
- Let $r_{x,i}^*, \tilde{r}_{x,i}^* \in \{0,1\}^\ell$ be defined as follows:
- For $i \in [n]$
  1. Compute $(r_{1,i}^*, r_{2,i}^*, \dots, r_{N,i}^*) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
  2. Sample $(\tilde{r}_{1,i}^*, \tilde{r}_{2,i}^*, \dots, \tilde{r}_{N,i}^*) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$
- For all $i \in [n]$ run $\mathsf{AuxEquiv.Equivocate}(1^{\kappa'}, \mathsf{aux}^*) \rightarrow (\sigma_i^*, y_{i,0}^*, y_{i,1}^*)$.
- For all $i \in [n]$, set $y_i^* = y_{i,s_i^*}^*$.

Game 3 In this game in **Part 2** we move to $c_{x,i,0}^*$ committing to $y_{i,0}^*$ and $c_{x,i,1}^*$ committing to $y_{i,1}^*$ for all $x \in [N], i \in [n]$ independently of the value of s_i^*.

- It chooses a bit $b \in \{0,1\}$ and sets $c^* = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m_b^*$.
- For $i \in [n]$, $x \in [N]$
  1. If $s_i^* = 0$
     (a) $c_{x,i,0}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 0), y_{i,0}^*; r_{x,i}^*)$
     (b) $c_{x,i,1}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 1), y_{i,1}^*; \tilde{r}_{x,i}^*)$
  2. If $s_i^* = 1$
     (a) $c_{x,i,0}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 0), y_{i,0}^*; \tilde{r}_{x,i}^*)$
     (b) $c_{x,i,1}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x^*, 1), y_{i,1}^*; r_{x,i}^*)$
- Finally, it sends $\mathsf{com}^* = \left(\mathsf{tag}^*, \mathsf{aux}^*, c^*, (\sigma_i^*, \left(c_{x,i,0}^*, c_{x,i,1}^*\right)_{x\in[N]})_{i\in[n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

Game 4. In this game c^* is chosen uniformly at random (instead of choosing $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m_b^*$).

6.2 Analysis

Next, we show by a sequence of lemmas that no non-uniform adversary with runtime $\mathsf{poly}(2^{\kappa^c})$ where $c \in (0,1)$ can distinguish between any two adjacent games with non-negligible advantage. In the last game, we show that the advantage of any such adversary is negligible. We will let $\mathsf{adv}_{\mathcal{A}}^x$ denote the quantity $\Pr[b' = b] - \frac{1}{2}$ in Game x when interacting with adversary $\mathcal{A}$. Let $\mathcal{A}$ make $\mathsf{Q} = \mathsf{Q}(\kappa)$ queries in the **Pre Challenge Phase** and **Post Challenge Phase**.

The first and possibly most involved part of this analysis involves arguing that the adversary's distinguishing advantage between Games 0 and 1 is negligible. These games differ in the way the

oracle opens up the adversary's commitment queries. In the following subsection, we develop a set of useful lemmas that will help us argue that Games 0 and 1 are indistinguishable. In the following subsection, we will use these lemmas and develop additional lemmas that allow us to show that no non-uniform adversary with runtime $\mathsf{poly}(2^{\kappa^c})$ where $c \in (0,1)$ can distinguish between Games 0 and 3 with non-negligible advantage.

6.2.1 Valuation Mode Switching

Recall that in Game 0 decommitment queries on com are answered with $\mathsf{CCA.Val}(\mathsf{com})$. Whereas in Game 1 they are answered by invoking the algorithm $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$ where $\mathcal{L}$ is a list of (aux, s) pairs that tell the alternative valuation algorithm which seed value to try for commitments associated with aux values on the list.

The proof can be broken down into two main parts. First, we show that for any query to com which is either (A) not equivocating or (B) has an aux value on the generated list $\mathcal{L}$, the response to evaluating com whether using CCA.Val or CCA.ValAlt is the same. The proof for this is provided immediately below and culminates in Lemma 6.1, which follows from a combination of the logic of [KW19] for (A) and properties in the injective extension of the Hinting PRG for (B).

The second part of the proof is more involved, and will show that it is only with negligible probability that an attacker produces a problematic query not matching (A) or (B). Combining these two parts helps us show that no attacker can distinguish between Game 0 and Game 1.

First Part of the Proof.

Definition 6.1. For an adversary $\mathcal{A}$, let $\mathcal{L}^{(0)}, \mathcal{L}^{(1)}, \ldots, \mathcal{L} = \mathcal{L}^{(\mathsf{Q})} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, \mathcal{A})$. For all $j \in [\mathsf{Q}]$, we denote by the set $\mathcal{W}^{(j)} = \{\mathsf{aux} \mid \exists s \in \{0,1\}^n, (\mathsf{aux}, s) \in \mathcal{L}^{(j)}\}$ as the aux values that are stored in $\mathcal{L}^{(j)}$.

Observation 1. For all polynomials $p(\cdot)$ and any list $\mathcal{L}^*$ with length at most p, for all algorithms $\mathcal{A}$ running in time $\mathsf{poly}(2^{\kappa^v})$ and making a polynomial number $p'(\kappa)$ queries, $j \in p'(\kappa)$, $\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^*)$ runs in time $\mathsf{poly}(2^{\kappa^v})$.

Proof. Observe that CCA.Exp simply runs $\mathcal{A}$, which runs in time $\mathsf{poly}(2^{\kappa^v})$ time, and answers a polynomial number of com queries. To run CCA.Equiv (then running CCA.ValAlt) requires $O(n \cdot N) \in \mathsf{poly}(\kappa)$ calls to Small.Val, which by 2^{κ^v}-efficincy of Small, requires $\mathsf{poly}(2^{\kappa^v})$ time. Hence, the entire procedure can be run in $\mathsf{poly}(2^{\kappa^v})$ time. □

Claim 6.3. Let $\mathcal{L}^*$ be any list of $\{(\mathsf{aux}_i, \mathsf{CCA.FindSeed}(\mathsf{aux}_i))\}_i$ pairs. If $(\mathsf{com.aux}, \_) \in \mathcal{L}^*$, then, $\mathsf{CCA.Val}(\mathsf{com}) = \mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}^*)$ - i.e. the oracle outputs are identical on such com.

Proof. Recall any (aux, s) pair in $\mathcal{L}$ is computed from aux, $\mathsf{CCA.FindSeed}(\mathsf{aux})$, so $\mathsf{CCA.FindAlt}(\mathsf{com})$ will return $\mathsf{CCA.FindSeed}(\mathsf{aux})$, from which CCA.Val and CCA.ValAlt are computed identically. □

Claim 6.4. Let $\mathcal{L}^*$ be any list of $\{(\mathsf{aux}_i, \mathsf{CCA.FindSeed}(\mathsf{aux}_i))\}_i$ pairs. For any commitment com such that $\mathsf{CCA.Equiv}(\mathsf{com}) = \bot$, $\mathsf{CCA.Val}(\mathsf{com}) = \mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}^*)$.

Proof. We perform a case analysis. Let x^* be the smallest index where the bits of $\mathsf{tag}^*, \mathsf{tag}$ differ. If $\mathsf{CCA.FindSeed}(\mathsf{com.aux}) = \mathsf{CCA.FindAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$, then we are done as both oracles behave identically given the same candidate seed. Let the candidate seeds be different. Consider the following cases.

- Case 1: $\mathsf{CCA.FindSeed}(\mathsf{com.aux}) = s^*, \mathsf{CCA.FindAlt}(x^*, \mathsf{com}, \mathcal{L}^*) = s \neq s^*$ and $s^* \neq \perp, s \neq \perp$.
  Let i be the first index where s and s^* differ and let s_i, s_i^* be the bits at those respective positions. Since CCA.FindSeed returned s^*, we know that $\mathsf{com.aux} = (\mathsf{HPRG.pp}, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s^*))$. If $\mathsf{HPRG.CheckParams}(\mathsf{HPRG.pp}, n) = 0$, then check procedures in both CCA.Val and CCA.ValAlt fail and they both output $\perp$.

  Else since $\mathsf{HPRG.CheckParams}(\mathsf{HPRG.pp}, n) = 1$, from Definition 5.2 and Claim 6.3, there cannot exist another $s \neq s^*$ such that $\mathsf{com.aux} = (\mathsf{HPRG.pp}, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s))$. Thus the check after CCA.FindAlt must fail and it returns $\perp$. If the check after CCA.FindSeed fails then we are done as both procedures output $\perp$. Thus assume that the check after CCA.FindSeed does not fail.

  - **If $\mathbf{s_i^* = 1}$ and $\mathbf{s_i = 0}$**, we have from CCA.FindAlt that, $\mathsf{AuxEquiv.Decom}(\mathsf{com.aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x^*,i,0})) = 0$. But from running CCA.Check after CCA.FindSeed and not failing we have that, $\mathsf{AuxEquiv.Decom}(\mathsf{com.aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x^*,i,1})) = 1$ (the Small.Recover and Small.Val procedure open similarly as the check does not fail). Thus $\mathsf{CCA.Equiv}(\mathsf{com}) \neq \perp$, i.e. it is an equivocating query and this is not possible.
  - **If $\mathbf{s_i^* = 0}$ and $\mathbf{s_i = 1}$**, we have from CCA.FindAlt that, either $\mathsf{AuxEquiv.Decom}(\mathsf{com.aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x^*,i,0})) = 1$ or $\mathsf{AuxEquiv.Decom}(\mathsf{com.aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x^*,i,0})) = \perp$. From check after CCA.FindSeed we have that, $\mathsf{AuxEquiv.Decom}(\mathsf{com.aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x^*,i,0})) = 0$ (the Small.Recover and Small.Val procedure open similarly as the check does not fail). As $\mathsf{AuxEquiv.Decom}(\mathsf{com.aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x^*,i,0}))$ are deterministic computations evaluating to different outputs, this is clearly not possible.

- Case 2: $\mathsf{CCA.FindSeed}(\mathsf{com.aux}) = \perp, \mathsf{CCA.FindAlt}(x^*, \mathsf{com}, \mathcal{L}^*) = s$.
  Since $\mathsf{CCA.FindSeed}(\mathsf{com.aux}) = \perp$, we know $\forall \tilde{s} \in \{0,1\}^n$, $\mathsf{com.aux} \neq (\mathsf{HPRG.pp}, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, \tilde{s}))$. Clearly CCA.Val outputs $\perp$.

  Since this holds for all seeds, it also holds for seed s found by CCA.FindAlt, thus $\mathsf{com.aux} \neq (\mathsf{HPRG.pp}, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s))$, so which means CCA.ValAlt will return $\perp$ when performing CCA.Check.

- Case 3: $\mathsf{CCA.FindSeed}(\mathsf{com.aux}) = s, \mathsf{CCA.FindAlt}(x^*, \mathsf{com}, \mathcal{L}^*) = \perp$.
  CCA.FindAlt cannot return $\perp$, so this is impossible.

□

Lemma 6.1. Let $\mathcal{L}^*$ be any list of $\{(\mathsf{aux}_i, \mathsf{CCA.FindSeed}(\mathsf{aux}_i))\}_i$ pairs. For all $\mathsf{com}^*, \mathsf{tag}^*$, if $\mathsf{CCA.Val}(\mathsf{com}^*) \neq \mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}^*, \mathcal{L}^*)$, then $\mathsf{CCA.Equiv}(\mathsf{com}^*) \neq \perp$ and $(\mathsf{com}^*.\mathsf{aux}, \_) \notin \mathcal{L}^*$

Proof. This follows directly from Claim 6.3 and Claim 6.4. □

Second Part of the Proof: Avoiding Bad Queries. The alternative valuation algorithm uses a list $\mathcal{L}$ to lookup the seeds associated with any aux value on the list, where this list $\mathcal{L}$ is defined as in Figure 6. In this second part of the proof, we want to argue that only with negligible probability will there be a query com that is both equivocating and that has an aux value not on this list.

In doing this, our goal is to use the binding security of the auxiliary input equivocal commitment scheme. We will use this property to argue that any attacker with a polynomial bounded amount of non-uniform advice will have a bounded number of aux values that it can make equivocating queries for where the number of such values is proportional to the amount of advice. Completing this argument faces the following challenge. Any reduction algorithm to the auxiliary input equivocal commitment scheme will not have sufficient running time to run $\mathsf{CCA.Val}(\mathsf{com})$ and thus will need to use a list itself to answer decryption queries. The reduction algorithm $\mathcal{R}$ thus will need to carry both this list as well as $\mathcal{A}$'s advice as its own advice string. Thus, to have a valid reduction, the size of the total advice of $\mathcal{R}$ dictates the number of equivocating responses that must be produced to violate the binding game. It is for this reason we need an underlying auxiliary input equivocal commitment scheme where the binding security is $\frac{|\mathsf{advice}|}{2n'}$ where n' is the length of aux plus length of the hinting PRG seed.

In addition, the most common equivocal queries when answering with $\mathsf{CCA.Val}(\mathsf{com})$ might be different than the most common ones when answering from a list. Thus, we need to be careful about how we define our list $\mathcal{L}$. Notice from Figure 6 that this is done recursively. Namely, we set $\mathcal{L}^{(0)}$ to be empty, then define $\mathcal{L}^{(j)}$ recursively for every $j \in [1, Q]$ as a function of $\mathcal{L}^{(j-1)}$, and finally set $\mathcal{L} = \mathcal{L}^{(Q)}$ where recall that $Q = Q(\kappa)$ is the total number of decommitment queries that the adversary makes. The list $\mathcal{L}^{(j)}$ is defined as the list of the most common equivocating aux values made over the first j queries when the first j queries are answered by using the $\mathsf{CCA.ValAlt}$ algorithm with list $\mathcal{L}^{(j-1)}$.

Recall that our goal is to show that the probability of a query com that is equivocating, but has an aux value not on $\mathcal{L}^{(Q)}$, is negligible. We break this proof down into two parts.

First, for $j \in [1, Q]$, let $\mathcal{W}^{(j)}$ be the set of all aux values in the list $\mathcal{L}^{(j)}$. In the upcoming lemma (Lemma 6.2), we will show that for $j \in [1, Q]$, the probability of an equivocating query in the first j queries with an aux value not in $\mathcal{W}^{(j)}$ is negligible when the first j queries are answered using $\mathcal{L}^{(j-1)}$. This is done by two reductions to auxiliary input equivocal binding security. The first shows that the maximum probability that any *particular* $\mathsf{aux} \notin \mathcal{W}^{(j)}$ appears in an equivocating com is negligible. Then we use this fact along with a second reduction to show that the probability that any $\mathsf{aux} \notin \mathcal{W}^{(j)}$ appears in an equivocating com is negligible. In the latter case, a union bound does not suffice due to there being an exponential number of such aux values, so we must use another reduction.

Next, in Lemma 6.3 (and its Corollary 6.1) – which is the main result of this subsubsection, we show that for $j \in [1, q]$, the probability of an equivocating query in the first j queries with an aux value not in $\mathcal{W}^{(j)}$ is negligible when the first j queries are answered using $\mathcal{L}^{(j)}$ (as opposed to $\mathcal{L}^{(j-1)}$, which was what appeared in the statement of Lemma 6.2). This is done by induction on j, and via the use of Lemma 6.2.

Lemma 6.2. Suppose $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is $(2^{\kappa^\delta}, \frac{|\mathsf{advice}|}{2n'})$ binding secure (Definition 4.4) auxiliary input equivocal commitment scheme. Then, for any non-uniform adversary $\mathcal{A}$ running in time $\mathsf{poly}(2^{\kappa^v})$ where $v \geq 1$ and making $\mathsf{Q} = \mathsf{poly}(\kappa)$ queries, letting $\mathcal{L}^{(1)}, \ldots \mathcal{L}^{(\mathsf{Q})} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, \mathcal{A})$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that for all $\kappa \in \mathbb{N}$,

$$\forall j \in [\mathsf{Q}] \quad \Pr_{r_{\mathcal{A}}, r_{\mathcal{C}}}\left[\exists \mathsf{aux} \in \{0,1\}^{n'-n} \backslash \mathcal{W}^{(j)} : (\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)}; r_{\mathcal{A}}, r_{\mathcal{C}})\right] \leq \mathsf{negl}(\kappa)$$

Proof. First consider a weaker, restricted claim:

Claim 6.5. Suppose $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is $(2^{\kappa^\delta}, \frac{|\mathsf{advice}|}{2n'})$ binding secure auxiliary input equivocal commitment scheme. Then, for any non-uniform adversary $\mathcal{A}$ running in time $\mathsf{poly}(2^{\kappa^v})$, there exists a negligible function $\mathsf{negl}(\kappa)$ such that for all $\kappa \in \mathbb{N}$,

$$\max_{\mathsf{aux} \in \{0,1\}^{n'-n} \setminus \mathcal{W}^{(j)}} \left(\Pr \left[(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)}) \right] \right) \leq \mathsf{negl}(\kappa)$$

Proof. Assume for sake of contradiction that there is an adversary $\mathcal{A}$ and function $j = j(\kappa)$ such that there exists a polynomial $r(\kappa)$ for which the above event occurs with probability $\geq \frac{1}{r(\kappa)}$ for infinitely many κ. Then consider the following reduction (which takes non-uniform advice $\mathcal{L}^{(j-1)}$ produced from $\mathsf{CCA.AdviceList}(1^\kappa, \mathcal{A})$ as well as the non-uniform advice of $\mathcal{A}$):

Reduction $\mathcal{R}_{I,\mathcal{A},j}(1^{\kappa'})$:

- **Non Uniform Computation:** Receive list $\mathcal{L}^{(j-1)}$ and the advice taken by algorithm $\mathcal{A}$ as non-uniform advice. Assume it receives $j(\kappa)$ as advice as well. [8]
- For $i \in [I]$:
  - Let $\mathcal{E}_i \leftarrow \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)})$.
- Let ζ be a subset of $(\mathsf{aux}, c, d_0, d_1)$ entries from $\bigcup_i \mathcal{E}_{,i}$, where only a single entry for each unique aux value is kept.
- Output ζ.

Claim 6.6. For all $I \in \mathsf{poly}(\kappa)$, $\mathcal{R}_I$ is a non-uniform algorithm with $|\mathsf{advice}_\mathcal{R}| = 2 \cdot |\mathsf{advice}_\mathcal{A}|$ bits of advice and runs in time $\mathsf{poly}(2^{\kappa'^\delta}) = \mathsf{poly}(2^{\kappa^v})$.

Proof. Observe that $\mathcal{R}$ takes in $\mathcal{A}$ which uses non-uniform advice $\mathsf{advice}_\mathcal{A}$, and $\mathcal{L}^{(j-1)}$, which contains $\frac{\mathsf{advice}_\mathcal{A}}{n'}$ entries of length n', making the total non-uniform advice $2 \cdot |\mathsf{advice}_\mathcal{A}|$. Since it runs $\mathsf{CCA.Exp}$ a polynomial number of times, and $\mathsf{CCA.Exp}$ runs in time $\mathsf{poly}(2^{\kappa^v})$ by Observation 1, $\mathcal{R}$ does as well. □

To prove Claim 6.5, we will set $I = \kappa \cdot r(\kappa)$

Fact 6.1. For $I = \kappa \cdot r(\kappa)$, $\mathcal{R}_I$ outputs K entries $\{(\mathsf{aux}^{(i)}, c^{(i)}, d_0^{(i)}, d_1^{(i)})\}_{i \in [K]}$ such that

$$\begin{gathered} \forall i \in [K], \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_0^{(i)}) = 0, \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_1^{(i)}) = 1 \\ \forall i \neq \iota \in [K], \mathsf{aux}^{(i)} \neq \mathsf{aux}^{(\iota)} \end{gathered}$$

for a $K > \frac{|\mathsf{advice}_\mathcal{R}|}{2n'}$ with non-negligible probability.

[8] We're non-uniformly given some list $\mathcal{L}^{(j-1)}$, but we don't know the explicit j as a function of κ. To be super explicit in our reduction, we include in the non uniform advice. Additionally, we ignore this accounting of storing $\mathsf{poly}(\log(\kappa))$ bits in our advice as this can be brute forced in polynomial time and isn't required explicitly in the advice. For the sake of intelligibility, we don't list the brute forcing.

Proof. First, notice that since in our construction, we picked $\mathcal{L}^{(j)}$ in CCA.AdviceList to maximize appearance in CCA.Exp,

$$\max_{\mathsf{aux} \in \{0,1\}^{n'-n} \setminus \mathcal{W}^{(j)}} \left(\Pr\left[(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)}) \right] \right) \geq \frac{1}{r(\kappa)}$$

then

$$\forall \mathsf{aux} \in \mathcal{W}^{(j)} \ \Pr\left[(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)}) \right] \geq \frac{1}{r(\kappa)}$$

Thus, since each $\mathcal{E}_{j,i}$ is generated independently, we can upper bound the probability an $\mathsf{aux} \in \mathcal{W}^{(j)}$ doesn't appear in any such $\mathcal{E}_{j,i}$ as,

$$\Pr\left[(\mathsf{aux}, \_, \_, \_) \notin \bigcup_{i \in [I]} \mathcal{E}_{j,i} \right] \leq \left(1 - \frac{1}{r(\kappa)}\right)^{r(\kappa)\cdot\kappa} \leq e^{-\kappa}$$

Since ξ contains all the unique aux values of $\bigcup_i \mathcal{E}_{j,i}$, by union bound, the probability that there are at least $|\mathcal{W}^{(j)}|+1$ entries in the set ξ is $\geq 1 - \frac{|\mathcal{W}^{(j)}|+1}{e^\kappa}$. Recall that since by construction $\mathcal{W}^{(j)}$ has $\frac{|\mathsf{advice}_\mathcal{A}|}{n'} = \frac{\mathsf{advice}_\mathcal{R}}{2n'}$ entries, which complete our proof. Since by assumption, $\mathcal{A}$ is successful with probability $\geq \frac{1}{r(\kappa)}$ infinitely often, our reduction is as well. □

Since $\mathcal{R}$ breaks the security of the auxilary input equivocal commitment scheme from Definition 4.4, this completes our proof of Claim 6.5.

□

Now we return to proving our original lemma statement. Note that using Claim 6.5 and simply union bounding over all $\mathsf{aux} \notin \mathcal{W}^{(j)}$ is not strong enough to claim this statement, as there are exponentially many such values.

Assume for sake of contradiction that for some $j = j(\kappa) \in (\kappa : \mathbb{N} \to \mathsf{Q}(\kappa))$, there exists polynomial $t(\kappa)$ for which there exists some commitments such that $\mathcal{B}$ queries $\mathsf{aux}^* \notin \mathcal{W}^{(j)}$ with probability $> \frac{1}{t(\kappa)}$ for infinitely many κ.

Recall reduction $\mathcal{R}_I(1^{\kappa'})$ from Claim 6.5. We will prove the full lemma using the same $\mathcal{R}$ for $I = t(\kappa) \cdot \kappa \cdot \frac{|\mathsf{advice}_\mathcal{R}|}{2n'}$, which still runs in $\mathsf{poly}(2^{\kappa^v})$ time by Claim 6.6.

Fact 6.2. $I = t(\kappa) \cdot \kappa \cdot \frac{|\mathsf{advice}_\mathcal{R}|}{2n'}$, $\mathcal{R}_I$ outputs K entries $\{(\mathsf{aux}^{(i)}, c^{(i)}, d_0^{(i)}, d_1^{(i)})\}_{i \in [K]}$ such that

$$\begin{gathered} \forall i \in [K], \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_0^{(i)}) = 0, \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_1^{(i)}) = 1 \\ \forall i \neq \iota \in [K], \mathsf{aux}^{(i)} \neq \mathsf{aux}^{(\iota)} \end{gathered}$$

for a $K \geq \frac{|\mathsf{advice}_\mathcal{R}|}{2n'}$ with non-negligible probability.

Proof. Let $K = \frac{|\mathsf{advice}_\mathcal{R}|}{2n'}$. We can partition the tuples output by $\mathcal{R}$ into K 'groups' of size $t(\kappa) \cdot \kappa$, where the first set of $t(\kappa) \cdot \kappa$ tuples are in the first group, the next set of $t(\kappa) \cdot \kappa$ tuples are in the second group, and so on. Notice by the same analysis as before that the probability an element

not in $\mathcal{W}^{(j)}$ is queried in each 'group' is $1 - \left(1 - \frac{1}{t(\kappa)}\right)^{t(\kappa)\cdot\kappa} \geq 1 - e^{-\kappa}$ for infinitely many κ. From a union bound, the probability that there exists a group where we fail to query an element not in $\mathcal{W}^{(j)}$ is $K \cdot e^{-\kappa}$.

From Claim 6.5, let any particular element $\notin \mathcal{W}^{(j)}$ occur with probability at most $\upsilon(\kappa)$ where υ is some negligible function. For $i \in [K]$, let $\mathsf{aux}^{(i)} \notin \mathcal{W}^{(j)}$ be the aux value in the equivocating query for Group i. For any $i, \iota \in [K]$, where $i \neq \iota$, the probability that $\mathsf{aux}^{(i)} = \mathsf{aux}^{(\iota)} \notin \mathcal{W}^{(j)}$ is $\leq \upsilon(\kappa)$. Because fixing $\mathsf{aux}^{(i)} \in \{0,1\}^{n'-n}$, the probability that any query in a different group is equal to this value when $\mathsf{aux}^{(i)} \notin \mathcal{W}^{(j)}$ is $\upsilon(\kappa)$.

Probability that there exists $i, \iota \in [K]$ where $i \neq \iota$ and $\mathsf{aux}^{(i)} = \mathsf{aux}^{(\iota)}$ and $\mathsf{aux}^{(i)} \notin \mathcal{W}^{(j)}$ is $\leq \left(\frac{K\cdot(K-1)}{2}\right) \upsilon(\kappa)$.

Thus, we output $< K$ pairs with probability at most $K \cdot e^{-\kappa} + \left(\frac{K\cdot(K-1)}{2}\right) \upsilon(\kappa)$. Since the advice is polynomial in κ, $K = \mathsf{poly}(\kappa)$ and υ is a negligible function, we have that we fail with negligible probability.

□

Since $\mathcal{R}$ breaks the security of the auxilary input equivocal commitment scheme from Definition 4.4, we have that for every adversary $\mathcal{A}$, $j \in [\mathsf{Q}]$, there exists a negligible function such that the probability $\mathcal{A}$ queries something outside the list $\mathcal{W}^{(j)}$ in experiment $\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)})$ is negligible.

To complete the lemma proof, we need to argue that for every adversary $\mathcal{A}$, there's a negligible function such that for all $j \in [\mathsf{Q}]$, the probability $\mathcal{A}$ queries something outside the list $\mathcal{W}^{(j)}$ in experiment $\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)})$ is negligible. Consider setting the function

$$j^*(\kappa) = \mathsf{argmax}_{j\in[\mathsf{Q}(\kappa)]} \Pr\left[\exists \mathsf{aux} \in \{0,1\}^{n'-n} \backslash \mathcal{W}^{(j)} : (\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j-1)})\right]$$

By the claim above, this is negligible, and upper bounds the probability $\forall j \in [\mathsf{Q}(\kappa)]$, completing our proof of Lemma 6.2.

□

Induction Lemma 6.2 showed that it is difficult for an attacker to find equivocal queries in the first j queries that were not on list $\mathcal{L}^{(j)}$, when adversarial queries were answered using list $\mathcal{L}^{(j-1)}$. We now want to show that it is difficult for the attacker to find equivocal queries in the first j queries that are not on list $\mathcal{L}^{(j)}$, *when adversarial queries are answered using list $\mathcal{L}^{(j)}$ itself*. We will use induction to prove this, where we assume that it is difficult for the attacker to produce equivocal queries in the first $(j-1)$ queries that are not on list $\mathcal{L}^{(j-1)}$, when adversarial queries are answered using list $\mathcal{L}^{(j-1)}$, and then use 6.2 to show that the same must hold for j.

We want to show that,

$$\Pr\left[\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L} = \mathcal{L}^{(\mathsf{Q})}) : \mathsf{aux} \notin \mathcal{W}^{(\mathsf{Q})}\right] = \mathsf{negl}(\kappa).$$

We complete the proof by induction on the statement, for every $j \in \{0, \ldots, \mathsf{Q}\}$,

$$\Pr\left[\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j)}) : \mathsf{aux} \notin \mathcal{W}^{(j)}\right] = \mathsf{negl}(\kappa).$$

For our base case, $\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, 0, \emptyset)$ will never answer a query and always output $\emptyset$, trivially satisfying our claim. For our induction step, we show the following lemma.

Lemma 6.3. Assuming conditions for Lemma 6.2 hold; and the induction hypothesis, for $j \in \{0, \ldots, \mathsf{Q}-1\}$, if there exists a function $\mu^{(j)}$ such that,

$$\mu^{(j)}(\kappa) = \Pr\left[\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j)}) : \mathsf{aux} \notin \mathcal{W}^{(j)}\right],$$

then,

$$\Pr\left[\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j+1)}) : \mathsf{aux} \notin \mathcal{W}^{(j+1)}\right] \leq \mu^{(j)}(\kappa) + \nu(\kappa).$$

Proof. Note that the above probabilities are over the random coins of $\mathcal{A}$, and the challenger in experiment $\mathsf{CCA.Exp}$. Let $r_\mathcal{A}$ denote the random coins for $\mathcal{A}$ and $r_\mathcal{C}$ denote the random coins for $\mathcal{C}$.

Let $\nu(\kappa)$ be the function from Lemma 6.2 such that,

$$\Pr_{r_\mathcal{A}, r_\mathcal{C}}\left[\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j+1, \mathcal{L}^{(j)}; r_\mathcal{A}, r_C) : \mathsf{aux} \notin \mathcal{W}^{(j+1)}\right] = \nu(\kappa).$$

Let $\mathsf{aux}^\odot$ be the value stored in $\mathcal{E}$ when running $\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j+1, \mathcal{L}^{(j)}; r_\mathcal{A}, r_\mathcal{C})$ on $\mathcal{A}$'s, $j+1$-th query. If no such query is made, let $\mathsf{aux}^\odot = \perp$. Rewriting the above expression from Lemma 6.2 when split across the $j+1$-th query, we have,

$$\Pr_{r_\mathcal{A}, r_\mathcal{C}}\Big[\Big(\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j)}; r_\mathcal{A}, r_\mathcal{C}) : \mathsf{aux} \notin \mathcal{W}^{(j+1)}\Big) \bigvee \Big(\mathsf{aux}^\odot \notin \mathcal{W}^{(j+1)} \wedge \mathsf{aux}^\odot \neq \perp\Big)\Big] = \nu(\kappa).$$

From a union bound on the induction hypothesis and the above equation, we have that,

$$\Pr_{r_\mathcal{A}, r_\mathcal{C}}\Big[\Big(\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j)}; r_\mathcal{A}, r_\mathcal{C}) : \mathsf{aux} \notin \Big(\mathcal{W}^{(j+1)} \cap \mathcal{W}^{(j)}\Big)\Big) \bigvee \Big(\mathsf{aux}^\odot \notin \mathcal{W}^{(j+1)} \wedge \mathsf{aux}^\odot \neq \perp\Big)\Big] \leq \mu^{(j)}(\kappa) + \nu(\kappa).$$

We define the following two sets Good where the above bad event of querying outside $\mathcal{W}^{(j+1)} \cap \mathcal{W}^{(j)}$ or $\mathsf{aux}^\odot$ being outside $\mathcal{W}^{(j+1)}$ doesn't happen.

$$\mathsf{Good}^\odot = \Big\{(r_\mathcal{A}, r_\mathcal{C}) \ : \ \Big(\forall(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j)}; r_\mathcal{A}, r_\mathcal{C}), \mathsf{aux} \in \Big(\mathcal{W}^{(j+1)} \cap \mathcal{W}^{(j)}\Big)\Big) \bigwedge \Big(\mathsf{aux}^\odot \in \mathcal{W}^{(j+1)} \vee \mathsf{aux}^\odot = \perp\Big)\Big\}.$$

Let $\mathsf{aux}^\dagger$ be the value stored in $\mathcal{E}$ when running $\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j+1, \mathcal{L}^{(j+1)}; r_\mathcal{A}, r_\mathcal{C})$ on $\mathcal{A}$'s, $j+1$-th query. If no such query is made, let $\mathsf{aux}^\dagger = \perp$.

$$\mathsf{Good}^\dagger = \Big\{(r_\mathcal{A}, r_\mathcal{C}) \ : \ \Big(\forall(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, j, \mathcal{L}^{(j+1)}; r_\mathcal{A}, r_\mathcal{C}), \mathsf{aux} \in \Big(\mathcal{W}^{(j+1)} \cap \mathcal{W}^{(j)}\Big)\Big) \bigwedge \Big(\mathsf{aux}^\dagger \in \mathcal{W}^{(j+1)} \vee \mathsf{aux}^\dagger = \perp\Big)\Big\}.$$

Note that for any $(r_\mathcal{A}, r_\mathcal{C}) \in \mathsf{Good}^\odot$ instance, for all queries com, $\mathsf{com.aux} \in \big(\mathcal{W}^{(j+1)} \cap \mathcal{W}^{(j)}\big)$, $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}^{(j)})$ and $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}^{(j+1)})$ behave identically and thus respond

identically. Both have a seed $\tilde{s}$ stored from the procedure $\mathsf{CCA.FindSeed}(\mathsf{com.aux})$. Additionally, for this instance, $\mathsf{aux}^{\odot} = \mathsf{aux}^{\dagger}$ as the queries and responses on first j queries have been identical. $\mathsf{aux}^{\odot}$ and $\mathsf{aux}^{\dagger}$ would only be different if one of the previous queries was answered differently. Thus, $(r_{\mathcal{A}}, r_{\mathcal{C}})$ must be in $\mathsf{Good}^{\dagger}$ as well. We have, $\mathsf{Good}^{\odot} \subseteq \mathsf{Good}^{\dagger}$. By a similar argument, $\mathsf{Good}^{\dagger} \subseteq \mathsf{Good}^{\odot}$ and we have that the two sets are equal. Thus, $|\mathsf{Good}^{\dagger}| = |\mathsf{Good}^{\odot}|$.

The bad instances are thus equal when answering using $\mathcal{L}^{(j+1)}$ and we have,

$$\begin{aligned}
\Pr_{r_{\mathcal{A}}, r_C}\Big[\Big(\exists(\mathsf{aux}, _, _, _) \in \mathsf{CCA.Exp}(1^{\kappa}, \mathcal{A}, j, \mathcal{L}^{(j+1)}) : \mathsf{aux} \notin \Big(\mathcal{W}^{(j+1)} \cap \mathcal{W}^{(j)}\Big)\Big) \\
\bigvee \Big(\mathsf{aux}^{\dagger} \notin \mathcal{W}^{(j+1)} \wedge \mathsf{aux}^{\odot} \neq \bot\Big)\Big] \\
\leq \mu^{(j)}(\kappa) + \nu(\kappa) \\
\Pr_{r_{\mathcal{A}}, r_C}\Big[\Big(\exists(\mathsf{aux}, _, _, _) \in \mathsf{CCA.Exp}(1^{\kappa}, \mathcal{A}, j, \mathcal{L}^{(j+1)}) : \mathsf{aux} \notin \mathcal{W}^{(j+1)}\Big) \\
\bigvee \Big(\mathsf{aux}^{\dagger} \notin \mathcal{W}^{(j+1)} \wedge \mathsf{aux}^{\odot} \neq \bot\Big)\Big] \\
\leq \mu^{(j)}(\kappa) + \nu(\kappa) \\
\Pr_{r_{\mathcal{A}}, r_C}\Big[\exists(\mathsf{aux}, _, _, _) \in \mathsf{CCA.Exp}(1^{\kappa}, \mathcal{A}, j+1, \mathcal{L}^{(j+1)}) : \mathsf{aux} \notin \mathcal{W}^{(j+1)}\Big] \\
\leq \mu^{(j)}(\kappa) + \nu(\kappa)
\end{aligned}$$

□

Using induction, we have the following claim.

Corollary 6.1. Assuming the conditions in Lemma 6.3 hold,

$$\Pr\Big[\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^{\kappa}, \mathcal{A}, j, \mathcal{L} = \mathcal{L}^{(\mathsf{Q})}) : \mathsf{aux} \notin \mathcal{W}^{(\mathsf{Q})}\Big] = \mathsf{negl}(\kappa).$$

Proof. Combining the probabilities from the previous lemma statement, we get,

$$\Pr\Big[\exists(\mathsf{aux}, \_, \_, \_) \in \mathsf{CCA.Exp}(1^{\kappa}, \mathcal{A}, j, \mathcal{L} = \mathcal{L}^{(\mathsf{Q})}) : \mathsf{aux} \notin \mathcal{W}^{(\mathsf{Q})}\Big] \leq \mu^{(0)}(\kappa) + \mathsf{Q} \cdot \nu(\kappa).$$

Since $\mu^{(0)}(\kappa) = 0$, $\nu(\kappa)$ is negligible and Q is polynomial in security parameter, we have the claim. □

6.2.2 Analyzing sequence of Games

Lemma 6.4. Suppose $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is $(2^{\kappa^{\delta}}, \frac{|\mathsf{advice}|}{2n'})$ binding secure Definition 4.4 auxilary input equivocal commitment scheme. HPRG be an injectively extended hinting PRG and $\mathsf{Small.Val}$ is v-efficient. Then, for an non-uniform adversary $\mathcal{A}$ running in time $\mathsf{poly}(2^{\kappa^c})$ where $c \in (0,1)$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}^0_{\mathcal{A}} - \mathsf{adv}^1_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$ where equivocal commitment is run on security parameter $\kappa' = \kappa^{\frac{v}{\delta}}$.

Proof. Suppose for sake of contradiction we have an adversary $\mathcal{A}$ which has noticeable difference ϵ in advantage between Games 0 and 1. Since Games 0 and 1 only differ in the Val oracle, then with probability at least ϵ, $\mathcal{A}$ must query CCA.Val on some com for which $\mathsf{CCA.Val}(\cdot)$ and $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \cdot, \mathcal{L})$ differ. From Lemma 6.1, we know that for this query, $\mathsf{com.aux} \notin \mathcal{W}^{(\mathsf{Q})}$ and $\mathsf{CCA.Equiv}(\mathsf{com}) \neq \perp$. This means $\mathcal{A}$ on running Game 1 must query a com such that $\mathsf{com.aux} \notin \mathcal{W}^{(\mathsf{Q})}$ and $\mathsf{CCA.Equiv}(\mathsf{com}) \neq \perp$. Consider the experiment $\mathsf{CCA.Exp}(1^\kappa, \mathcal{A}, \mathcal{L} = \mathcal{L}^{(\mathsf{Q})})$, such a query thus must be stored in $\mathcal{E}$. From Corollary 6.1 for $j = \mathsf{Q}$, we know that this happens with $\mathsf{negl}(\kappa)$ probability. Thus ϵ must be negligible.

□

Lemma 6.5. Assuming that the equivocal commitment is statistically equivocal from Definition 4.5. For any adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}^1_{\mathcal{A}} - \mathsf{adv}^2_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$ where equivocal commitment is run on security parameter $\kappa' = \kappa^{\frac{v}{\delta}}$.

Proof. From Definition 4.5, we know that the statistical distance between $(\mathsf{aux}^*, \sigma_i^*, y_i^*)$ in Games 1 and 2 is negligible. Since the rest of the inputs to the games are the same, this bounds the statistical distance of the output by a negligible function $\mathsf{negl}(\kappa)$ as well. □

Lemma 6.6. Assuming that the base commitment scheme is 2^{κ^c}-subexponentially CCA secure, 2^{κ^v}-efficient CCA commitment from Definition 3.5 and Definition 3.2. For any non-uniform adversary $\mathcal{A}$ that runs in time $\mathsf{poly}(2^{\kappa^c})$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}^2_{\mathcal{A}} - \mathsf{adv}^3_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$.

Proof. Let $\mathcal{A}$ be an adversary given advice that has non-negligible advantage given by the polynomial $p(\cdot)$ in distingushing between the two games, i.e. for infinitely many $\kappa \in \mathbb{N}$,

$$|\mathsf{adv}^2_{\mathcal{A}} - \mathsf{adv}^3_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}.$$

Define Game 2_0 as Game 2. For all $j \in [N \cdot n]$, we define Game 2_j same as Game 2_{j-1}, with the following additional changes:
We can write $j = (i' - 1) \cdot N + (x' - 1)$ where $x' \in [N], i' \in [n]$ from Euclidean division, and we change the way $c^*_{x',i',\bar{s_i^*}}$ is generated from

$$c^*_{x',i',\bar{s^*_{i'}}} = \mathsf{Small.Com}(1^\kappa, (x', \mathsf{tag}_{x'}, \bar{s^*_{i'}}), y^*_{i',\bar{s^*_{i'}}}; \tilde{r}^*_{x',i'})$$

to

$$c^*_{x',i',\bar{s_{i'}^*}} = \mathsf{Small.Com}(1^\kappa, (x', \mathsf{tag}_{x'}, \bar{s^*_{i'}}), y^*_{i',\bar{s}^*_{i'}}; \tilde{r}^*_{x',i'})$$

Observe that Game $2_{N \cdot n}$ is exactly Game 3.
Thus $\exists j = j(\kappa) \in [N \cdot n]$, for infinitely many $\kappa \in \mathbb{N}$,

$$|\mathsf{adv}^{2_{j-1}}_{\mathcal{A}} - \mathsf{adv}^{2_j}_{\mathcal{A}}| \geq \frac{1}{p(\kappa)(N \cdot n)}.$$

We will show a reduction $\mathcal{B}_j$ that achieves a non-negligible advantage to the security of Small.Com. We present the non-uniform computation that the algorithm performs up front followed by the steps of the algorithm where it tries to break the hiding of the commitment scheme.

Reduction $\mathcal{B}_j(1^\kappa)$:

Non-Uniform Computation:

- Run $\mathcal{L} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, (\mathcal{A}, \mathsf{advice}))$ non-uniformly.
- Compute equivocations.
  - Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
  - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
  - Let $(\mathsf{HPRG.pp}^*, n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(w, N\cdot\ell)})$.
  - Sample $s^* = s_1^* \dots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
  - Set $\mathsf{aux}^* = (\mathsf{HPRG.pp}^*, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}^*, s^*))$.
  - For all $i \in [n]$ run $\mathsf{AuxEquiv.Equivocate}(1^{\kappa'}, \mathsf{aux}^*) \to (\sigma_i^*, y_{i,0}^*, y_{i,1}^*)$.
  - Let L be $(s^*, \mathsf{aux}^*, \{(\sigma_i^*, y_{i,0}^*, y_{i,1}^*)\}_{i\in n})$.

1. $\mathcal{A}$ sends a challenge $\mathsf{tag}^* \in \{0,1\}^N$ to $\mathcal{B}_j$.
2. Let $c^*_{x',i',\bar{s}^*_{i'}}$ be the commitment changed in Game 2_j as described above.
3. Send challenge tag $(x', \mathsf{tag}^*_{x'}, \bar{s}^*_{i'})$ to challenger.
4. **Pre Challenge Phase:**
   - For every com query,
   $$\mathsf{com} = \Big(\mathsf{tag}, \mathsf{aux}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]})\Big)$$
   to $\mathcal{B}_j$.
   - $\mathcal{B}_j$ answers by running $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$. This can be done efficiently using $\mathcal{B}_j$'s own Pre Challenge oracle access to $\mathsf{Small.Val}$, $\mathcal{L}$ generated non-uniformly and runs $\mathsf{CCA.FindAlt}$ manually. Since $\mathsf{CCA.FindAlt}$ is only run on an index x^* such that $\mathsf{tag}^*_{x^*} \neq \mathsf{tag}_{x^*}$, it will never call $\mathsf{Small.Val}$ on $(x', \mathsf{tag}^*_{x'}, \bar{s}^*_{i'})$.
5. **Challenge Phase:**
   - $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^w$.
   - Select a random bit β.
6. **Part 1:**
   - Recall $\kappa' = \kappa^{\frac{v}{\delta}}$, $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Now using L, we retreive the non-uniform equivocations, i.e. parse L as $(s^*, \mathsf{aux}^*, \{\sigma_i^*, y_{i,0}^*, y_{i,1}^*\}_{i\in n})$.
   - Parse aux^* as $(\mathsf{HPRG.pp}^*, \mathsf{aux}')$.
   - Let $r_{x,i}, \tilde{r}_{x,i} \in \{0,1\}^\ell$ be defined as follows:
   - For $i \in [n]$
     (a) Compute $(r_{1,i}, r_{2,i}, \dots, r_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
     (b) Sample $(\tilde{r}_{1,i}, \tilde{r}_{2,i}, \dots, \tilde{r}_{N,i}) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$

(a) Submit $m_0^* = y^*_{i',s^*_{i'}}, m_1^* = y^*_{i',\bar{s}^*_{i'}}$ to challenger

(b) Receive $\mathsf{com}^* = \mathsf{Small.Com}((x', \mathsf{tag}^*_{x'}, \bar{s^*_{i'}}), m_b^*; r)$ from challenger

(c) Set $c^*_{x',i',\bar{s^*_{i'}}} = \mathsf{com}^*$

(d) Run **Part 2** of **Challenge Phase** using the message m_β and with the exception that $c^*_{x',i',\bar{s^*_{i'}}}$ is computed as noted above and submit output to $\mathcal{A}$.

7. **Post Challenge Phase:** Proceeds exactly as Pre Challenge Phase.
8. Receive bit guess β' from $\mathcal{A}$.
9. If $\beta = \beta'$, output 0. Otherwise, output 1.

Claim 6.7. $\mathcal{B}_j$ is a non-uniform algorithm that gets polynomial size advice and runs in time $\mathsf{poly}(2^{\kappa^c})$.

Proof. Since the procedure CCA.AdviceList outputs a polynomial size list $\mathcal{L}$ and since n and output of AuxEquiv.Equivocate is polynomial in κ, L is polynomial in κ and $\mathcal{B}_j$ gets polynomial size advice. The runtime of $\mathcal{B}_j$ includes running the algorithm $\mathcal{A}$ that runs in time $\mathsf{poly}(2^{\kappa^c})$ and other algorithms that run in $\mathsf{poly}(\kappa)$. Note that it doesn't need to run Small.Val as it uses the oracle from the security of the small commitment scheme and it gets equivocations non-uniformly. □

Claim 6.8. The advantage of $\mathcal{B}_j$ in winning the message hiding game for the base commitment scheme Small.Com from Definition 3.5 is $\geq \frac{1}{2p(\kappa)(N\cdot n)}$ for infinitely many $\kappa \in \mathbb{N}$.

Proof. First note observe that if $\beta = 0$, then

$$c^*_{x',i',\bar{s^*_{i'}}} = \mathsf{Small.Com}((x', \mathsf{tag}^*_{x'}, \bar{s^*_{i'}}), y^*_{i',s^*_{i'}}; r)$$

which is exactly what it is in Game 2_{j-1}, and similarly, if $\beta = 1$

$$c^*_{x',i',\bar{s^*_{i'}}} = \mathsf{Small.Com}((x', \mathsf{tag}^*_{x'}, \bar{s^*_{i'}}), y^*_{i',\bar{s}^*_{i'}}; r)$$

which is what it is in Game 2_j.

Let q be the probability $\mathcal{A}$ wins Game 2_j and $\mathcal{A}$ wins Game 2_{j-1} with probability $q \pm \frac{1}{p(\kappa)\cdot N\cdot n}$. $\mathcal{B}_j$ wins if $\beta = \beta'$ and $b = 0$ - i.e. $\mathcal{A}$ wins Game 2_{j-1} or if $\beta \neq \beta'$ and $b = 1$ - i.e. $\mathcal{A}$ loses Game 2_j. Thus for infinitely many $\kappa \in \mathbb{N}$, the probability of $\mathcal{B}_j$ winning is given by,

$$\frac{1}{2}\left(q \pm \frac{1}{p(\kappa)\cdot N\cdot n}\right) + \frac{1}{2}(1-q) = \frac{1}{2} \pm \frac{1}{2\cdot p(\kappa)\cdot N\cdot n}.$$

□

As Small.Com is a secure scheme, the proof of the lemma follows immediately by contradiction from the above claims.

□

Lemma 6.7. Assuming that the hinting PRG is subexponentially secure with $T = 2^{\kappa^\gamma}$ where $\gamma \in (0,1)$ from Definition 5.1. For any non-uniform 2^{κ^c}-subexponentially secure adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$ where hinting PRG is run on security parameter $\kappa'' = \kappa'^{\frac{1}{\gamma}} = \kappa^{\frac{v}{\delta\cdot\gamma}}$.

Proof. Let $\mathcal{A}$ be a non-uniform adversary given advice that has non-negligible advantage given by the polynomial $p(\cdot)$ in distingushing between the two games, i.e. for infinitely many $\kappa \in \mathbb{N}$,

$$|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}.$$

We will construct a $\mathsf{poly}(2^{\kappa^c})$ time non-uniform adversary $\mathcal{B}$ which has advantage $\frac{1}{2p(\kappa)}$ in the hinting PRG Game as per Definition 5.1 where inputs were called on security parameter κ''.

Reduction $\mathcal{B}\left(\mathsf{HPRG.pp}, \left(r_0^\beta, r_{\mathsf{ext}}^\beta, \left\{r_{i,b}^\beta\right\}_{i\in[n],b\in\{0,1\}}\right)\right)$:

Non-Uniform Computation:

- Run $\mathcal{L} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, (\mathcal{A}, \mathsf{advice}))$ non-uniformly.

1. Choose a random bit $a \in \{0,1\}$.
2. Run $\mathcal{A}$
   (a) **Pre Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$.
   (b) $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^w$.
   (c) **Challenge Phase:**
   - Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
   - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Let $r_{x,i,b} \in \{0,1\}^\ell$ be defined as follows:
   - For $i \in [n]$, $b \in \{0,1\}$
     i. Split up $(r_{1,i,b}, r_{2,i,b}, \ldots, r_{N,i,b}) = \underline{r_{i,b}^\beta}$
   - For all $i \in [n]$ run $\mathsf{AuxEquiv.Equivocate}(1^{\kappa'}, r_{\mathsf{ext}}^\beta) \to (\sigma_i^*, y_{i,0}^*, y_{i,1}^*)$.
   (d) **Part 2:**
   - Set $c^* = \underline{r_0^\beta} \oplus m_a^*$.
   - For $i \in [n]$, $x \in [N]$, $b \in \{0,1\}$
     i. $c_{x,i,b}^* = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, b), y_{i,b}^*; \underline{r_{x,i,b}})$
   - Finally, it sends $\mathsf{com}^* = \left(\mathsf{tag}^*, r_{\mathsf{ext}}^\beta, \mathsf{HPRG.pp}^*, c^*, (\sigma_i^*, \left(c_{x,i,0}^*, c_{x,i,1}^*\right)_{x\in[N]})_{i\in[n]}\right)$ as the commitment. All of the randomness is used as the decommitment string.
   (e) **Post Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$.
   (f) Receive a' from $\mathcal{A}$.
3. If $a' = a$, then output $\beta' = 1$. Otherwise output $\beta' = 0$.

Claim 6.9. $\mathcal{B}$ is a non-uniform algorithm that outputs polynomial size advice and runs in time $\mathsf{poly}(2^{\kappa''^\gamma}) = \mathsf{poly}(2^{\kappa'}) = \mathsf{poly}(2^{\kappa^{\frac{v}{\delta}}})$.

Proof. $\mathcal{B}$ runs CCA.ValAlt that runs in time $\mathsf{poly}(|m|, 2^{\kappa^v})$ from Claim 6.1. Additionally, it runs AuxEquiv.Equivocate n times that runs in time $\mathsf{poly}(2^{\kappa'})$. Since message lengths are polynomial in κ and N, ℓ are polynomial in κ implying n is $\mathsf{poly}(\kappa)$. The whole algorithm runs in time $\mathsf{poly}(2^{\kappa''^{\gamma}})$. □

Claim 6.10. If $\mathcal{A}$ has advantage $|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}$, $\mathcal{B}$ has advantage in the HPRG game in Definition 5.1 $\geq \frac{1}{2p(\kappa)}$

Proof. We observe that when $\beta = 1$ in the HPRG Game - when $\mathcal{B}$ receives

$$\Bigg(r^1_0 = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0), r^1_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s),$$
$$\left\{r^1_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i), r^1_{i,\bar{s}_i} \xleftarrow{R} \{0,1\}^{\ell}\right\}_{i\in[n]}\Bigg)$$

$\mathcal{A}$ is run on exactly Game 3, and when $\beta = 0$ - i.e. when $\mathcal{B}$ receives

$$\Bigg(r^0_0 \xleftarrow{R} \{0,1\}^{\ell}, r^0_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s),$$
$$\left\{r^0_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i), r^0_{i,\bar{s}_i} \xleftarrow{R} \{0,1\}^{\ell}\right\}_{i\in[n]}\Bigg)$$

$\mathcal{A}$ is run is identical to Game 4 (barring the fact that we are replacing c^* with $m^*_\beta \oplus r^0_0$ rather than just $c^* \xleftarrow{R} \{0,1\}^{\ell}$, but these are identically distributed). So suppose $\mathcal{A}$ has probability p of winning Game 4. Then we can see that $\mathcal{B}$ wins the HPRG ($\beta' = \beta$) game either when $\mathcal{A}$ is run on Game 3 and wins, or when $\mathcal{A}$ is run on Game 4 and loses. These events happen with probabilities

$$\frac{1}{2}\left(p \pm \frac{1}{p(\kappa)}\right) + \frac{1}{2}(1-p) = \frac{1}{2} \pm \frac{1}{2 \cdot p(\kappa)}$$

for infinitely many $\kappa \in \mathbb{N}$. □

Since $\mathcal{B}$'s advantage must be negligible by Definition 5.1, a contradiction, which concludes our proof. □

Lemma 6.8. For any adversary $\mathcal{A}$, $\mathsf{adv}^4_{\mathcal{A}} = 0$.

Proof. The challenge commitment is independent of the message. Thus the probability of any adversary guessing an independent random bit is $\frac{1}{2}$. □

From the above lemmas we can conclude that $\mathsf{adv}^0_{\mathcal{A}} = \mathsf{negl}(\kappa)$. This completes the proof of the theorem. □

7 Compilation of Transformations

We show how to combine our transformations Amplify and OneToMany to prove that if we start with a base scheme that is secure against non-uniform "same tag" adversaries (see Definition 3.6) for $32 \cdot \mathsf{ilog}(q, \kappa)$ tags where the notation $\mathsf{ilog}(q, \kappa)$ denotes $\underbrace{\lg \lg \cdots \lg}_{q \text{ times}}(\kappa)$ [9] and q is some constant,

[9] The notation $\mathsf{ilog}(0, \kappa)$ is defined as κ.

then using our described transformations, we can construct a scheme that is secure against non-uniform adversaries (see Definition 3.5) for $16 \cdot 2^\kappa$ tags.

Our sequence of transformations is very similar to [GKLW21], where we start with a base scheme BaseCCA that satisfies property Definition 3.7. We then remove the same tag restriction on the adversary by using the transformation OneToMany in Section E and then amplify the tag space by using the transformation Amplify in Section 6 $q+1$ times. The two main deviations from the formal treatment of [GKLW21] is due to our proof technique, i.e. we need to keep track of the message and efficiency of the val oracle when we perform the sequence of transformations.

We remind the reader that the order of the sequence of transformations is important as to perform Amplify and OneToMany we need the commitment scheme to be recoverable from randomness. Additionally, OneToMany does computation that is polynomial in the number of tags for the input scheme. Thus, we must remove the "same tag" restriction from our adversary before amplifying our tags with Amplify. Based on the sequence of transformations we have discussed, our tag space will amplify as follows. At the end of OneToMany, we will end up with $16 \cdot \mathsf{ilog}(q, \kappa)$ sized tag space. And after $q+1$ applications of Amplify, we will end up with $16 \cdot 2^\kappa$ sized tag space. One application of Amplify converts a $4N$ tag space scheme to a 2^N tag space scheme. Thus on input a $4 \cdot 4 \cdot \mathsf{ilog}(q, \kappa)$ tag space, one gets a $2^{4 \cdot \mathsf{ilog}(q,\kappa)} = 16 \cdot \mathsf{ilog}(q-1, \kappa)$ tag space.

Additionally, when using the schemes in a sequence of transformations we need to keep track of the message spaces we chose in our output scheme. For instance, to perform the transformation Amplify and OneToMany, the constructions output committment σ to each seed bit of the hinting PRG. The base scheme here takes in the decommitment string of σ as input. Thus the length of the base scheme being transformed should be able to support messages of this length for the transformation to be correct. Let the length of the decommitment string be denoted by a polynomial function $\mathsf{DecomLen}(\cdot)$ that takes as input the security parameter κ [10]. Thus for the transformations Amplify and OneToMany, u (input message length of the base scheme) should be equal to $\mathsf{DecomLen}(\kappa')$ where κ' is the security parameter input to the equivocal commitment. In our transformations κ' is set as $\kappa^{\frac{v}{\delta}}$ where there exists a constant δ such that the setupless equivocal commitment scheme is 2^{κ^δ}-hiding secure and the base scheme is 2^{κ^v}-efficient[11].

Our formal transformation is below. We start with a base commitment scheme BaseCCA and output the scheme $(\mathsf{AmplifiedCCA}^{q+1}.\mathsf{Com}, \mathsf{AmplifiedCCA}^{q+1}.\mathsf{Val})$. We list a few assumptions on our transformation -

- Let there exist variables $\delta, \gamma, \tilde{v}$ such that $\delta \in (0,1)$ and the setupless equivocal commitment scheme is 2^{κ^δ}-hiding secure, $\gamma \in (0,1)$ and the hinting PRG with injective extension is 2^{κ^γ}-secure and the dependence of seed on the security parameter be such that seed length $n = \Theta(\kappa^{\tilde{v}})$.

- We start with a base scheme that is 2^κ-efficient and secure against non-uniform "same tag" 2^{κ^c}-subexponentially secure adversaries for tag space $32\mathsf{ilog}(q, \kappa)$ tags for any constant q.

  If the base scheme runs in time some constant $\mathsf{poly}(2^{\kappa^a})$ where $a \in (0,1)$ then the scheme is 2^κ-efficient. Otherwise, on input security parameter κ, we can run the scheme with parame-

[10] The length of the decommitment string can depend on aux, but since aux is also called with a polynomial function in κ based on the hinting PRG construction, we simplify the notation. In our specific construction for AuxEquiv in Section 4, the decommitment string length doesn't depend on aux.

[11] Recall from Definition 3.2 that a 2^{κ^v}-efficient scheme with $v \geq 1$ implies that the runtime of Small.Val is polynomial in 2^{κ^v}.

ters $\kappa^{\frac{1}{a}}$ to get a 2^{κ}-efficent scheme that is still 2^{κ^c} sub-exponentially secure with $c \in (0,1)$ for some constant c. Thus we can wlog claim that we start with a 2^{κ}-efficient scheme. This will help simplify notation.

- Let the base scheme support messages of length $u = \mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{1}{\delta}})$ and the final scheme support messages of length w.

Recall that the transformations OneToMany (Section E) and Amplify (Section 6) take in the following parameters - a scheme to be transformed, hinting PRG with injective extension HPRG, setupless equivocal commitment scheme AuxEquiv, the length of the messages supported by the output scheme and an efficiency parameter v such that the output scheme is 2^{κ^v}-efficient.

$\mathsf{CompiledAmplify}(\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val}, u), \mathsf{HPRG}, \mathsf{AuxEquiv}, w)$

1. $\mathsf{AmplifiedCCA}^0 \leftarrow \mathsf{OneToMany}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{AuxEquiv}, \mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{v_0}{\delta}}), v_0)$ where $v_0 = \frac{\tilde{v}}{\delta \cdot \gamma}$.
2. For $i \in [q]$,
   (a) $\mathsf{AmplifiedCCA}^i \leftarrow \mathsf{Amplify}(\mathsf{AmplifiedCCA}^{i-1}, \mathsf{HPRG}, \mathsf{AuxEquiv}, \mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{v_i}{\delta}}), v_i)$ where $v_i = \left(\frac{\tilde{v}}{\delta \cdot \gamma}\right)^{i+1}$.
3. $\mathsf{AmplifiedCCA}^{q+1} \leftarrow \mathsf{Amplify}(\mathsf{AmplifiedCCA}^q, \mathsf{HPRG}, \mathsf{AuxEquiv}, w, v_{q+1})$ where $v_{q+1} = \left(\frac{\tilde{v}}{\delta \cdot \gamma}\right)^{q+2}$.
4. Output $(\mathsf{AmplifiedCCA}^{q+1}.\mathsf{Com}, \mathsf{AmplifiedCCA}^{q+1}.\mathsf{Val})$

Below we analyze CompiledAmplify by stating theorems on correctness, efficiency and security.

Theorem 7.1. For every $\kappa \in \mathbb{N}$, any constant q, any polynomial w, let $\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val}, u)$ be a perfectly correct CCA commitment scheme for message space $\{0,1\}^u$ by Definition 3.1 with tag space $32 \cdot \mathsf{ilog}(q, \kappa)$. Let $\mathsf{AuxEquiv} = (\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ be a perfectly correct equivocal commitment scheme by Definition 4.2. Let there exist a constant δ such that $u = \mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{1}{\delta}})$.

Then, we have that the scheme $\mathsf{CompiledAmplify}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{AuxEquiv}, w)$ is a perfectly correct CCA commitment scheme for $16 \cdot 2^{\kappa}$ tags.

Proof. By the assumption that BaseCCA is a correct CCA commitment scheme, for $32 \cdot \mathsf{ilog}(q, \kappa)$ tags and correctness of AuxEquiv, we can apply Claim E.2 to conclude $\mathsf{AmplifiedCCA}^0$ is a correct CCA commitment scheme for $16 \cdot \mathsf{ilog}(q, \kappa)$ tags. Note that RandomBaseCCA is secure for messages in space $\{0,1\}^u$ where $u = \mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{1}{\delta}})$ and OneToMany calls Same.Com on a decommitment string called on security parameter $\kappa^{\frac{1}{\delta}}$ (Recall that the setupless equivocal commitment scheme is $2^{\kappa^{\delta}}$-hiding secure). Thus the correctness holds.

Similarly, using again the correctness of AuxEquiv with correctness of Claim 6.2, we can conclude inductively that $\mathsf{AmplifiedCCA}^i$ is a perfectly correct CCA commitment scheme for $16 \cdot \mathsf{ilog}(q - i, \kappa)$ tags, where $i \in [q]$. Observe that during the transformation AuxEquiv is called on security parameter $\kappa' = \kappa^{\frac{v_{i-1}}{\delta}}$, which was the length set as the output of $\mathsf{AmplifiedCCA}^{i-1}$.

And finaly the final scheme $\mathsf{AmplifiedCCA}^{q+1}$ is a perfectly correct CCA commitment scheme for $16 \cdot 2^{\kappa}$ tags and any message space w via correctness of $\mathsf{AmplifiedCCA}^q$ and AuxEquiv and the message length of the base scheme being set to $\mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{v_q}{\delta}})$. □

Theorem 7.2. For every $\kappa \in \mathbb{N}$, any constant q, any polynomial w, let $\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val}, u)$ be an 2^{κ}-efficient CCA commitment scheme by Definition 3.2 with tag space $32 \cdot \mathsf{ilog}(q, \kappa)$. Let $\mathsf{AuxEquiv} = (\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ be an efficient equivocal commitment scheme by Definition 4.3. Let there exist constants $\delta, \gamma, \tilde{v}$ such that setupless equivocal commitment scheme is $2^{\kappa^{\delta}}$-hiding secure and $u = \mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{1}{\delta}})$; $\gamma \in (0,1)$ and the hinting PRG with injective extension is $2^{\kappa^{\gamma}}$-secure; the dependence of seed on the security parameter be such that $n = \Theta(\kappa^{\tilde{v}})$.

Then, $\mathsf{CompiledAmplify}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{AuxEquiv}, w)$ is an $2^{\kappa^{v}_{q+1}}$-efficient CCA commitment scheme for $16 \cdot 2^{\kappa}$ tags where $v_{q+1} = \left(\frac{\tilde{v}}{\delta \cdot \gamma}\right)^{q+2}$.

Proof. By the assumption that $\mathsf{BaseCCA}$ is a 2^{κ}-efficient CCA commitment scheme, and using Claim E.1, we can conclude that $\mathsf{AmplifiedCCA}^0$ is a $2^{\kappa^{v_0}}$-efficient scheme where $v_0 = \frac{\tilde{v}}{\delta \cdot \gamma}$. The claim needs the condition that the number of tags are polynomial in κ.

Now we inductively apply Claim 6.1 $i \in [q+1]$ to state that $\mathsf{AmplifiedCCA}^i$ is a $2^{\kappa^{v}_{i}}$-efficient scheme where $v_i = \left(\frac{\tilde{v}}{\delta \cdot \gamma}\right)^{i+1}$. From the hypothesis, $\mathsf{AmplifiedCCA}^{i-1}$ is $2^{\kappa^{v}_{i-1}}$-efficient. Using Claim 6.1 with the fact that the equivocal commitment scheme is efficient by Definition 4.3, we have that $\mathsf{AmplifiedCCA}^i$ is $2^{\kappa^{v}_{i}}$-efficient. Note that again the number of tags stay polynomial in κ for $i \in [q]$. Also observe that it is crucial that we only apply the inductive step a constant number of times, as each transformation expands the output of the committed string by a polynomial factor. □

Theorem 7.3. For every $\kappa \in \mathbb{N}$, any constant q, any polynomial w, let $\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val}, u)$ be a CCA commitment scheme that is hiding against non-uniform "same tag" $2^{\kappa^{c}}$-subexponential adversaries according to Definition 3.6 for tag space $32 \cdot \mathsf{ilog}(q, \kappa)$. $\mathsf{HPRG} = (\mathsf{HPRG.Setup}, \mathsf{HPRG.Eval})$ be a hinting PRG scheme with injective extension that is $T = 2^{\kappa^{\gamma}}$ secure by Definition 5.1 for $\gamma \in (0,1)$. $\mathsf{AuxEquiv} = (\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ be an equivocal commitment without setup scheme that is $T = 2^{\kappa^{\delta}}$ binding secure Definition 4.4 and statistically hiding for some constant $\delta \in (0,1)$. Let u be equal to $\mathsf{AuxEquiv.DecomLen}(\kappa^{\frac{1}{\delta}})$.

Then, $\mathsf{CompiledAmplify}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{AuxEquiv}, w)$ is a CCA commitment scheme that is hiding against non-uniform $2^{\kappa^{c}}$-subexponential adversaries according to Definition 3.5 for tag space $16 \cdot 2^{\kappa}$.

Proof.

Claim 7.1. $\mathsf{AmplifiedCCA}^0$ is a CCA commitment scheme secure against non-uniform $2^{\kappa^{c}}$-subexponential adversaries with the recover from randomness property on tag space $16 \cdot \mathsf{ilog}(q, \kappa)$.

Proof. By assumption in Theorem 7.3, HPRG is a $2^{\kappa^{\gamma}}$ secure hinting PRG scheme with injective extension, $\mathsf{AuxEquiv}$ is a $2^{\kappa^{\delta}}$ secure equivocal commitment without setup scheme. $\mathsf{BaseCCA}$ is a CCA commitment scheme that is hiding against non-uniform $2^{\kappa^{c}}$-subexponential "same tag" adversaries with the recover from randomness property on tag space $32 \cdot \mathsf{ilog}(q, \kappa)$. Apply Theorem E.1 with $N = 16 \cdot \mathsf{ilog}(q, \kappa)$ to get the result. □

Claim 7.2. For all $i \in [q]$, $\mathsf{AmplifiedCCA}^i$ is a CCA commitment scheme secure against non-uniform $2^{\kappa^{c}}$-subexponential adversaries with the recover from randomness property on tag space $16 \cdot \mathsf{ilog}(q - i, \kappa)$. $\mathsf{AmplifiedCCA}^{q+1}$ is a CCA commitment scheme secure against non-uniform $2^{\kappa^{c}}$-subexponential adversaries with the recover from randomness property on tag space $16 \cdot 2^{\kappa}$.

Proof. We will proceed with induction on $i \in [q+1]$. The base case is true by Claim 7.1. By assumption in Theorem 7.3, HPRG is a 2^{κ^γ} secure hinting PRG scheme with injective extension and $\mathsf{AuxEquiv}$ is a 2^{κ^δ} secure equivocal commitment without setup scheme. By our induction hypothesis, $\mathsf{AmplifiedCCA}^{i-1}$ a CCA commitment scheme secure against non-uniform 2^{κ^c}-subexponential adversaries with the recover from randomness property on tag space $16 \cdot \mathsf{ilog}(q-i+1,\kappa)$. We apply Theorem 6.1 with $N = 4 \cdot \mathsf{ilog}(q+1-i,\kappa)$. Since $i \leq q+1$, we know $N \leq \kappa \in \mathsf{poly}(\kappa)$, so the theorem applies, giving us that $\mathsf{AmplifiedCCA}^{i}$ is a CCA commitment scheme secure against non-uniform 2^{κ^c}-subexponential adversaries with the recover from randomness property on tag space $2^{4\cdot\mathsf{ilog}(q-i+1,\kappa)} = 16 \cdot \mathsf{ilog}(q-i,\kappa)$. Thus at the end of the induction, we end up with $\mathsf{AmplifiedCCA}^{q}$, a CCA commitment scheme secure against non-uniform 2^{κ^c}-subexponential adversaries with the recover from randomness property on tag space $16 \cdot \kappa$.

We finally apply Theorem 6.1 with $N = 4 \cdot \kappa$ on $\mathsf{AmplifiedCCA}^{q}$ to get $\mathsf{AmplifiedCCA}^{q+1}$ as a CCA commitment scheme secure against non-uniform 2^{κ^c}-subexponential adversaries with the recover from randomness property on tag space $16 \cdot 2^{\kappa}$. □

By applying Claim 7.2, we conclude $\mathsf{AmplifiedCCA}^{q+1}$ is a non-uniform 2^{κ^c}-subexponentially secure CCA commitment with tag space $16 \cdot 2^{\kappa}$.

□

We import the following theorems about instantiating base schemes, from prior work.

Theorem 7.4. [KK19] For every constant $c > 0$, there exist correct, polynomially efficient, binding (3.3), same-tag CCA secure commitments with randomness recovery satisfying Definition 3.6 against non-uniform adversaries, with tag space $(c \lg \lg \lg \kappa)$, message space $u = \mathsf{poly}(\kappa)$ that make black-box use of subexponential quantum hard non-interactive commitments and subexponential classically hard non-interactive commitments in BQP, both with randomness recovery.

Theorem 7.5. [LPS17] For every constant $c > 0$, there exist correct, polynomially efficient, weak binding (3.4), same-tag CCA secure commitments with randomness recovery satisfying same-tag CCA security according to Definition 3.6 against non-uniform adversaries, with tag space $(c \lg \lg \lg \kappa)$, that make black-box use of subexponential time-lock puzzles [LPS17].

We remark that while [LPS17, KK19] prove that their constructions satisfy non-malleability with respect to commitment, their proof techniques also extend to exhibit same-tag CCA security against non-uniform adversaries. In a nutshell, both these works rely on two simultaneous axes of hardness to build their base schemes. As a consequence of this in the same-tag setting, for any pair of tags $(\mathsf{tag}, \widetilde{\mathsf{tag}})$ corresponding to the challenge query and CCA oracle queries of the adversary respectively, there is an oracle that inverts all commitments generated under $\tilde{\mathsf{tag}}$ but where commitments under tag remain secure in the presence of this oracle. In both these works [LPS17, KK19], we note that while the specific oracle is only used to invert parallel queries of the adversary (thereby obtaining many-many non-malleability), the oracle is actually capable of inverting (unbounded) polynomially many *adaptive* queries, thereby also achieving same-tag CCA security. In [LPS17], this oracle over-extracts, therefore achieving the weaker property of same-tag CCA security with weak binding. The [KK19] scheme does not suffer from over-extraction and achieves the stronger notion of (standard) binding. The [KK19] scheme can be observed to satisfy randomness recovery by relying on the recovery algorithm of the underlying commitments. The [LPS17] scheme outputs a commitment to a bit b as

$$f(s;r), r', \langle s, r' \rangle \oplus b$$

which satisfies randomness recovery given all the randomness used to commit.

Combining this theorem with Theorem 7.3, we obtain the following corollaries.

Corollary 7.1. There exists a perfectly correct, polynomially efficient, binding (Definition 3.3) and CCA secure commitment satisfying Definition 3.5 against non-uniform adversaries, with tag space 2^κ for security parameter κ, that makes black-box use of subexponential quantum hard one-way functions, subexponential classically hard one-way functions in BQP, subexponential hinting PRGs and subexponential keyless collision-resistant hash functions.

Corollary 7.2. There exists a perfectly correct, polynomially efficient, binding (Definition 3.3) and CCA secure commitment satisfying Definition 3.5 against non-uniform adversaries, with tag space 2^κ for security parameter κ, that makes black-box use of subexponential time-lock puzzles as used in [LPS17], subexponential hinting PRGs and subexponential keyless collision-resistant hash functions.

Finally, we point out that while all our formal theorems discuss CCA security, our transformations also apply as is to the case of amplifying parallel CCA security (equivalently, concurrent non-malleability w.r.t. commitment). That is, given a base scheme that is only same-tag parallel CCA secure (or non-malleable w.r.t. commitment) for small tags, our transformations yield a scheme for all tags that is parallel CCA secure (or concurrent non-malleable w.r.t. commitment) for tags in 2^κ, without the same tag restriction.

8 Acknowledgments

We thank Daniel Wichs for a useful discussion about the construction of our new Hinting PRGs, anonymous reviewers for helpful feedback on a preliminary version of this work, and Nir Bitansky and Rachel Lin for answering our questions about keyless collision-resistant hash functions.

References

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 468–499, 2017.

[Bar02] Boaz Barak. Constant-Round Coin-Tossing with a Man in the Middle or Realizing the Shared Random String Model. In *FOCS 2002*, pages 345–355, 2002.

[BFMR18] Brandon Broadnax, Valerie Fetzer, Jörn Müller-Quade, and Andy Rupp. Non-malleability vs. cca-security: The case of commitments. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, volume 10770 of *Lecture Notes in Computer Science*, pages 312–337. Springer, 2018.

[BGJ+17] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Dakshita Khurana, and Amit Sahai. Round optimal concurrent MPC via strong simulation. In *Theory of Cryptography*

- 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I, pages 743–775, 2017.

[BGJ+18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 459–487, 2018.

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, pages 645–677, 2017.

[BKP18] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 671–684. ACM, 2018.

[BL18a] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 500–532, 2018.

[BL18b] Nir Bitansky and Huijia Lin. One-message zero knowledge and non-malleable commitments. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 209–234. Springer, 2018.

[BOV07] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. *SIAM J. Comput.*, 37(2):380–400, 2007.

[BP15] Nir Bitansky and Omer Paneth. Zaps and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 401–427. Springer, 2015.

[CCG+21] Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Oblivious transfer from trapdoor permutations in minimal rounds. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 518–549. Springer, 2021.

[CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual*

ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998, pages 141–150. ACM, 1998.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive Hardness and Composable Security in the Plain Model from Standard Assumptions. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '10, pages 541–550, 2010.

[COSV16] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Concurrent non-malleable commitments (and more) in 3 rounds. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 270–299. Springer, 2016.

[COSV17] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Four-round concurrent non-malleable commitments from one-way functions. In *Annual International Cryptology Conference*, pages 127–157. Springer, 2017.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-Malleable Cryptography (Extended Abstract). In *STOC 1991*, 1991.

[DPP93] Ivan B Damgård, Torben P Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *Annual International Cryptology Conference*, pages 250–265. Springer, 1993.

[GKLW21] Rachit Garg, Dakshita Khurana, George Lu, and Brent Waters. Black-box non-interactive non-malleable commitments. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 159–185. Springer, 2021.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 25âĂŞ32, New York, NY, USA, 1989. Association for Computing Machinery.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, 2012.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.

[Goy11] Vipul Goyal. Constant Round Non-malleable Protocols Using One-way Functions. In *STOC 2011*, pages 695–704. ACM, 2011.

[GPR16] Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In *STOC*, pages 1128–1141, New York, NY, USA, 2016. ACM.

[GR19] Vipul Goyal and Silas Richelson. Non-malleable commitments using goldreich-levin list decoding. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 686–699. IEEE Computer Society, 2019.

[GRRV14] Vipul Goyal, Silas Richelson, Alon Rosen, and Margarita Vald. An algebraic approach to non-malleability. In *FOCS 2014*, pages 41–50, 2014.

[GVW20] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. New constructions of hinting prgs, owfs with encryption, and more. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 527–558. Springer, 2020.

[HHPV18] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 488–520, 2018.

[HM96] Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 201–215. Springer-Verlag, 1996.

[Khu17] Dakshita Khurana. Round optimal concurrent non-malleability from polynomial hardness. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 139–171. Springer, 2017.

[Khu21] Dakshita Khurana. Non-interactive distributional indistinguishability and non-malleable commitments. In *Advances in Cryptology - EUROCRYPT 2021*, 2021.

[KK19] Yael Tauman Kalai and Dakshita Khurana. Non-interactive non-malleability from quantum supremacy. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 552–582, 2019.

[KS17] Dakshita Khurana and Amit Sahai. How to achieve non-malleability in one or two rounds. In Umans [Uma17], pages 564–575.

[KW19] Venkata Koppula and Brent Waters. Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 671–700. Springer, 2019.

[LP] Huijia Lin and Rafael Pass. Constant-round Non-malleable Commitments from Any One-way Function. In *STOC 2011*, pages 705–714.

[LP09] Huijia Lin and Rafael Pass. Non-malleability Amplification. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pages 189–198, 2009.

[LPS17] Huijia Lin, Rafael Pass, and Pratik Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In Umans [Uma17], pages 576–587.

[LPV] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Concurrent Non-malleable Commitments from Any One-Way Function. In *TCC 2008*, pages 571–588.

[Pas13] Rafael Pass. Unprovable security of perfect NIZK and non-interactive non-malleable commitments. In *TCC*, pages 334–354, 2013.

[PPV08] Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. Adaptive One-Way Functions and Applications. In *Advances in Cryptology — CRYPTO '08*, pages 57–74, 2008.

[PR05] Rafael Pass and Alon Rosen. Concurrent Non-Malleable Commitments. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of ComputerScience*, FOCS '05, pages 563–572, 2005.

[PR08] Rafael Pass and Alon Rosen. New and Improved Constructions of Nonmalleable Cryptographic Protocols. *SIAM J. Comput.*, 38(2):702–752, 2008.

[PW10] Rafael Pass and Hoeteck Wee. Constant-round non-malleable commitments from sub-exponential one-way functions. In *EUROCRYPT 2010*, pages 638–655, 2010.

[Uma17] Chris Umans, editor. *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*. IEEE Computer Society, 2017.

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 205–223. Springer, 2007.

[Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *FOCS 2010*, pages 531–540, 2010.

A Preliminaries

A.1 (k, ϵ) Strong Extractors

Definition A.1. A distribution χ has min-entropy k if $\max_x \Pr[x = x' \xleftarrow{R} \chi] = 2^{-k}$

Definition A.2. A function $\mathsf{SExt} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a strong (k, ϵ) extractor if for all distributions χ with domain $\{0,1\}^n$ and min-entropy k, the following distribution

$$x \xleftarrow{R} \chi,\ s \xleftarrow{R} \{0,1\}^d,\ (s||\mathsf{SExt}(x, s))$$

has statistical distance at most ϵ from the uniform distribution on $d + m$ bits.

A.2 Diffie Hellman Assumptions

Let $\{\mathcal{G}_\kappa\}_\kappa$ be a collection of groups. We say the Decisional Diffie Hellman assumption holds for $\{\mathcal{G}_\kappa\}_\kappa$ if for any PPT adversary $\mathcal{A}$,

$$\left|\Pr\begin{bmatrix} g \xleftarrow{R} \mathcal{G}_\kappa \\ a,b \xleftarrow{R} [|\mathcal{G}_\kappa|] \\ \mathcal{A}(g,g^a,g^b,g^{ab})=1 \end{bmatrix} - \Pr\begin{bmatrix} g,h \xleftarrow{R} \mathcal{G}_\kappa \\ a,b \xleftarrow{R} [|\mathcal{G}_\kappa|] \\ \mathcal{A}(g,g^a,g^b,h)=1 \end{bmatrix}\right| < \mathsf{negl}(\kappa)$$

where g is a generator of $\mathcal{G}_\kappa$.

Similarly, we say the Computational Diffie Hellman assumption holds for $\{\mathcal{G}_\kappa\}_\kappa$ if for any PPT adversary $\mathcal{A}$,

$$\Pr\begin{bmatrix} g \xleftarrow{R} \mathcal{G}_\kappa \\ a,b \xleftarrow{R} [|\mathcal{G}_\kappa|] \\ g^{ab}=\mathcal{A}(g,g^a,g^b) \end{bmatrix} < \mathsf{negl}(\kappa)$$

A.3 Learning With Errors Assumptions

Let n and q be > 0 and χ be a noise distribution over $\mathbb{Z}_q$. We say $\mathsf{LWE}_{n,q,\chi}$ holds if for any PPT adversary $\mathcal{A}$ and polynomial N,

$$\left|\Pr\begin{bmatrix} a_i \xleftarrow{R} \mathbb{Z}_q^n \; \forall i \in [N] \\ a,s \xleftarrow{R} \mathbb{Z}_q^n \\ e,e_i \xleftarrow{R} \chi \; \forall i \in [N] \\ \mathcal{A}(\{a_i, a_i^{\mathrm{T}}\cdot s + e_i\}_i, (a, a^{\mathrm{T}}\cdot s + e)) = 1 \end{bmatrix} - \Pr\begin{bmatrix} a_i \xleftarrow{R} \mathbb{Z}_q^n \; \forall i \in [N] \\ a,s \xleftarrow{R} \mathbb{Z}_q^n, r \xleftarrow{R} \mathbb{Z}_q \\ e_i \xleftarrow{R} \chi \; \forall i \in [N] \\ \mathcal{A}(\{a_i, a_i^{\mathrm{T}}\cdot s + e_i\}_i, (a, r)) = 1 \end{bmatrix}\right| < \mathsf{negl}(\kappa)$$

B Distinct Strong Collision Resistance in the Auxiliary-Input Random Oracle Model

We show that distinct strong-collision-resistant hashing exists in the model of random oracles with auxiliary inputs of Unruh [Unr07].

In this model, the adversary $\mathcal{A}$ consists of two parts $(\mathcal{A}_1, \mathcal{A}_2)$. First $\mathcal{A}_1(\mathcal{R})$, who is completely unbounded, obtains a full description of a (shrinking) random oracle R and outputs some (short) auxiliary information z about the random oracle. Then at the second stage $\mathcal{A}_2^{\mathcal{R}}(z)$ obtains this auxiliary input, as well as oracle access to $\mathcal{R}$, and attempts to output K collisions in $\mathcal{R}$ with all distinct entries.

We show that $\mathcal{A}$ cannot output pairwise collisions with everywhere-distinct inputs that are significantly larger than the size of the auxiliary input. First, we prove the following theorem.

Theorem B.1. For $\ell > \kappa$, let $\mathcal{R}$ denote a random function from the set of functions $\{0,1\}^\ell \to \{0,1\}^\kappa$. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2^{(\cdot)})$, where $\mathcal{A}_1$ is an unbounded algorithm that outputs $z \in \{0,1\}^\zeta$, and $\mathcal{A}_2^{(\cdot)}$ is an unbounded algorithm that makes T oracle queries and outputs

$$\mathbf{X} := \Big((X_1^0, X_1^1), (X_2^0, X_2^1), \ldots, (X_K^0, X_K^1)\Big) \in \{0,1\}^{\ell \times 2K}.$$

Then,

$$\Pr_{\mathcal{R},\mathcal{A}}\left[\begin{array}{c|l} \forall (i,b)\neq(j,c), X_i^b \neq X_j^c, & z \leftarrow \mathcal{A}_1(\mathcal{R}) \\ \forall i, \mathcal{R}(X_i^0)=\mathcal{R}(X_i^1) & \left((X_1^0,X_1^1),(X_2^0,X_2^1),\ldots,(X_K^0,X_K^1)\right) \leftarrow \mathcal{A}_2^{\mathcal{R}}(z) \end{array}\right] \leq 2^{-K(\kappa-2\log T\log K)+\zeta}$$

Proof. We assume w.l.o.g that $\mathcal{A}$ is deterministic and that the oracle queries made by $\mathcal{A}_2$ are always distinct. Let $\mathcal{S}$ be the set of oracles $\mathcal{R}$ for which $\mathcal{A}$ successfully finds a distinct strong K-collision. We show that

$$|\mathcal{S}| \leq 2^{\kappa\cdot 2^\ell - K(\kappa-2\log T\log K)+\zeta},$$

which suffices since the total number of oracles $\mathcal{R}$ is $2^{\kappa\cdot 2^\ell}$.

Fix any such $\mathcal{R} \in \mathcal{S}$, and consider a corresponding execution of $\mathcal{A}$. Let z be the resulting auxiliary input, let $\mathbf{X} = \left((X_1^0,X_1^1),(X_2^0,X_2^1),\ldots,(X_K^0,X_K^1)\right)$ be the resulting distinct strong K-collision, and let $Q=\{Q_1,\ldots,Q_T\}$ be the set of oracle queries that $\mathcal{A}_2^{\mathcal{R}}(z)$ makes. We represent $\mathcal{R}$ as follows:

- z,
- $L_{Q\cap X} = \{(i,j)\in[T]\times[k] \mid \exists b \text{ such that } Q_i = X_j^b\}$,
- $\mathcal{R}(X_1^0),\mathcal{R}(X_2^0),\ldots,\mathcal{R}(X_k^0)$.
- $L_{Q\setminus X} := \{\mathcal{R}(Q_i) \mid Q_i \notin X\}$,
- $L_{\overline{Q\cup X}} := \{R(Y) \mid Y \notin X\cup Q\}$.

Note that the auxiliary input size is ζ, the set $L_{Q\cap X}$ can be represented by at most $|\mathbf{X}|\cdot\log T\cdot\log K = 2K\cdot\log T\cdot\log K$, the values $\mathcal{R}(X_1^0),\mathcal{R}(X_2^0),\ldots,\mathcal{R}(X_k^0)$ by $K\kappa$ bits, and the last two sets $L_{Q\setminus X}$ and $L_{Q\cup X}$ by $\kappa\cdot(2^\ell-|\mathbf{X}|) = \kappa\cdot(2^\ell-2K)$ bits (together). In sum, this representation costs

$$\kappa\cdot 2^\ell - 2K(\kappa-\log T\log K)+\zeta+K\kappa$$

which equals

$$\kappa\cdot 2^\ell - K(\kappa-2\log T\log K)+\zeta$$

as required. To see that this representation is unique, note that it allows to reconstruct $\mathcal{R}$ as follows. First emulate $\mathcal{A}_2^R(z)$. When it makes its i^{th} query Q_i, if there exists j such that $(i,j)\in L_{Q\cap X}$, answer with $\mathcal{R}(X_j^0)$. Otherwise answer from $L_{Q\setminus X}$, and keep track of the current location in the list. Finally, obtain all of $\mathbf{X}$. At this point, we have all the pairs $(Y,\mathcal{R}(Y))$ such that $Y\in Q\cup X$, and we can complete $L_{\overline{Q\cup X}}$ to a full description of the function $\mathcal{R}$. □

Corollary B.1. For $\ell > \kappa$, let $\mathcal{R}$ denote a random function from the set of functions $\{0,1\}^\ell \to \{0,1\}^\kappa$. Let $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2^{(\cdot)})$, where $\mathcal{A}_1$ is an unbounded algorithm that outputs $z\in\{0,1\}^\zeta$, and $\mathcal{A}_2^{(\cdot)}$ is an unbounded algorithm that makes $T=\mathsf{poly}(\kappa)$ oracle queries and outputs

$$\mathbf{X} := \left((X_1^0,X_1^1),(X_2^0,X_2^1),\ldots,(X_K^0,X_K^1)\right) \in \{0,1\}^{\ell\times 2K}.$$

Then for $K = \zeta = \mathsf{poly}(\kappa)$,

$$\Pr_{\mathcal{R},\mathcal{A}}\left[\begin{array}{c} \forall (i,b) \neq (j,c), X_i^b \neq X_j^c, \\ \forall i, \mathcal{R}(X_i^0) = \mathcal{R}(X_i^1) \end{array} \middle| \begin{array}{l} z \leftarrow \mathcal{A}_1(\mathcal{R}) \\ \left((X_1^0, X_1^1), (X_2^0, X_2^1), \ldots, (X_K^0, X_K^1)\right) \leftarrow \mathcal{A}_2^{\mathcal{R}}(z) \end{array}\right] \leq 2^{-\kappa/2}$$

Proof. Since $2 \log T \log K < \frac{\kappa}{2}$, we have that $2^{-K(\kappa - 2\log T \log K)+\zeta} < 2^{-K(\kappa/2)+\zeta}$. Thus, the theorem above implies that $\mathcal{A}$ on advice of length ζ outputs more than $K = \zeta$ distinct strong collisions with probability at most $2^{-\kappa/2}$. □

C Analysis of Section 4

C.1 Analysis of Construction

Please refer to the construction in Section 4.

Lemma C.1. $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is a correct equivocal commitment scheme as per Definition 4.2.

Proof. We can see that

$$\mathsf{AuxEquiv.Decom}(\mathsf{aux}, c, d) = \mathsf{AuxEquiv.Decom}(\mathsf{aux}, (g, \mathsf{SExt}(g, v) \oplus b, \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v))), v).$$

Since $\mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v))$, we output $\mathsf{SExt}(g, v) \oplus (\mathsf{SExt}(g, v) \oplus b) = b$, which is correct. □

Lemma C.2. $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is an efficient equivocal commitment scheme as per Definition 4.3.

Proof. We can see both AuxEquiv.Com and AuxEquiv.Decom simply sample a polynomial number of random bits and evaluate the strong extractor and keyless collision resistant hash function on some aux length, both of which are efficient in $\mathsf{poly}(\kappa, |\mathsf{aux}|)$. For AuxEquiv.Equivocate, we note that it does some efficient computation (including evaluating the hash), for each of $2^{3\kappa}$ items in the domain of the seeded extractor, giving it runtime $\mathsf{poly}(\kappa, |\mathsf{aux}|) \cdot 2^{3\kappa} \in \mathsf{poly}(2^\kappa, |\mathsf{aux}|)$. □

Lemma C.3. If $\mathsf{H}(\cdot)$ is a (T, K) distinct strong multi-collision resistant keyless hash function against non-uniform adversaries, then the above is a (T, K) binding secure commitment scheme as per Definition 4.4.

Proof. We will show that if there exists an algorithm $\mathcal{A}$ given non-uniform polynomial $\mathsf{advice}(\kappa)$ running in time $\mathsf{poly}(\mathsf{T}(\kappa))$ such that for $K = \mathsf{K}(|\mathsf{advice}|)$ $\mathcal{A}$ has non-negligible advantage given by the polynomial $p(\cdot)$ for infinitely many $\kappa \in \mathbb{N}$,

$$\Pr\left[\begin{array}{c} \left((\mathsf{aux}^{(1)}, c^{(1)}, d_0^{(1)}, d_1^{(1)}), \ldots, \right. \\ \left. (\mathsf{aux}^{(K)}, c^{(K)}, d_0^{(K)}, d_1^{(K)})\right) \leftarrow \mathcal{A}(1^\kappa) \end{array} : \begin{array}{c} \forall i \in [K], \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_0^{(i)}) = 0, \\ \mathsf{AuxEquiv.Decom}(\mathsf{aux}^{(i)}, c^{(i)}, d_1^{(i)}) = 1 \\ \forall i \neq j \in [K], \mathsf{aux}^{(i)} \neq \mathsf{aux}^{(j)} \end{array}\right] \geq \frac{1}{p(\kappa)}.$$

Then there exists an algorithm $\mathcal{B}$ given non-uniform polynomial $\mathsf{advice}(\kappa)$ running in time $\mathsf{poly}(\mathsf{T}(\kappa))$ such that for $K = \mathsf{K}(|\mathsf{advice}|)$ and for infinitely many $\kappa \in \mathbb{N}$,

$$\Pr\left[\left(X_1^{(0)}, X_1^{(1)}, \ldots, X_K^{(0)}, X_K^{(1)}\right) \leftarrow \mathcal{B}(1^\kappa) : \begin{array}{c} \forall (i,b) \neq (j,c) \in [K] \times \{0,1\}, X_i^{(b)} \neq X_j^{(c)} \\ \forall i \in [K], \mathsf{H.Hash}(1^\kappa, X_i^{(0)}) = \mathsf{H.Hash}(1^\kappa, X_i^{(1)}) \end{array}\right] \geq \frac{1}{p(\kappa)}.$$

Consider the $\mathcal{B}$ which simply runs $\mathcal{A}$ with advice to output $\forall i \in [K], (\mathsf{aux}^{(i)}, c^{(i)}, d_0^{(i)}, d_1^{(i)}) \leftarrow \mathcal{A}(1^\kappa)$ and outputs $X_i^{(0)} = (\mathsf{aux}^{(i)}, d_0^{(i)}), X_i^{(1)} = (\mathsf{aux}^{(i)}, d_1^{(i)})$. Assume without loss of generality that $\mathsf{AuxEquiv.Decom}(c^{(i)}, d_0^{(i)}) = 0 \wedge \mathsf{AuxEquiv.Decom}(c^{(i)}, d_1^{(i)}) = 1$. Let $c^{(i)} = g, w, h$, so by our definition of $\mathsf{AuxEquiv.Decom}$, we know that for every $i \in [K]$, $\mathsf{H.Hash}(1^\kappa, X_i^{(0)}) = h' = \mathsf{H.Hash}(1^\kappa, X_i^{(1)})$.

Since $\mathsf{SExt}(g^{(i)}, d_0^{(i)}) \oplus w^{(i)} = 0 \neq 1 = w^{(i)} \oplus \mathsf{SExt}(g^{(i)}, d_1^{(i)})$, we can conclude that $d_0^{(i)} \neq d_1^{(i)}$. In addition, for $i \neq j$, $\mathsf{aux}^{(i)} \neq \mathsf{aux}^{(j)}$. Thus, we have that $X_i^{(b)} \neq X_j^{(c)}$ for any b, c. From the securify definition (Definition 4.1), we have a contradiction.

□

Lemma C.4. If SExt is a $(k, \epsilon) = (\kappa, \mathsf{negl}(\kappa))$ Strong Seeded extractor, then the above is an equivocal commitment scheme as per Definition 4.5.

Proof. The proof will proceed through a series of hybrids, where we consider the distribution of $(\mathsf{aux}, c, d) = (\mathsf{aux}, (g, w, h), v)$. We will show that the output of each hybrid is statistically close to the previous.

$\mathcal{D}_0$: This is the distribution initial distribution (c, d) generated by $\mathsf{AuxEquiv.Com}(1^\kappa, \mathsf{aux}, b)$

- $g \xleftarrow{R} \{0,1\}^\kappa$
- $v \xleftarrow{R} \{0,1\}^{3\kappa}$
- $h \leftarrow \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v))$
- $w \leftarrow \mathsf{SExt}(g, v) \oplus b$

$\mathcal{D}_1$:Here, we select v from $\mathcal{V}_t$

- $g \xleftarrow{R} \{0,1\}^\kappa$
- Sample $t \xleftarrow{R} \{0,1\}^{3\kappa}$
- Define $\mathcal{V}_t = \{v : \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))\}$
- $h \leftarrow \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))$
- $v \xleftarrow{R} \mathcal{V}_t$
- $w \leftarrow \mathsf{SExt}(g, v) \oplus b$

$\mathcal{D}_2$:We check that the t we picked can be equivocated

- $g \xleftarrow{R} \{0,1\}^\kappa$
- Sample $t \xleftarrow{R} \{0,1\}^{3\kappa}$
- Define $\mathcal{V}_t = \{v : \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))\}$

- Partition $\mathcal{V}_t = \mathcal{V}_t^0 \cup \mathcal{V}_t^1$ where $\mathcal{V}_t^i = \{v : v \in \mathcal{V}_t \wedge \mathsf{SExt}(g, v) = i\}$, output bots if either $\mathcal{V}_t^0$ or $\mathcal{V}_t^1$ are $\emptyset$.
- $h \leftarrow \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))$
- $v \xleftarrow{R} \mathcal{V}_t$
- $w \leftarrow \mathsf{SExt}(g, v) \oplus b$

$\mathcal{D}_3$:Here, we switch to picking w randomly.

- $g \xleftarrow{R} \{0,1\}^\kappa$
- Sample $t \xleftarrow{R} \{0,1\}^{3\kappa}$
- $w \xleftarrow{R} \{0,1\}$
- Define $\mathcal{V}_t = \{v : \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))\}$
- Partition $\mathcal{V}_t = \mathcal{V}_t^0 \cup \mathcal{V}_t^1$ where $\mathcal{V}_t^i = \{v : v \in \mathcal{V}_t \wedge \mathsf{SExt}(g, v) = i\}$, output bots if either $\mathcal{V}_t^0$ or $\mathcal{V}_t^1$ are $\emptyset$.
- $h \leftarrow \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))$
- $v \xleftarrow{R} \mathcal{V}_t^{b \oplus w}$

$\mathcal{D}_4$:This is the final distribution (c, d_b) generated by $\mathsf{AuxEquiv.Equivocate}(1^\kappa, \mathsf{aux})$

- $g \xleftarrow{R} \{0,1\}^\kappa$
- Sample $t \xleftarrow{R} \{0,1\}^{3\kappa}$
- $w \xleftarrow{R} \{0,1\}$
- Compute $\mathcal{V}_t = \{v : \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))\}$
- Partition $\mathcal{V}_t = \mathcal{V}_t^0 \cup \mathcal{V}_t^1$ where $\mathcal{V}_t^i = \{v : v \in \mathcal{V}_t \wedge \mathsf{SExt}(g, v) = i\}$, output bots if either $\mathcal{V}_t^0$ or $\mathcal{V}_t^1$ are $\emptyset$.
- Sample $v_0 \xleftarrow{R} \mathcal{V}_t^w, v_1 \xleftarrow{R} \mathcal{V}_t^{w \oplus 1}$. Output bots if no such v_0 or v_1 exist.
- $h \leftarrow \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))$
- $v \leftarrow v_b$

Before proving statistical indistinguishability of these distributions, we prove the following simple claims to help our arguments.

Claim C.1. For any $\mathsf{aux} \in \{0,1\}^*$, let $\mathcal{V}_t$ denote the distribution of v' sampled uniformly at random conditioned on $\mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v'))$. With probability at least $1 - 2^{-\kappa}$ over the randomness of sampling $t \leftarrow \{0,1\}^{3\kappa}$,

$$H_\infty(\mathcal{V}_t) > \kappa.$$

Proof. We will prove this claim by a counting argument. Towards a contradiction, assume that with probability greater than $2^{-\kappa}$ over the randomness of sampling $t \leftarrow \{0,1\}^{3\kappa}$,

$$H_\infty(\mathcal{V}_t) \leq \kappa$$

This implies that there exists a set $\mathbb{S}$ consisting of elements in $\{0,1\}^{3\kappa}$ such that:

- $|\mathbb{S}| > 2^{-\kappa} \cdot 2^{3\kappa} = 2^{2\kappa}$ and,
- For every $t \in \mathbb{S}$,

$$H_\infty(\mathcal{V}_t) \leq \kappa$$

This implies that $\mathbb{S}$ can be partitioned into at least $N = |\mathbb{S}|/2^\kappa > 2^\kappa$ sets $\mathbb{S}_1, \ldots \mathbb{S}_N$ such that $\exists(h_1, \ldots h_N)$ satisfying the following constraints:

1. For all $i \in [N], y \in \mathbb{S}_i$, $\mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, y)) = h_i$,
2. For every $i, j \in [N]$ such that $i \neq j$, $h_i \neq h_j$.

Note that for all $i \in [N], h_i \in \{0,1\}^\kappa$ (since h_i is the output of the hash function). This, together with the fact that $N > 2^\kappa$ contradicts point 2 above, which completes our proof. □

Claim C.2. For any $t \in \{0,1\}^{3\kappa}$, for at least $1 - 2^{-\kappa}$ fraction of $t \in \{0,1\}^{3\kappa}$,

$$\Delta\Big((U_\kappa, t, \mathsf{SExt}(U_\kappa, \mathcal{V}_t)), (U_\kappa, t, U_1)\Big) \leq \epsilon(\kappa)$$

where U_x denotes the uniform distribution over $\{0,1\}^x$ and where $\mathcal{V}_t$ denotes the distribution of ν sampled uniformly at random conditioned on $\mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, \nu))$ and SExt is a $(\kappa, \epsilon(\kappa))$ Strong Extractor.

Proof. To prove this, we first reiterate that by Claim C.1, with probability at least $1 - 2^{-\kappa}$ over the randomness of sampling $t \leftarrow \{0,1\}^{3\kappa}$,

$$H_\infty(\mathcal{V}_t) > \kappa \tag{1}$$

where $\mathcal{V}_t$ denotes the distribution of ν sampled uniformly at random conditioned on $\mathsf{H.Hash}(1^\kappa, t) = \mathsf{H.Hash}(1^\kappa, \nu)$.

By definition of the $(\kappa, \epsilon(\kappa))$ strong seeded extractor, for any distribution $\mathcal{V}$ such that $H_\infty(\mathcal{V}) = \kappa$,

$$\Delta\Big((U_\kappa, \mathsf{SExt}(U_\kappa, \mathcal{V})), (U_\kappa, U_1)\Big) \leq \epsilon(\kappa) \tag{2}$$

where Δ denotes statistical distance and U_x denotes the uniform distribution over $\{0,1\}^x$. Combining Equation 1 and Equation 2, with probability $1 - 2^{-\kappa}$ over the randomness of sampling $t \leftarrow \{0,1\}^{3\kappa}$,

$$\Delta\Big((U_\kappa, t, \mathsf{SExt}(U_\kappa, \mathcal{V}_t)), (U_\kappa, t, U_1)\Big) \leq \epsilon(\kappa) \tag{3}$$

This completes the proof. □

Claim C.3. The distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are identical.

Proof. Notice the distribution only changes the way v is selected, so it suffices to show that for any fixed v', $v = v'$ is selected with equal probability in $\mathcal{D}_0$ and $\mathcal{D}_1$. In $\mathcal{D}_0$, this probability is clearly $2^{-3\kappa}$. In $\mathcal{D}_1$, let us consider $\mathcal{V}'$ be the set of $\{v : \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v)) = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, v'))\}$. The probability that the t chosen is $\in \mathcal{V}'$ is equal to $|\mathcal{V}'| \cdot 2^{-3\kappa}$, and the probability that the v chosen is v' conditioned on t chosen $\in \mathcal{V}'$ is then $\frac{1}{|\mathcal{V}'|}$, making the total probability

$$|\mathcal{V}'| \cdot 2^{-3\kappa} \cdot \frac{1}{|\mathcal{V}'|} = 2^{-3\kappa}$$

as well (Note if $t \notin \mathcal{V}'$, then the probability of selecting v' is 0). □

Claim C.4. The distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are statistically close - i.e. $\Delta\left(\mathcal{D}_1, \mathcal{D}_2\right) = \mathsf{negl}(\kappa)$.

Proof. The only difference in the two distributions is that we output bots when either $\mathcal{V}_t^0$ or $\mathcal{V}_t^1$ are $\emptyset$. From Claim C.1, we know that the set $\mathcal{V}_t$ has enough entropy to apply the strong extractor guarantee. From Claim C.2, we know that for at least $1 - 2^{-\kappa}$ fraction of $t \in \{0,1\}^{3\kappa}$,

$$\Delta\Big((U_\kappa, t, \mathsf{SExt}(U_\kappa, \mathcal{V}_t)), (U_\kappa, t, U_1)\Big) \leq \epsilon(\kappa).$$

Note that if we output bot, then the statistical difference when g, t were chosen is exactly half, i.e.

$$\Delta\Big((g, t, \mathsf{SExt}(g, \mathcal{V}_t)), (g, t, U_1)\Big) = \frac{1}{2}$$

because the extractor is always outputting either 0 or 1. Let us define the bad set as following,

$$\mathsf{BAD}_\kappa = \left\{(g,t) \text{ such that } t \in \{0,1\}^{3\kappa}, g \in \{0,1\}^\kappa, \Delta\Big((g, t, \mathsf{SExt}(g, \mathcal{V}_t)), (g, t, U_1)\Big) > \sqrt{\epsilon(\kappa)}\right\}.$$

By an averaging argument, for at least $1 - 2^{-\kappa}$ fraction of $t \in \{0,1\}^{3\kappa}$ it is the case that for at least $1 - \sqrt{\epsilon(\kappa)}$ fraction of $g \in \{0,1\}^\kappa$,

$$\Delta\Big((g, t, \mathsf{SExt}(g, \mathcal{V}_t)), (g, t, U_1)\Big) \leq \sqrt{\epsilon(\kappa)} \tag{4}$$

Equation 4 implies that

$$\Pr_{g \leftarrow \{0,1\}^\kappa, t \leftarrow \{0,1\}^{3\kappa}}[(g,t) \in \mathsf{BAD}_\kappa] \leq 2^{-\kappa} \cdot \sqrt{\epsilon(\kappa)} \tag{5}$$

Furthermore, since $\sqrt{\epsilon(\kappa)} < \frac{1}{2}$ for a negligible function ϵ, for every $(g,t) \notin \mathsf{BAD}_\kappa$ we have that bot will not be output.

This implies that

$$\Delta(\mathcal{D}_1, \mathcal{D}_2) \leq 2^{-\kappa} \cdot \sqrt{\epsilon(\kappa)}$$

which completes the proof of the claim. □

Claim C.5. The distributions $\mathcal{D}_2$ and $\mathcal{D}_3$ are statistically close - i.e. $\Delta\left(\mathcal{D}_2, \mathcal{D}_3\right) = \mathsf{negl}(\kappa)$.

Proof. $\mathcal{D}_2$ and $\mathcal{D}_3$ differ in the way v, w are sampled. In $\mathcal{D}_2$, v is sampled randomly from the set $\mathcal{V}_t$ and w is set to $\mathsf{SExt}(g, v) \oplus b$. In $\mathcal{D}_3$, w is a randomly sampled bit and v is sampled randomly from the set $\mathcal{V}_t^{b \oplus w}$. We wish to show the following for all $b \in \{0,1\}$,

$$\Delta\Big((\mathsf{aux}, g, \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t)), \mathsf{SExt}(g, \mathcal{V}_t) \oplus b, \mathcal{V}_t), (\mathsf{aux}, g, \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t)), U_1, \mathcal{V}_t^{b \oplus U_1})\Big) = \mathsf{negl}(\kappa).$$

Note that the choice of g, v, b deterministically fixes $w = \mathsf{SExt}(g, v) \oplus b$ and $h = \mathsf{H.Hash}(1^\kappa, (\mathsf{aux}, t))$ in both distributions and independently of all other variables in both games, so it suffices to show for $t \xleftarrow{R} \{0,1\}^{3\kappa}$,

$$\Delta\Big((g, \mathcal{V}_t), (g, \mathcal{V}_t^{b \oplus U_1})\Big) = \mathsf{negl}(\kappa).$$

First observe that the distributions

$$(g, \mathcal{V}_t) \stackrel{d}{=} (g, \mathcal{V}_t^{\mathsf{SExt}(g, \mathcal{V}_t)})$$

as we can see the probability of any fixed v' being picked is

$$\Pr[v = v'] = \Pr[v = v' | \mathsf{SExt}(g, v) = \mathsf{SExt}(g, v')] \cdot \Pr[\mathsf{SExt}(g, v) = \mathsf{SExt}(g, v')]$$

As a direct consequence of Claim C.2 we know that with all but $2^{-\kappa}$ probability over the choice of t,

$$\Delta\Big((g, \mathsf{SExt}(g, \mathcal{V}_t)), (g, U_1)\Big) = \mathsf{negl}(\kappa)$$

Since $b \oplus U_1$ is still the uniform distribution for any $b \in \{0, 1\}$, we can conclude,

$$\Delta\Big((g, \mathsf{SExt}(g, \mathcal{V}_t)), (g, b \oplus U_1)\Big) = \mathsf{negl}(\kappa)$$

. Thus,

$$\Delta\Big((g, \mathcal{V}_t^{\mathsf{SExt}(g, \mathcal{V}_t)}), (g, \mathcal{V}_t^{b \oplus U_1})\Big) = \mathsf{negl}(\kappa) + 2^{-\kappa}$$

which proves our above claim. □

Claim C.6. The distributions $\mathcal{D}_3$ and $\mathcal{D}_4$ are identical.

Proof. The two distributions are exactly same. In distribution $\mathcal{D}_4$, we sample both v_0, v_1 and output v_b. In distribution $\mathcal{D}_3$, we only sample v_b and output the same. □

From the above claims, we can conclude that the construction is statistically equivocal according to Definition 4.5. □

C.2 Analysis of Amplification

Claim C.7. (AuxEquiv.Com, AuxEquiv.Decom, AuxEquiv.Equivocate) is an efficient setupless equivocal commitment scheme.

Proof. Recall AuxEquiv.Com, AuxEquiv.Decom, AuxEquiv.Equivocate simply run the respective Small.AuxEquiv.Com, Small.AuxEquiv.Decom, Small.AuxEquiv.Equivocate $p(\kappa) \in \mathsf{poly}(\kappa)$ times. By the efficiency of the underlying scheme, we can see $\mathsf{poly}(\kappa, |\mathsf{aux}|) \cdot \mathsf{poly}(\kappa) \in \mathsf{poly}(\kappa, |\mathsf{aux}|)$ and $\mathsf{poly}(2^\kappa, |\mathsf{aux}|) \cdot \mathsf{poly}(\kappa) \in \mathsf{poly}(2^\kappa, |\mathsf{aux}|)$ □

Claim C.8. (AuxEquiv.Com, AuxEquiv.Decom, AuxEquiv.Equivocate) is a correct setupless equivocal commitment scheme.

Proof. By correctness of Small.AuxEquiv, $\mathsf{Small.AuxEquiv.Decom}((\mathsf{aux}, i),$ $\mathsf{Small.AuxEquiv.Com}(1^\kappa, (\mathsf{aux}, i), b)) = b$, so AuxEquiv.Decom returns b as well. □

Claim C.9. (AuxEquiv.Com, AuxEquiv.Decom, AuxEquiv.Equivocate) is a $(\mathsf{T}(\cdot), \mathsf{K}(\cdot)/p(\kappa))$ setupless equivocal commitment scheme.

Proof. We will show that if there exists an algorithm $\mathcal{A}$ given non-uniform polynomial $\mathsf{advice}(\kappa)$ running in time $\mathsf{poly}(\mathsf{T}(\kappa))$ such that $\mathcal{A}$ outputs $K = \mathsf{K}(\cdot)/p(\kappa)$ equivocations on AuxEquiv, there exists $\mathcal{B}$ has outputs $\mathsf{K}(\cdot)$ equivocations on Small.AuxEquiv.

$$\text{Let } \Big((\mathsf{aux}^{(1)}, c^{(1)}, d_0^{(1)}, d_1^{(1)}), \ldots, (\mathsf{aux}^{(K)}, c^{(K)}, d_0^{(K)}, d_1^{(K)})\Big) \leftarrow \mathcal{A}(1^\kappa)$$

For each $i \in [K]$

- Parse $c^{(i)}$ as $(c_1^{(i)}, \ldots, c_{p(\kappa)}^{(i)})$
- Parse $d_0^{(i)}$ as $(d_{0,1}^{(i)}, \ldots, d_{0,p(\kappa)}^{(i)})$
- Parse $d_1^{(i)}$ as $(d_{1,1}^{(i)}, \ldots, d_{1,p(\kappa)}^{(i)})$
- Output $\left((\mathsf{aux}^{(i)}, 1), c_1^{(i)}, d_{0,1}^{(i)}, d_{1,1}^{(i)}\right), \ldots, \left((\mathsf{aux}^{(i)}, p(\kappa)), c_{p(\kappa)}^{(i)}, d_{0,p(\kappa)}^{(i)}, d_{1,p(\kappa)}^{(i)}\right)$

Recall by correctness $\mathsf{Small.AuxEquiv.Decom}((\mathsf{aux}^{(i)}, j), c_j^{(i)}, d_{b,j}^{(i)}) = b$, so $\mathcal{B}$ has successfully output $p(\kappa) \cdot K = \mathsf{K}(\cdot)$ equivocations. Note that since $\mathsf{aux}^{(i)} \neq \mathsf{aux}^{(i')}$ for $i \neq i'$, we can see that all the aux output by $\mathcal{B}$ are not equal as well.

$\mathcal{B}$ simply runs $\mathcal{A}$ and returns it's output, so also runs in $\mathsf{poly}(\mathsf{T}(\kappa))$ time.

□

Claim C.10. $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is an equivocal setupless equivocal commitment scheme.

Proof. Consider the distributions $\mathcal{D}_j$ defined as follows:

- For $i \in [j]$, set $(c_i, d_i) \leftarrow \mathsf{Small.AuxEquiv.Com}(1^\kappa, (\mathsf{aux}, i), b)$
- For $i \in [j{+}1, p(\kappa)]$, compute $\mathsf{Small.AuxEquiv.Equivocate}(1^\kappa, \mathsf{aux}) \to (c^*, d_0^*, d_1^*)$ and set $(c_i, d_i) = (c^*, d_b^*)$
- Output $\mathsf{aux}, \{c_i\}_{i \in [p(\kappa)]}, \{d_i\}_{i \in [p(\kappa)]}$

Notice that each $\mathcal{D}_{j-1}$ and $\mathcal{D}_j$ only differ on c_j, d_j. By the equivocal property of $\mathsf{Small.AuxEquiv}$, these distributions have negligible statistical distance. Since $\mathcal{D}_0$ is exactly the output of $\mathsf{AuxEquiv.Com}$ and $\mathcal{D}_{p(\kappa)}$ is exactly the output of $\mathsf{AuxEquiv.Equivocate}$, since there are only a polynomial number of distributions, we can conclude that these distributions have negligible statistical distance as well.

□

D Constructing Hinting PRGs with Injective Extension

D.1 Security Analsyis of Construction from Section 5.2

Theorem D.1. Let $\mathsf{LossyHint}$ be a 2^{κ^δ}-secure $\rho(\kappa)$-lossy Lossy hinting function. Let $\mathcal{F}$ be a 2^{κ^δ}-secure κ^δ-leakage resilient injective one way function. Then the following construction 5.2 is a 2^{κ^δ}-secure hinting PRG with injective extension.

We will proceed through a sequence of series of experiments from $\beta = 0$ to $\beta = 1$ in the hinting PRG security game. Let $\mathcal{P}_{\mathcal{A}}^i$ be the probability that an adversary $\mathcal{A}$ returns 1 on game i. We will show that the difference in probability $\mathcal{A}$ returns 1 in the experiment where $\beta = 0$ and $\beta = 1$ are negligibly close.

Experiment 0 This is the distribution received by adversary $\mathcal{A}$ generate using the hinting PRGscheme above when $\beta = 0$.

- $n = (\rho(\kappa) + \ell)^{1/\delta}$
- $\mathsf{LossyHint.pp} \leftarrow \mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r_{Setup}})$
- $f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^n)$
- $t_1, \ldots t_\ell \xleftarrow{R} (\{0,1\}^n)^\ell$
- $s \xleftarrow{R} \{0,1\}^n$
- For $i \in [\ell]$ set $b_i \leftarrow \mathsf{hcb}(s, t_i)$
- $r_0 = b_1 b_2 \ldots b_\ell$
- $r_{\mathsf{ext}} = f(s)$
- $r_{i,s_i} = \mathsf{LossyHint.Eval}(\mathsf{pp}, s, i)$
- $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell \; \forall \; i \in [n]$
- Output $n, (\mathsf{LossyHint.pp}, f, t_1, \ldots t_\ell), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Experiment 1

- $n = (\rho(\kappa) + \ell)^{1/\delta}$
- $\mathsf{LossyHint.pp} \leftarrow \mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r_{Setup}})$
- $f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^n)$
- $t_1, \ldots t_\ell \xleftarrow{R} (\{0,1\}^n)^\ell$
- $s \xleftarrow{R} \{0,1\}^n$
- $\mathsf{LossyHint.hint} = \mathsf{LossyHint.Sim}_1(\mathsf{r_{Setup}}, \mathsf{LossyHint.pp}, s)$
- For $i \in [\ell]$ set $b_i \leftarrow \mathsf{hcb}(s, t_i)$
- $r_0 = b_1 b_2 \ldots b_\ell$
- $r_{\mathsf{ext}} = f(s)$
- ~~$r_{i,s_i} = \mathsf{LossyHint.Eval}(\mathsf{pp}, s, i)$~~
- ~~$r_{i,s_i} \xleftarrow{R} \{0,1\}^\ell \; \forall \; i \in [n]$~~
- $r_{i,b} = \mathsf{LossyHint.Sim}_2(\mathsf{r_{Setup}}, \mathsf{LossyHint.hint}, \mathsf{LossyHint.pp}, i, b)$
- Output $n, (\mathsf{LossyHint.pp}, f, t_1, \ldots t_\ell), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Experiment 2.j for $j \in [\ell]$

- $n = (\rho(\kappa) + \ell)^{1/\delta}$
- $\mathsf{LossyHint.pp} \leftarrow \mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r_{Setup}})$
- $f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^n)$
- $t_1, \ldots t_\ell \xleftarrow{R} (\{0,1\}^n)^\ell$
- $s \xleftarrow{R} \{0,1\}^n$
- $\mathsf{LossyHint.hint} = \mathsf{LossyHint.Sim}_1(\mathsf{r_{Setup}}, \mathsf{LossyHint.pp}, s)$
- For $i \in [\ell]$ if $i \leq j$ set $b_i \xleftarrow{R} \{0,1\}$. Else set $b_i \leftarrow \mathsf{hcb}(s, t_i)$
- $r_0 = b_1 b_2 \ldots b_\ell$
- $r_{\mathsf{ext}} = f(s)$
- $r_{i,b} = \mathsf{LossyHint.Sim}_2(\mathsf{r_{Setup}}, \mathsf{LossyHint.hint}, \mathsf{LossyHint.pp}, i, b)$
- Output $n, (\mathsf{LossyHint.pp}, f, t_1, \ldots t_\ell), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Experiment 3

- $n = (\rho(\kappa) + \ell)^{1/\delta}$
- $\mathsf{LossyHint.pp} \leftarrow \mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r_{Setup}})$
- $f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^n)$
- $t_1, \ldots t_\ell \xleftarrow{R} (\{0,1\}^n)^\ell$
- $s \xleftarrow{R} \{0,1\}^n$
- $\mathsf{LossyHint.hint} = \mathsf{LossyHint.Sim}_1(\mathsf{r_{Setup}}, \mathsf{LossyHint.pp}, s)$
- For $i \in [\ell]$ set $b_i \xleftarrow{R} \{0,1\}$.
- $r_0 = b_1 b_2 \ldots b_\ell$
- $r_{\mathsf{ext}} = f(s)$
- ~~$r_{i,b} = \mathsf{LossyHint.Sim}_2(\mathsf{r_{Setup}}, \mathsf{LossyHint.hint}, \mathsf{LossyHint.pp}, i, b)$~~
- $r_{i,s_i} = \mathsf{LossyHint.Eval}(\mathsf{pp}, s, i)$
- $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell \; \forall \; i \in [n]$
- Output $n, (\mathsf{LossyHint.pp}, f, t_1, \ldots t_\ell), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Lemma D.1. If $\mathsf{LossyHint}$ is a 2^{κ^δ}-secure lossy hinting function, then for any $\mathsf{poly}(2^{\kappa^\delta})$ time algorithm $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathcal{P}^0_{\mathcal{A}} - \mathcal{P}^1_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$.

Proof. Let $\mathcal{A}$ be an algorithm for the above experiments. Consider $\mathcal{B}$, an adversary for the security game of lossy hinting functions

$$\mathcal{B}\left(\mathsf{LossyHint.pp}, \left(s, r^\beta_{\mathsf{ext}}, \left\{r^\beta_{i,b}\right\}_{i\in[n],b\in\{0,1\}}\right)\right)$$

- $f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^n)$
- $t_\ell, \ldots t_\ell \xleftarrow{R} (\{0,1\}^n)^\ell$
- For $i \in [\ell]$ set $b_i \leftarrow \mathsf{hcb}(s, t_i)$
- $r_0 = b_1 b_2 \ldots b_\ell$
- Run $\mathcal{A}(n, (\mathsf{LossyHint.pp}, f, t_1, \ldots t_\ell), r_0, r^\beta_{\mathsf{ext}}, \left\{r^\beta_{i,b}\right\}_{i\in[n],b\in\{0,1\}})$

We can observe that when $\beta = 0$, the input to $\mathcal{A}$ is exactly Experiment 0, and similarly, is Experiment 1 when $\beta = 1$. By the security of lossy hinting functions, the advantage of $\mathcal{B}$ in distinguishing β is negligible, and so $|\mathcal{P}^0_{\mathcal{A}} - \mathcal{P}^1_{\mathcal{A}}|$ must be negligible as well. □

Lemma D.2. For any adversary $\mathcal{A}$, $|\mathcal{P}^1_{\mathcal{A}} - \mathcal{P}^{2.0}_{\mathcal{A}}| = 0$.

Proof. These experiments are identical. □

Lemma D.3. If $\mathsf{LossyHint}$ is a $\rho(\cdot)$-lossy lossy hinting function and f is κ^δ leakage resilient one way function (both 2^{κ^δ}-secure), then for any $\mathsf{poly}(2^{\kappa^\delta})$ adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}, j \in [\ell]$, $|\mathcal{P}^{2.j-1}_{\mathcal{A}} - \mathcal{P}^{2.j}_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$.

Proof. Note Experiments $2.j-1$ and $2.j$ differ only in how bit b_j is computed. Note in the former, it is computed as $\mathsf{hcb}(s; t_j)$, while in the latter, it is a uniformly random bit. Consider the function $g[\mathsf{r_{Setup}}, t_{j+1}, \ldots t_\ell](s) = \mathsf{Sim}_1(\mathsf{r_{Setup}}, s), \mathsf{pp}, \mathsf{hcb}(s, t_{j+1}), \ldots, \mathsf{hcb}(s, t_\ell)$. Since the output of g is $l(\kappa) + \ell - j \leq \rho(\kappa) + \ell \leq n^\delta$ bits, by definition of leakage resilience, $f'(s) = f(s)||g[\mathsf{r_{Setup}}, t_{j+1}, \ldots t_\ell](s)$ is one way. Let $\mathcal{A}$ be an adversary for the above experiments. Consider the program $\mathcal{C}$

$$\mathcal{C}(f||g[\mathsf{r_{Setup}}, t_{j+1}, \ldots, t_\ell], y||(\mathsf{LossyHint.hint}, b_{j+1}, \ldots, b_\ell), t_j, b)$$

- $n = (\rho(\kappa) + \ell)^{1/\delta}$
- $\mathsf{LossyHint.pp} \leftarrow \mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r_{Setup}})$
- $t_1, \ldots t_{j-1} \xleftarrow{R} (\{0,1\}^n)^\ell$
- For $i \in [j-1]$, set $b_i \xleftarrow{R} \{0,1\}$
- $r_0 = b_1, \ldots, b_{j-1}, b, b_{j+1}, \ldots, b_\ell$
- $r_{\mathsf{ext}} = y$

- $r_{i,b} = \mathsf{LossyHint.Sim}_2(\mathsf{r}_{\mathsf{Setup}}, \mathsf{LossyHint.hint}, \mathsf{LossyHint.pp}, i, b)$
- Output $\mathcal{A}(n, (\mathsf{LossyHint.pp}, f, t_1, \ldots t_\ell), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}})$

Observe that when $b = \mathsf{hcb}(s, t_j)$, the distribution given to $\mathcal{A}$ is exactly that of Game $2.j-1$, and similarly, is Game $2.j$ when $b \xleftarrow{R} \{0,1\}$. By the security of hardcore bits on the function $f'(s)$, the advantage of $\mathcal{C}$ should be negligible in distinguishing these two cases, and so must be the difference $|\mathcal{P}^{2.j-1}_{\mathcal{A}} - \mathcal{P}^{2.j}_{\mathcal{A}}|$. □

Lemma D.4. If LossyHint is a 2^{κ^δ}-secure lossy hinting function, then for any $\mathsf{poly}(2^{\kappa^\delta})$ adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathcal{P}^{2.\ell}_{\mathcal{A}} - \mathcal{P}^{3}_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$.

Proof. This proof proceeds analogously to the Lemma D.1 Let $\mathcal{A}$ be an adversary for the above experiments. Consider $\mathcal{B}$, an adversary for the security game of lossy hinting functions

$$\mathcal{B}\left(\mathsf{LossyHint.pp}, \left(s, r^{\beta}_{\mathsf{ext}}, \left\{r^{\beta}_{i,b}\right\}_{i\in[n],b\in\{0,1\}}\right)\right)$$

- $f \leftarrow \mathsf{Gen}_{\mathcal{F}}(1^n)$
- $t_\ell, \ldots t_\ell \xleftarrow{R} (\{0,1\}^n)^\ell$
- For $i \in [\ell]$ set $b_i \xleftarrow{R} \{0,1\}$
- $r_0 = b_1 b_2 \ldots b_\ell$
- Run $\mathcal{A}(n, (\mathsf{LossyHint.pp}, f, t_1, \ldots t_\ell), r_0, r^{\beta}_{\mathsf{ext}}, \left\{r^{\beta}_{i,b}\right\}_{i\in[n],b\in\{0,1\}})$

We can observe that when $\beta = 1$, the input to $\mathcal{A}$ is exactly Experiment $2.\ell$, and similarly, is Experiment 3 when $\beta = 0$. By the security of lossy hinting functions, the advantage of $\mathcal{B}$ in distinguishing β is negligible, and so must $|\mathcal{P}^{2}_{\mathcal{A}} - \mathcal{P}^{3}_{\mathcal{A}}|$. □

Since Experiment 3 is exactly the distribution of $hintingPRG$ when $\beta = 1$, we can conclude that for any $\mathsf{poly}(2^{\kappa^\delta})$ time adversary $|\mathcal{P}^{0}_{\mathcal{A}} - \mathcal{P}^{3}_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$, which is it's advantage in the hinting PRG security game.

D.2 Lossy Hinting Functions from Decisional Diffie Hellman

We present a construction of lossy hinting functions from Decisional Diffie Hellman (DDH). The construction follows along the same lines of the construction of hinting PRGs from DDH as per [KW19]. But in the security proof, except going through the full sequence of experiments, we simply need to execute the DDH indistinguishability argument utilized in Experiment 0 to 1 in [KW19]. We omit the later statistical and computational arguments. For simplicity, we first present construction relying on the DDH assumption, but a similar construction of lossy hinting functions relying only on Computational Diffie Hellman (CDH) can be found in the following section.

$\mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r}_{\mathsf{Setup}})$

- Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.

$\mathsf{ExtractBits}(1^\kappa, 1^\ell, g_1, \ldots, g_m)$

- For each $i \in m$, interpret each g_i as a natural number in $[p]$[a]. We then extract ℓ bits from $g_1, \ldots, g_m$, where $p = |\mathcal{G}|$.
- Let $c = \sum_{i=1}^{m} (g_i - 1) \cdot p^{i-1}$.
- Output binary representation of $c \mod 2^\ell$.

[a] We assume there is an efficiently computable bijective mapping from $\mathcal{G} \to [p]$

Figure 7: Routine ExtractBits

- For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
- Let $m = \left\lceil \frac{\ell+\kappa}{\log p} \right\rceil$
- For $i \in [n], j \in [m], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
- For $i \in [n], j \in [m], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

- Output $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [m], b \in \{0,1\}}$

$\mathsf{LossyHint.Eval}\Big(\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [m], b \in \{0,1\}}, s \in \{0,1\}^n, i \in [n]\Big)$

- For $j \in [m]$, set $z_j = \prod_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k]$
- Output $\mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$

$\mathsf{LossyHint.Sim}_1(\mathsf{pp}, \mathsf{r}_{\mathsf{Setup}}, s)$

- Compute $h = \prod_{i=1}^{n} g_{i,s_i}$
- Output $\mathsf{hint} = h$

$\mathsf{LossyHint.Sim}_2(\mathsf{pp}, \mathsf{r}_{\mathsf{Setup}}, \mathsf{hint} = h, i \in [n], b \in \{0,1\})$

- For $j \in [m]$, set $z_j = h^{\rho_{i,j,b}}$
- Output $\mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$

Analysis We will proceed through a sequence of series of experiments from $\beta = 0$ to $\beta = 1$ in the initial lossy hinting function security game. Let $\mathcal{P}_{\mathcal{A}}^i$ be the probability that an adversary $\mathcal{A}$ returns 1 on experiment i. We will show that the difference in probability $\mathcal{A}$ returns 1 in the experiment where $\beta = 0$ and $\beta = 1$ are negligibly close.

Experiment 0 This is the output given to the adversary in the lossy hinting function security game when $\beta = 0$.

- Run LossyHint.Setup
  - Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
  - For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
  - Let $m = \left\lceil \frac{\ell+\kappa}{\log p} \right\rceil$
  - For $i \in [n], j \in [m], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
  - For $i \in [n], j \in [m], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

  - $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [m], b \in \{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate r_{i,s_i} for $i \in [n]$
  - For $j \in [m]$, set $z_j = \prod_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k]$
  - Set $r_{i,s_i} \leftarrow \mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$
- Generate $r_{i,\overline{s}_i}$ for $i \in [n]$
  - Set $r_i, \overline{s}_i \xleftarrow{R} \{0,1\}^\ell$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Experiment 1^k **for** $k \in [n]$

- Run LossyHint.Setup
  - Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
  - For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
  - Let $m = \left\lceil \frac{\ell+\kappa}{\log p} \right\rceil$
  - For $i \in [n], j \in [m], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
  - For $i \in [n], j \in [m], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

  - $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [m], b \in \{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$

- Generate r_{i,s_i} for $i \in [n]$
  - For $j \in [m]$, set $z_j = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,s_i}}$
  - Set $r_{i,s_i} \leftarrow \mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$
- Generate $r_{i,\bar{s}_i}$ for $i \in [n]$
  - If $i \leq k$
    For $j \in [m]$,
    * Set $z_j \xleftarrow{R} \mathcal{G}$
    Set $r_{i,\bar{s}_i} \leftarrow \mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}$

Experiment 2^k

- Run LossyHint.Setup
  - Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
  - For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
  - Let $m = \left\lceil \frac{\ell+\kappa}{\log p} \right\rceil$
  - For $i \in [n], j \in [m], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
  - For $i \in [n], j \in [m], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i,\bar{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

  - $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i\in[n],j\in[m],b\in\{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate r_{i,s_i} for $i \in [n]$
  - For $j \in [m]$, set $z_j = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,s_i}}$
  - Set $r_{i,s_i} \leftarrow \mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$
- Generate $r_{i,\bar{s}_i}$ for $i \in [n]$
  For $j \in [m]$,
  - If $i \cdot m + j - m \leq k$
    * Set $z_j = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,\bar{s}_i}}$
  - Else
    * Set $z_j \xleftarrow{R} \mathcal{G}$
  - Set $r_{i,\bar{s}_i} \leftarrow \mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}$

Experiment 3 In this experiment, all the $r_{i,b}$ values are computed using LossyHint.hint rather than multiplying the underlying $g_{i,j,b}$ group elements. This is exactly the output computed by $\mathsf{LossyHint.Sim}_1, \mathsf{LossyHint.Sim}_2$.

- Run LossyHint.Setup
  - Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
  - For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
  - Let $m = \left\lceil \frac{\ell+\kappa}{\log p} \right\rceil$
  - For $i \in [n], j \in [m], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
  - For $i \in [n], j \in [m], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \bar{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

  - $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i\in[n],j\in[m],b\in\{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate hint $= h = \prod_{k=1}^{n} g_{k,s_k}$
- Generate $r_{i,b}$ for $i \in [n], b \in \{0,1\}$
  - For $j \in [m]$, set $z_j = h^{\rho_{i,j,b}}$
  - Set $r_{i,b} \leftarrow \mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}$

Claim D.1. Suppose $g_1, \ldots g_m$ are uniformly random group elements. Then $\forall m \geq \frac{\ell+\kappa}{\log p}$ the output of $\mathsf{ExtractBits}(1^\kappa, 1^\ell, g_1, \ldots, g_m)$ is statistically close to a uniformly random bitstring on ℓ bits.

Proof. First, we observe that the c generated by ExtractBits is a uniformly random element in $[0, p^m - 1]$, as we can take the p-ary representation of c, and find the corresponding group elements which map to the appropriate digits. Since the mapping is surjective and $|\mathcal{G}|^m = p^m$, this mapping is bijective and hence c is uniform. Thus, for any $0 \leq c' < 2^\ell$, we can bound the probability that $c \mod 2^\ell = c'$ with $\left[\frac{\frac{p^m}{2^\ell}-1}{p^m}, \frac{\frac{p^m}{2^\ell}+1}{p^m}\right]$. Thus, we can bound the total statistical difference from the uniform distribution over all 2^ℓ outputs with

$$2^\ell \cdot \frac{2}{p^m} \leq 2 \cdot \frac{2^\ell}{p^{\frac{\ell+\kappa}{\log p}}} = \frac{2}{2^\kappa}$$

, which is negligible. □

Note since LossyHint.Setup calls ExtractBits with $m = \left\lceil \frac{\ell+\kappa}{\log p} \right\rceil$, the above claim holds.

Lemma D.5. For any adversary $\mathcal{A}$ $|\mathcal{P}^0_{\mathcal{A}} - \mathcal{P}^{1^0}_{\mathcal{A}}| = 0$.

Proof. In Experiment 1^k for $k \in [n]$, we deviate from Experiment 0 for values $i \in [n]$ where $i \leq k$. Since $k = 0$, the experiments are identical. □

Lemma D.6. For any adversary $\mathcal{A}$, $k \in [n]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathcal{P}^{1^{k-1}}_{\mathcal{A}} - \mathcal{P}^{1^k}_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$.

Lemma D.7. We observe Experiment 1^{k-1} and Experiment 1^k differ only in how $r_{k,\overline{s}_k}$ is generated. In Experiment 1^{k-1}, it is generated as $r_{k,\overline{s}_k} \xleftarrow{R} \{0,1\}^\ell$, while in Experiment 1^k, it is generated as $\mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$ for uniformly random z_i. From Claim D.1, these are statistically close.

Lemma D.8. For any adversary $\mathcal{A}$ $|\mathcal{P}^{1^n}_{\mathcal{A}} - \mathcal{P}^{2^0}_{\mathcal{A}}| = 0$.

Proof. In Experiment 2^k, we deviate from Experiment 1^n for values $i \in [n]$ and $j \in [m]$, where $i \cdot m + j - m \leq k$. Since $k = 0$ and $i, j \geq 1$, these experiments are identical. □

Lemma D.9. Assuming $\mathsf{poly}(2^{\log|\mathcal{G}|^\delta})$-hardness of DDH, for any $\mathsf{poly}(2^{\kappa^\delta})$ adversary $\mathcal{A}$, for any $k \in [n \cdot m]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}^{2^{k-1}}_{\mathcal{A}} - \mathsf{adv}^{2^k}_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$.

Proof. Observe these experiments only differ on $r_{i,j,\overline{s}_i}$ such that $i \cdot m + j - m = k$. Since $j \in [m]$, we can see there exists a unique i^*, j^* for this. Suppose $\mathcal{A}$ can distinguish between Experiments 2^{k-1} and 2^k with advantage ϵ. Then consider the following algorithm $\mathcal{B}$ which runs in $\mathsf{poly}(\mathcal{A})$ time and breaks DDH with advantage at least $\epsilon - \mathsf{negl}(\kappa)$.

$$\mathcal{B}\left(g, g_1 = g^a, g_2, h'\right)$$

- Generate $s \xleftarrow{R} \{0,1\}^\ell$
- For $(i,b) \in [n] \times \{0,1\} \backslash i^*, s^*$, set $a_{i,b}$ to be a uniformly random group elements of $\mathbb{Z}_p$
- For $(i,b) \in [n] \times \{0,1\} \backslash i^*, s^*$, set $g_{i,b} = g^{a_{i,b}}$ to be a uniformly random group element of $\mathcal{G}$
- Let $m = \frac{\ell + \kappa}{\log p}$
- For $(i,j,b) \in [n] \times [m] \times [b] \backslash \{(i^*, j^*, \overline{s}_{i^*})\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$
- Set $g_{i^*, s_{i^*}} = g_1 / \prod_{k \neq i^*} g_{k,s_k}$
- For $(i,j,b) \in [n] \times [m] \times [b] \backslash \{(i^*, j^*, \overline{s}_{i^*})\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ g_{k,\beta}^{a_{i,j,b}} \text{ otherwise} \end{cases}$$

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i^*,j^*,\overline{s}_{i^*}}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i^*, s_{i^*}) \\ g_2^{a_{k,\beta}} \text{ otherwise} \end{cases}$$

- $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [m], b \in \{0,1\}}$

- Generate $r_{i,s_i} \leftarrow \mathsf{LossyHint.Eval}(\mathsf{pp}, s, i)$
- Generate $r_{i,\overline{s}_i}$ for $i \in [n]$
  - For $j \in [m]$:
  - If $i \cdot m + j - m \leq k$
    - Set $z_j = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,\overline{s}_i}}$
  - Else If $i \cdot m + j - m = k$
    - Set $z_j = h'$
  - Else
    - Set $z_j \xleftarrow{R} \mathcal{G}$
  - Set $r_{i,\overline{s}_i} \leftarrow \mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$
- Run $\mathcal{A}(\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}})$

Observe that the above reduction computes public parameters LossyHint.pp consistent with setting $\rho_{i^*,j^*,\overline{s}_{i^*}} = a$ and $g_{i^*,s_{i^*}} = g_1 / \prod_{k\neq i^*} g_{k,s_k}$. Since g_1 is a uniformly random group element independent of g, $g_{i^*,s_{i^*}}$ is also a uniformly random element of $\mathcal{G}$ independent of other $g_{i,b}$. Now note that $\prod_{k=1}^{n} g_{k,s_k}^{\rho_{i^*,j^*,\overline{s}_{i^*}}} = g_2^a$, so $z_{j^*} = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i^*,j^*,\overline{s}_{i^*}}}$ when g, g_1, g_2, h' is a DDH triple, which is exactly the output of Experiment 2^k. Conversely, when h' is uniformly random, so is z_{j^*}, which is the output of Experiment 2^{k-1}.

□

Lemma D.10. For any adversary $\mathcal{A}$ $|\mathcal{P}_{\mathcal{A}}^{2^{n\cdot m}} - \mathcal{P}_{\mathcal{A}}^{3}| = 0$.

Proof. We can observe that since $h = \prod_{k=1}^{n} g_{k,s_k}$,

$$h^{\rho_{i,j,b}} = \left(\prod_{k=1}^{n} g_{k,s_k}\right)^{\rho_{i,j,b}} = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,b}}$$

so the z values computed are equal. □

D.3 Lossy Hinting Function from Computational Diffie Hellman

$\mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r}_{\mathsf{Setup}})$

- Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
- For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
- For $i \in [n], j \in [\ell], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
- For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i,\overline{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

Also choose each $t_{i,j,b}$ as independent randomness for hardcore bit extraction.

- Output $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}, t_{i,j,b}\}_{i\in[n],j\in[\ell],b\in\{0,1\}}$

$\mathsf{LossyHint.Eval}\Big(\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}, t_{i,j,b}\}_{i\in[n],j\in[\ell],b\in\{0,1\}}, s \in \{0,1\}^n, i \in [n]\Big)$

- For $j \in [\ell]$, set $z_j = \mathsf{hcb}\left(\prod_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k], t_{i,j,s_i}\right)$
- Output $z_1 z_2 \ldots z_\ell$

$\mathsf{LossyHint.Sim}_1(\mathsf{pp}, \mathsf{r}_{\mathsf{Setup}}, s)$

- Compute $h = \prod_{i=1}^{n} g_{i,s_i}$
- Output $\mathsf{hint} = h$

$\mathsf{LossyHint.Sim}_2(\mathsf{pp}, \mathsf{r}_{\mathsf{Setup}}, \mathsf{hint} = h, i \in [n], b \in \{0,1\})$

- For $j \in [\ell]$, set $z_j = \mathsf{hcb}\left(h^{\rho_{i,j,b}}, t_{i,j,b}\right)$
- Output $z_1 z_2 \ldots z_\ell$

Analysis We will proceed through a sequence of series of experiments from $\beta = 0$ to $\beta = 1$ in the initial lossy hinting function security game. Let $\mathcal{P}^i_{\mathcal{A}}$ be the probability that an adversary $\mathcal{A}$ returns 1 on experiment i. We will show that the difference in probability $\mathcal{A}$ returns 1 in the experiment where $\beta = 0$ and $\beta = 1$ are negligibly close.

Experiment 0 This is the output given to the adversary in the lossy hinting function security game when $\beta = 0$.

- Run $\mathsf{LossyHint.Setup}$
  - Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
  - For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$,

    $$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

    Also choose each $t_{i,j,b}$ as independent randomness for hardcore bit extraction.
  - $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}, t_{i,j,b}\}_{i\in[n],j\in[\ell],b\in\{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate r_{i,s_i} for $i \in [n]$
  - For $j \in [\ell]$, set $z_j = \mathsf{hcb}(\prod_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k], t_{i,j,s_i})$
  - Set $r_{i,s_i} = z_1 z_2 \ldots z_\ell$
- Generate $r_{i,\overline{s}_i}$ for $i \in [n]$
  - Set $r_{i,\overline{s}_i} \xleftarrow{R} \{0,1\}^\ell$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}$

Experiment 1^k **for** $k \in [\ell \cdot n]$

- Run LossyHint.Setup
  - Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
  - For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \bar{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

    Also choose each $t_{i,j,b}$ as independent randomness for hardcore bit extraction.
  - $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}, t_{i,j,b}\}_{i \in [n], j \in [\ell], b \in \{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate r_{i,s_i} for $i \in [n]$
  - For $j \in [\ell]$, set $z_j = \mathsf{hcb}(\prod_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k], t_{i,j,s_i})$
  - Set $r_{i,s_i} = z_1 z_2 \ldots z_\ell$
- Generate $r_{i,\bar{s}_i}$ for $i \in [n]$
  For $j \in [\ell]$,
  - If $i \cdot \ell + j - \ell \leq k$
    - Set $z_j = \mathsf{hcb}\left(\prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,\bar{s}_i}}, t_{i,j,\bar{s}_i}\right)$
  - Else
    - Set $z_j \xleftarrow{R} \{0,1\}$
  - Set $r_{i,\bar{s}_i} = z_1 z_2 \ldots z_\ell$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Experiment 2 In this experiment, all the $r_{i,b}$ values are computed using LossyHint.hint rather than multiplying the underlying $g_{i,j,b}$ group elements. This is exactly the output computed by $\mathsf{LossyHint.Sim}_1, \mathsf{LossyHint.Sim}_2$.

- Run LossyHint.Setup
  - Generate $\mathcal{G}$ as the description of a prime order group of order p, where p is a κ bit prime.
  - For $i \in [n], b \in \{0,1\}$, set $g_{i,b}$ to be a uniformly random group element of $\mathcal{G}$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$.

– For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \bar{b}) \\ g_{k,\beta}^{\rho_{i,j,b}} \text{ otherwise} \end{cases}$$

Also choose each $t_{i,j,b}$ as independent randomness for hardcore bit extraction.

– $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}, t_{i,j,b}\}_{i\in[n],j\in[\ell],b\in\{0,1\}}$

- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate hint $= h = \prod_{k=1}^{n} g_{k,s_k}$
- Generate $r_{i,b}$ for $i \in [n], b \in \{0,1\}$
  - For $j \in [\ell]$ Set $z_j = \mathsf{hcb}\left(h^{\rho_{i,j,b}}, t_{i,j,\bar{s}_i}\right)$
  - Set $r_{i,b} = z_1 z_2 \ldots z_\ell$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}$

Lemma D.11. For any adversary $\mathcal{A}$ $|\mathcal{P}_{\mathcal{A}}^{0} - \mathcal{P}_{\mathcal{A}}^{1^0}| = 0$.

Proof. In Experiment 1^k, we deviate from Experiment 0 for values $i \in [n]$ and $j \in [m]$, where $i \cdot m + j - m \leq k$. Since $k = 0$ and $i, j \geq 1$, these experiments are identical. □

Lemma D.12. For any adversary $\mathcal{A}$, $k \in [n]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{1^{k-1}} - \mathcal{P}_{\mathcal{A}}^{1^k}| \leq \mathsf{negl}(\kappa)$.

Lemma D.13. We observe Experiment 1^{k-1} and Experiment 1^k differ only in how $r_{k,\bar{s}_k}$ is generated. In Experiment 1^{k-1}, it is generated as $r_{k,\bar{s}_k} \xleftarrow{R} \{0,1\}^\ell$, while in Experiment 1^k, it is generated as $\mathsf{ExtractBits}(1^\kappa, 1^\ell, z_1, \ldots, z_m)$ for uniformly random z_i. From Claim D.1, these are statistically close.

Lemma D.14. For any adversary $\mathcal{A}$ $|\mathcal{P}_{\mathcal{A}}^{1^n} - \mathcal{P}_{\mathcal{A}}^{2^0}| = 0$.

Proof. In Experiment 2^k, we deviate from Experiment 1^n for values $i \in [n]$ and $j \in [m]$, where $i \cdot m + j - m \leq k$. Since $k = 0$ and $i, j \geq 1$, these experiments are identical. □

Lemma D.15. Assuming $\mathsf{poly}(2^{\log|\mathcal{G}|^\delta})$-hardness of CDH, for any $\mathsf{poly}(2^{\kappa^\delta})$ adversary $\mathcal{A}$, for any $k \in [n \cdot m]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}_{\mathcal{A}}^{1^{k-1}} - \mathsf{adv}_{\mathcal{A}}^{1^k}| \leq \mathsf{negl}(\kappa)$.

Proof. Observe these experiments only differ on $r_{i,j,\bar{s}_i}$ such that $i \cdot m + j - m = k$. Since $j \in [m]$, we can see there exists a unique i^*, j^* for this. Suppose $\mathcal{A}$ can distinguish between Experiments 1^{k-1} and 1^k with advantage ϵ. Then consider the following algorithm $\mathcal{B}$ which runs in $\mathsf{poly}(\mathcal{A})$ time and distinguish the following distributions

$$\mathcal{D}_1 = (g, g_1 = g^a, g_2, \mathsf{hcb}(g_2^a; t), t) \ : \ g, g_2 \xleftarrow{R} \mathcal{G}; a \xleftarrow{R} \mathbb{Z}_p, t \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{hcb}}}$$

$$\mathcal{D}_2 = (g, g_1 = g^a, g_2, b', t) \ : \ g, g_2 \xleftarrow{R} \mathcal{G}; a \xleftarrow{R} \mathbb{Z}_p, b' \xleftarrow{R} \{0,1\}; t \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{hcb}}}$$

, which, by hardness of CDH and definition of hardcore bits should only occur with negligible advantage.

$$\mathcal{B}\left(g, g_1 = g^a, g_2, b', t\right)$$

- Generate $s \xleftarrow{R} \{0,1\}^\ell$
- For $(i,b) \in [n] \times \{0,1\} \backslash i^*, s^*$, set $a_{i,b}$ to be a uniformly random group elements of $\mathbb{Z}_p$
- For $(i,b) \in [n] \times \{0,1\} \backslash i^*, s^*$, set $g_{i,b} = g^{a_{i,b}}$ to be a uniformly random group element of $\mathcal{G}$
- For $(i,j,b) \in [n] \times [\ell] \times [b] \backslash \{(i^*, j^*, \overline{s}_{i^*})\}$, set $\rho_{i,j,b}$ to be a uniformly random element of $\mathbb{Z}_p$
- Set $g_{i^*,s_{i^*}} = g_1 / \prod_{k \neq i^*} g_{k,s_k}$
- For $(i,j,b) \in [n] \times [\ell] \times [b] \backslash \{(i^*, j^*, \overline{s}_{i^*})\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ g_{k,\beta}^{a_{i,j,b}} \text{ otherwise} \end{cases}$$

  Set $t_{i,j,b} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{hcb}}}$.

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i^*,j^*,\overline{s}_{i^*}}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i^*, s_{i^*}) \\ g_2^{a_{k,\beta}} \text{ otherwise} \end{cases}$$

  Set $t_{i^*,j^*,\overline{s}_{i^*}} = t$
- $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}, t_{i,j,b}\}_{i \in [n], j \in [m], b \in \{0,1\}}$
- Generate $r_{i,s_i} \leftarrow \mathsf{LossyHint.Eval}(\mathsf{pp}, s, i)$
- Generate $r_{i,\overline{s}_i}$ for $i \in [n]$
  - For $j \in [\ell]$:
  - If $i \cdot \ell + j - \ell \leq k$
    - Set $z_j = \mathsf{hcb}(\prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,\overline{s}_i}}, t_{i,j,\overline{s}_i})$
  - Else If $i \cdot \ell + j - \ell = k$
    - Set $z_j = b'$
  - Else
    - Set $z_j \xleftarrow{R} \{0,1\}$
  - Set $r_{i,\overline{s}_i} = z_1 z_2 \ldots z_\ell$
- Run $\mathcal{A}(\mathsf{pp}, s, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}})$

Observe that the above reduction computes public parameters $\mathsf{LossyHint.pp}$ consistent with setting $\rho_{i^*,j^*,\bar{s}_{i^*}} = a$ and $g_{i^*,s_{i^*}} = g_1 / \prod_{k \neq i^*} g_{k,s_k}$. Since g_1 is a uniformly random group element independent of g, $g_{i^*,s_{i^*}}$ is also a uniformly random element of $\mathcal{G}$ independent of other $g_{i,b}$. Now note that $\prod_{k=1}^{n} g_{k,s_k}^{\rho_{i^*,j^*,\bar{s}_{i^*}}} = g_2^a$, so $z_{j^*} = \mathsf{hcb}(\prod_{k=1}^{n} g_{k,s_k}^{\rho_{i^*,j^*,\bar{s}_{i^*}}}, t)$ when g, g_1, g_2, b' is drawn from $\mathcal{D}_1$, which is exactly the output of Experiment 2^k. Conversely, when b' is uniformly random from $\mathcal{D}_2$, so is z_{j^*}, which is the output of Experiment 2^{k-1}.

□

Lemma D.16. For any adversary $\mathcal{A}$ $|\mathcal{P}_{\mathcal{A}}^{1^{n \cdot m}} - \mathcal{P}_{\mathcal{A}}^{2}| = 0$.

Proof. We can observe that since $h = \prod_{k=1}^{n} g_{k,s_k}$,

$$h^{\rho_{i,j,b}} = \left(\prod_{k=1}^{n} g_{k,s_k} \right)^{\rho_{i,j,b}} = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,b}}$$

so the z values computed are equal. □

D.4 Lossy Hinting Functions from Learning with Errors

$\mathsf{LossyHint.Setup}(1^\kappa, 1^\ell, 1^n; \mathsf{r}_{\mathsf{Setup}})$

- Let $\mathsf{LWE}_{d,p,\chi}$ be the parameters for our LWE assumption where dimension is d, modulus is p and noise distribution is χ
- For $i \in [n], b \in \{0,1\}$, generate $a_{i,b} \xleftarrow{R} \mathbb{Z}_p^d$
- For $i \in [n], j \in [\ell], b \in \{0,1\}$, generate $v_{i,j,b} \xleftarrow{R} \mathbb{Z}_p^d$
- For $i, k \in [n], j \in [\ell], b, \beta \in \{0,1\}$, generate $e_{i,j,b}^{k,\beta} \xleftarrow{R} \chi$
- For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \bar{b}) \\ a_{k,\beta}^{\mathsf{T}} \cdot v_{i,j,b} + e_{i,j,b}^{k,\beta} \text{ otherwise} \end{cases}$$

- Output $\{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [\ell], b \in \{0,1\}}$

$\mathsf{LossyHint.Eval}\left(\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [\ell], b \in \{0,1\}}, s \in \{0,1\}^n, i \in [n]\right)$

- For $j \in [\ell]$, set $z_j = \mathsf{msb}(\sum_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k])$
- Output $z_1, z_2, \ldots z_\ell$

$\mathsf{LossyHint.Sim}_1(\mathsf{r}_{\mathsf{Setup}}, s)$

- Compute $a = \sum_{i=1}^{n} a_{i,s_i}$
- Output $\mathsf{hint} = a$

$\mathsf{LossyHint.Sim}_2(\mathsf{r}_{\mathsf{Setup}}, \mathsf{hint} = a, i \in [n], b \in \{0,1\})$

- For $j \in [m]$, set $z_j = \mathsf{msb}(a^T \cdot v_{i,j,b} + e_{i,j,b})$
- Output $z_1, z_2, \ldots z_\ell$

Analysis We will proceed through a sequence of experiments from $\beta = 0$ to $\beta = 1$ in the initial lossy hinting function security game, and show the advantage of any adversary in distinguishing said games must be negligible.

Experiment 0 This is the output given to the adversary in the lossy hinting function security game when $\beta = 0$.

- Run LossyHint.Setup
  - Let $\mathsf{LWE}_{d,p,\chi}$ be the parameters for our LWE assumption where dimension is d, modulus is p and noise distribution is χ
  - For $i \in [n], b \in \{0,1\}$, generate $a_{i,b} \xleftarrow{R} \mathbb{Z}_p^d$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$, generate $v_{i,j,b} \xleftarrow{R} \mathbb{Z}_p^d$
  - For $i, k \in [n], j \in [\ell], b, \beta \in \{0,1\}$, generate $e_{i,j,b}^{k,\beta} \xleftarrow{R} \chi$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \bar{b}) \\ a_{k,\beta}^{\mathsf{T}} \cdot v_{i,j,b} + e_{i,j,b}^{k,\beta} \text{ otherwise} \end{cases}$$

  - $\mathsf{pp} = \{\mathbf{H}_{i,j,b}\}_{i \in [n], j \in [\ell], b \in \{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate r_{i,s_i} for $i \in [n]$
  - For $j \in [\ell]$, set $z_j = \mathsf{msb}(\sum_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k])$
  - Set $r_{i,s_i} \leftarrow z_1, z_2, \ldots z_\ell$
- Generate $r_{i,\bar{s}_i}$ for $i \in [n]$
  - Set $r_i, \bar{s}_i \xleftarrow{R} \{0,1\}^\ell$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Experiment 1^k for $k \in [n \cdot m]$

- Run LossyHint.Setup
  - Let $\mathsf{LWE}_{d,p,\chi}$ be the parameters for our LWE assumption where dimension is d, modulus is p and noise distribution is χ
  - For $i \in [n], b \in \{0,1\}$, generate $a_{i,b} \xleftarrow{R} \mathbb{Z}_p^d$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$, generate $v_{i,j,b} \xleftarrow{R} \mathbb{Z}_p^d$
  - For $i, k \in [n], j \in [\ell], b, \beta \in \{0,1\}$, generate $e_{i,j,b}^{k,\beta} \xleftarrow{R} \chi$

- For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i,\bar{b}) \\ a_{k,\beta}^{\mathsf{T}} \cdot v_{i,j,b} + e_{i,j,b}^{k,\beta} \text{ otherwise} \end{cases}$$

- $\mathsf{pp} = \{\mathbf{H}_{i,j,b}\}_{i\in[n],j\in[\ell],b\in\{0,1\}}$

- Generate $s \xleftarrow{R} \{0,1\}^n$
- Generate r_{i,s_i} for $i \in [n]$
  - For $j \in [\ell]$, set $z_j = \mathsf{msb}(\sum_{k=1}^{n} \mathbf{H}_{i,j,s_i}[s_k, k])$
  - Set $r_{i,s_i} \leftarrow z_1, z_2, \ldots z_\ell$
- Generate $r_{i,\bar{s}_i}$ for $i \in [n]$
  - For $j \in [\ell]$:
  - If $i \cdot \ell + j - \ell \leq k$
    - Set $z_j = \mathsf{msb}(\sum_{k=1}^{n} a_{k,s_k}^{\mathsf{T}} \cdot v_{i,j,\bar{s}_i} + e_{i,j,\bar{s}_i}^{k,s_k})$
  - Else
    - Set $z_j \xleftarrow{R} \{0,1\}$
  - Set $r_{i,\bar{s}_i} \leftarrow z_1, z_2, \ldots, z_\ell$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}$

Experiment 2 In this experiment, all the $r_{i,b}$ values are computed using hint rather than multiplying the underlying $g_{i,j,b}$ group elements. This is exactly the output computed by $\mathsf{LossyHint.Sim}_1, \mathsf{LossyHint.Sim}_2$.

- Run LossyHint.Setup
  - Let $\mathsf{LWE}_{d,p,\chi}$ be the parameters for our LWE assumption where dimension is d, modulus is p and noise distribution is χ
  - For $i \in [n], b \in \{0,1\}$, generate $a_{i,b} \xleftarrow{R} \mathbb{Z}_p^d$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$, generate $v_{i,j,b} \xleftarrow{R} \mathbb{Z}_p^d$
  - For $i, k \in [n], j \in [\ell], b, \beta \in \{0,1\}$, generate $e_{i,j,b}^{k,\beta} \xleftarrow{R} \chi$
  - For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i,\bar{b}) \\ a_{k,\beta}^{\mathsf{T}} \cdot v_{i,j,b} + e_{i,j,b}^{k,\beta} \text{ otherwise} \end{cases}$$

  - $\mathsf{pp} = \{\mathbf{H}_{i,j,b}\}_{i\in[n],j\in[\ell],b\in\{0,1\}}$
- Generate $s \xleftarrow{R} \{0,1\}^n$

- Generate hint $= a = \sum_{i=1}^{n} a_{i,s_i}$
- Generate r_{i,s_i} for $i \in [n], b \in \{0,1\}$
  - For $j \in [\ell]$, set $z_j = \mathsf{msb}(a^{\mathsf{T}} \cdot v_{i,j,b} + e_{i,j,b})$
  - Set $r_{i,s_i} \leftarrow z_1, z_2, \ldots, z_\ell$
- Output $\mathsf{pp}, s, \{r_{i,b}\}_{i \in [n], b \in \{0,1\}}$

Lemma D.17. For any adversary $\mathcal{A}$ $|\mathcal{P}^0_{\mathcal{A}} - \mathcal{P}^{1^0}_{\mathcal{A}}| = 0$.

Proof. In Experiment 1^k, we deviate from Experiment 0 for values $i \in [n]$ and $j \in [\ell]$, where $i \cdot \ell + j - \ell \leq k$. Since $k = 0$ and $i, j \geq 1$, these experiments are identical. □

Lemma D.18. Assuming $\mathsf{LWE}_{d,p,\chi}$ is a secure assumption, for any $\mathsf{poly}(2^{\kappa^\delta})$ adversary $\mathcal{A}$, for any $k \in [n \cdot m]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathcal{P}^{1^{k-1}}_{\mathcal{A}} - \mathcal{P}^{1^k}_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$.

Proof. Observe these experiments only differ on $r_{i,j,\overline{s}_i}$ such that $i \cdot \ell + j - \ell = k$. Since $j \in [\ell]$, we can see there exists a unique i^*, j^* for this. Suppose there exists some $\mathcal{A}$ which distinguish between Experiments 1^{k-1} and 1^k with advantage ϵ. Then consider the following algorithm $\mathcal{B}$ which runs in $\mathsf{poly}(\mathcal{A})$ time and breaks LWE with advantage at least $\epsilon - \mathsf{negl}(\kappa)$.

$$\mathcal{B}\left(1^\kappa\right)$$

- Generate $s \xleftarrow{R} \{0,1\}^\ell$
- Query the challenger for $2n-1$ LWE queries and a single challenge $\{a_{k,\beta}, h_{k,\beta}\}_{k \in [n], \beta \in \{0,1\}}$ (where $a_{i^*,\overline{s}_{i^*}}, h_{i^*,\overline{s}_{i^*}}$ is the challenge)
- For $i \in [n], j \in [\ell], b \in \{0,1\}$, generate $v_{i,j,b} \xleftarrow{R} \mathbb{Z}_p^d$
- For $i, k \in [n], j \in [\ell], b, \beta \in \{0,1\}$, generate $e^{k,\beta}_{i,j,b} \xleftarrow{R} \chi$
- For $i \in [n], j \in [\ell], b \in \{0,1\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ a^{\mathsf{T}}_{k,\beta} \cdot v_{i,j,b} + e^{k,\beta}_{i,j,b} \text{ otherwise} \end{cases}$$

- For $(i,j,b) \in [n] \times [\ell] \times \{0,1\} \backslash \{(i^*, j^*, \overline{s}_{i^*})\}$, generate $v_{i,j,b} \xleftarrow{R} \mathbb{Z}_p^d$
- For $(i,j,b) \in [n] \times [\ell] \times \{0,1\} \backslash \{(i^*, j^*, \overline{s}_{i^*})\}$, generate $e^{k,\beta}_{i,j,b} \xleftarrow{R} \chi$ for all $k, \beta \in [n] \times \{0,1\}$
- Set $g_{i^*,s_{i^*}} = g_1 / \prod_{k \neq i^*} g_{k,s_k}$
- For $(i,j,b) \in [n] \times [\ell] \times [b] \backslash \{(i^*, j^*, \overline{s}_{i^*})\}$,

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i,j,b}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i, \overline{b}) \\ a^{\mathsf{T}}_{k,\beta} \cdot v_{i,j,b} + e^{k,\beta}_{i,j,b} \text{ otherwise} \end{cases}$$

$$\forall \beta \in \{0,1\}, k \in n,\ \mathbf{H}_{i^*,j^*,\overline{s}_{i^*}}[\beta, k] = \begin{cases} \perp \text{ if } (k,\beta) = (i^*, s_{i^*}) \\ h_{k,\beta} \text{ otherwise} \end{cases}$$

- $\mathsf{pp} = \mathcal{G}, \{\mathbf{H}_{i,j,b}\}_{i\in[n],j\in[m],b\in\{0,1\}}$
- Generate $r_{i,s_i} \leftarrow \mathsf{LossyHint.Eval}(\mathsf{pp}, s, i)$
- Generate $r_{i,\overline{s}_i}$ for $i \in [n]$
  - For $j \in [\ell]$:
  - If $i \cdot \ell + j - \ell \leq k$
    - Set $z_j = \mathsf{msb}(\sum_{k=1}^{n} a_{k,s_k}^{\mathsf{T}} \cdot v_{i,j,\overline{s}_i} + e_{i,j,\overline{s}_i}^{k,s_k})$
  - Else If $i, j = i^*, j^*$
    - Set $z_j = \mathsf{msb}(\sum_{k\neq i} H_{i,j,\overline{s}_i}[s_k, k] + h_{i,\overline{s}_i})$
  - Else
    - Set $z_j \xleftarrow{R} \{0,1\}$
  - Set $r_{i,\overline{s}_i} \leftarrow z_1, z_2, \ldots, z_\ell$
- Run $\mathcal{A}(\mathsf{pp}, s, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}})$

Note that when $(a_{i^*,\overline{s}_{i^*}}, h_{i^*,\overline{s}_{i^*}})$ is a true LWE sample, then $\sum_{k\neq i} H_{i^*,j^*,\overline{s}_{i^*}}[s_k, k] + h_{i^*,\overline{s}_{i^*}} = \sum_{k\neq i} H_{i^*,j^*,\overline{s}_{i^*}}[s_k, k] + a_{k,s_k}^{\mathsf{T}} \cdot v_{i^*,j^*,\overline{s}_{i^*}} + e$, so $z_{j^*} = \sum_{k=1}^{n} a_{k,s_k}^{\mathsf{T}} \cdot v_{i^*,j^*,\overline{s}_{i^*}} + e_{i^*,j^*,\overline{s}_{i^*}}^{k,s_k}$, which is exactly the output of Experiment 1^k. Conversely, when $h_{i^*,\overline{s}_{i^*}}$ is uniformly random, then so is z_{j^*}, which is the output of Experiment 1^{k-1}.

□

Lemma D.19. For any adversary $\mathcal{A}$ $|\mathsf{adv}_{\mathcal{A}}^{1^{n\cdot m}} - \mathsf{adv}_{\mathcal{A}}^{2}| = 0$.

Proof. We can observe that since $h = \prod_{k=1}^{n} g_{k,s_k}$,

$$h^{\rho_{i,j,b}} = \left(\prod_{k=1}^{n} g_{k,s_k}\right)^{\rho_{i,j,b}} = \prod_{k=1}^{n} g_{k,s_k}^{\rho_{i,j,b}}$$

so the z values computed are equal. □

D.5 Succinct Hinting PRG

We now show that the transformation discussed in Subsection 5.2.1 is in fact a succinct hinting PRG with injective extension.

Lemma D.20. $S.\mathsf{HPRG}$ is a 2^{κ^δ}-secure hinting PRG

Proof. Consider the following sequence of experiments

Experiment 0 This is the distribution received by adversary $\mathcal{A}$ generate using S.HPRG scheme above when $\beta = 0$.

- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- $r'_0 = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
- $r_0 = \mathsf{PRG}(r'_0, 1^\ell)$
- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r_{i,s_i} = \mathsf{PRG}(r'_{i,s_i}, 1^\ell)$
  - $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell$
- Output $n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n], b\in\{0,1\}}$

Experiment 1

- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- $r'_0 = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
- $r_0 = \mathsf{PRG}(r'_0, 1^\ell)$
- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r'_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$
  - $r_{i,b} = \mathsf{PRG}(r'_{i,b}, 1^\ell)\ \forall\ b \in \{0,1\}$
- Output $n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n], b\in\{0,1\}}$

Experiment 2

- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- $r'_0 \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$
- $r_0 = \mathsf{PRG}(r'_0, 1^\ell)$
- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r'_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$
  - $r_{i,b} = \mathsf{PRG}(r'_{i,b}, 1^\ell)\ \forall\, b \in \{0,1\}$
- Output $n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n], b\in\{0,1\}}$

Experiment 3

- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- $r_0 \xleftarrow{R} \{0,1\}^\ell$
- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r'_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$
  - $r_{i,b} = \mathsf{PRG}(r'_{i,b}, 1^\ell)\ \forall\, b \in \{0,1\}$
- Output $n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n], b\in\{0,1\}}$

Experiment 4 This is the distribution received by adversary $\mathcal{A}$ generate using $S.\mathsf{HPRG}$ scheme above when $\beta = 1$.

- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- $r_0 \xleftarrow{R} \{0,1\}^\ell$

- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r_{i,s_i} = \mathsf{PRG}(r'_{i,s_i}, 1^\ell)$
  - $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell$
- Output $n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}$

Claim D.2. Experiments 0 and 1 are computationally indistinguishable to all time $\mathsf{poly}(2^{\kappa^\delta})$ algorithms.

Proof. To see this, consider a sequence of n sub-experiments $0.j$. In sub-experiment $0.j$, we compute $r_{i,\overline{s_i}}$ as per experiment 1 for all $i \leq j$ - i.e. as $\mathsf{PRG}(r'_{i,\overline{s_i}}, 1^\ell) : r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$ and compute it as experiment 0 for all $i > j$ - i.e. as $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell$. Suppose we had a distinguisher $\mathcal{D}$ between sub-experiments $0.j-1$ and $0.j$. Then consider the following adversary $\mathcal{A}$ against the security of PRG

Reduction $\mathcal{A}(1^{\kappa^{\frac{\delta}{\epsilon}}})$:

- Receive PRG challenge $r \in \{0,1\}^\ell$
- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- $r'_0 = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
- $r_0 = \mathsf{PRG}(r'_0, 1^\ell)$
- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r_{i,s_i} = \mathsf{PRG}(r'_{i,s_i}, 1^\ell)$
    - If $i < j$, $r'_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}; r_{i,\overline{s_i}} = \mathsf{PRG}(r'_{i,\overline{s_i}}, 1^\ell)$
    - If $i = j$, $r_{i,\overline{s_i}} = r$
    - If $i > j$, $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell$
- Run $\mathcal{D}\left(n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}\right)$ and forward output.

Observe that when r is a PRG output, this is exactly experiment $0.j$, and when r is truly random, this is exactly experiment $0.j-1$. Since there are only a polynomial number of sub-experiments, we can conclude that sub-experiments 0.0 and $0.n$ are indistinguishable as well, which correspond exactly to experiments 0 and 1. Note that since $\mathcal{A}$ is run on security parameter $\kappa' = \kappa^{\frac{\delta}{\epsilon}}$, by $2^{\kappa'^\epsilon} = 2^{\kappa^\delta}$ subexponential security of PRG, $\mathcal{D}$ and $\mathcal{A}$ can run in $\mathsf{poly}(2^{\kappa^\delta})$ time.

□

Claim D.3. Experiments 1 and 2 are computationally indistinguishable to all time $\mathsf{poly}(2^{\kappa\delta})$ algorithms.

Proof. Here, we can rely on the security of the underlying hinting PRG HPRG. Let $\mathcal{D}$ be a distinguisher between Experiments 1 and 2. Then consider the following distinguisher $\mathcal{A}$ against the HPRG security

Reduction $\mathcal{A}\left(\mathsf{pp}, r_0'^{\beta}, r'_{\mathsf{ext}}, \left\{r'_{i,b}\right\}_{i\in[n],b\in\{0,1\}}\right)$:

- $r_0 = \mathsf{PRG}(r_0'^{\beta}, 1^{\ell})$
- $r_{\mathsf{ext}} = r'_{\mathsf{ext}}$
- $\forall i \in [n]$
  - $r_{i,b} = \mathsf{PRG}(r'_{i,b}, 1^{\ell})\ \forall\ b \in \{0,1\}$
- Run $\mathcal{D}\left(n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}\right)$ and forward output.

Observe that when $\beta = 0$, $r_0'^{\beta}$ comes from $\mathsf{HPRG.Eval}$ making the distribution received by $\mathcal{D}$ identical to Experiment 1, while when $\beta = 1$, it is truly random string, making the distribution received by $\mathcal{D}$ to be experiment 2.

□

Claim D.4. Experiments 2 and 3 are computationally indistinguishable to all time $\mathsf{poly}(2^{\kappa\delta})$ algorithms.

Proof. Let $\mathcal{D}$ be a distinguisher between Experiments 2 and 3. Then consider the following distinguisher $\mathcal{A}$ against the PRG security

Reduction $\mathcal{A}(1^{\kappa^{\frac{\delta}{\epsilon}}})$:

- Receive PRG challenge $r \in \{0,1\}^{\ell}$
- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^{\kappa}, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- Set $r_0 = r$
- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r'_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$
  - $r_{i,b} = \mathsf{PRG}(r'_{i,b}, 1^{\ell})\ \forall\ b \in \{0,1\}$
- Run $\mathcal{D}\left(n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n],b\in\{0,1\}}\right)$ and forward output.

Observe that when r is a PRG output, this is exactly experiment 2, and when r is truly random, this is exactly experiment 3. Similar to the distinguisher between experiments 0 and 1, since PRG is run on a seed of size $\kappa^{\frac{\delta}{\epsilon}}$, this is secure against $\mathsf{poly}(2^{\kappa^\delta})$ time adversaries.

□

Claim D.5. Experiments 3 and 4 are computationally indistinguishable to all time $\mathsf{poly}(2^{\kappa\delta})$ algorithms.

Proof. Much like the argument from experiments 0 to 1, consider a sequence of n sub-experiments $3.j$. In sub-experiment $3.j$, we compute $r_{i,\overline{s_i}}$ as per experiment 3 for all $i > j$ - i.e. as $\mathsf{PRG}(r'_{i,\overline{s_i}}, 1^\ell)$: $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$ and compute it as experiment 4 for all $i \leq j$ - i.e. as $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell$. Suppose we had a distinguisher $\mathcal{D}$ between sub-experiments $3.j-1$ and $3.j$. Then consider the following adversary $\mathcal{A}$ against the security of PRG

Reduction $\mathcal{A}(1^{\kappa^{\frac{\delta}{\epsilon}}})$:

- Receive PRG challenge $r \in \{0,1\}^\ell$
- $(\mathsf{pp}, n) = \mathsf{HPRG.Setup}(1^\kappa, 1^{\kappa^{\frac{\delta}{\epsilon}}})$
- $s \xleftarrow{R} \{0,1\}^n$
- $r_0 \xleftarrow{R} \{0,1\}^\ell$
- $r_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{pp}, s)$
- $\forall i \in [n]$
  - $r'_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{pp}, s, i)$
  - $r_{i,s_i} = \mathsf{PRG}(r'_{i,s_i}, 1^\ell)$
    - If $i < j$, $r_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^\ell$
    - If $i = j$, $r_{i,\overline{s_i}} = r$
    - If $i > j$, $r'_{i,\overline{s_i}} \xleftarrow{R} \{0,1\}^{\kappa^{\frac{\delta}{\epsilon}}}$; $r_{i,\overline{s_i}} = \mathsf{PRG}(r'_{i,\overline{s_i}}, 1^\ell)$
- Run $\mathcal{D}\left(n, ((\mathsf{pp}, \ell), n)), r_0, r_{\mathsf{ext}}, \{r_{i,b}\}_{i\in[n], b\in\{0,1\}}\right)$ and forward output.

Observe that when r is truly random, this is exactly experiment $3.j$, and when r is a PRG output, this is exactly experiment $3.j-1$. Since there are only a polynomial number of sub-experiments, we can conclude that sub-experiments 3.0 and $3.n$ are indistinguishable as well, which correspond exactly to experiments 3 and 4 respectively. Note that since $\mathcal{A}$ is run on security parameter $\kappa' = \kappa^{\frac{\delta}{\epsilon}}$, by $2^{\kappa'^\epsilon} = 2^{\kappa^\delta}$ subexponential security of PRG, $\mathcal{D}$ and $\mathcal{A}$ can run in $\mathsf{poly}(2^{\kappa^\delta})$ time.

□

□

Lemma D.21. S.HPRG is succinct

Proof. Observe the injective extension and seed length of of $S.\mathsf{HPRG}$ are simply that of HPRG. Since HPRG is invoked on parameters independent of ℓ, these lengths must be independent of ℓ as well. The public parameters are likewise that of HPRG, with ℓ in binary. Since ℓ is polynomially bounded in κ, this can be written in κ bits. □

E Removing the Same Tag Restriction

We discuss how to remove the "same-tag" restriction from a commitment scheme. We transform a non-uniform sub-exponentially CCA secure for $N' = 2N$ tags that only allows the adversary to query on one tag Definition 3.6 to a CCA secure scheme on N tags where the adversary does not have this restriction. We will perform the transformation using non uniform subexponentially secure primitives $\mathsf{AuxEquiv}$ (Section 4), extended hinting PRG (Section 5). The transformation runs polynomial in N and the runtime of the primitives involved, thus N should always stay polynomial in the security parameter. Looking ahead, when we combine this transformation with the amplification transformation in Section 7, we want to remove the restriction first before ending up with a scheme exponential in the number of tags.

The techniques in this section overlap substantially with Section 6 and we include the formal details for completeness. The main difference in the construction is that we commit to a tag $\mathsf{tag} \in [N]$ by including a commitment where $x \in [N] \setminus \{\mathsf{tag}\}$ to all tags $(x, 0)$ and $(x, 1)$. The idea is that we can answer all the queries made by the adversary by opening under the challenge tag tag^* (since the adversary is not allowed to query on tag^*, all other queries will have commitments to tag^*).

Let the hinting PRG scheme $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{ExtEval}, \mathsf{CheckParams})$ be a succinct $T = 2^{\kappa^\gamma}$ secure for some constant $\gamma \in (0, 1)$. Let $\mathsf{AuxEquiv}$ be $T = 2^{\kappa^\delta}$-binding secure and statistically hiding where $\delta \in (0, 1)$. and Let $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ be a 2^{κ^c}-subexponentially secure, weak binding, 2^{κ^v}-efficient CCA commitment scheme for $N'(\kappa) = N' = 4N$ tags where $c < 1$ and $v \geq 1$ for message length $u(\kappa)$[12]. We will assume tags take identities of the form $(i, \beta, \Gamma) \in [N] \times \{0, 1\} \times \{0, 1\}$ and that the $\mathsf{Small.Com}$ algorithm take in random coins of length $\ell(\kappa)$.

Let m be the message input to the commitment algoritm and length be denoted by $|m|$. Let $n' = n'(\kappa)$ be the length of the seed plus public parameters plus injective extension of the hinting PRGscheme when invoked on security parameter $\kappa^{\frac{v}{\delta\gamma}}$. Since the scheme is succinct, n' is a function of only κ'' (and hence κ) and not the block length, which we will specify later. By Lemma 4.2, we will set $\mathsf{AuxEquiv}$ to be $(2^{\kappa^\delta}, \frac{|\mathsf{advice}|}{2 \cdot n'})$-binding secure, and let $|y|$ refers to the length of the decommitment strings of said scheme. Finally, we run $\mathsf{Small.Com}$ on messages of size $|y|$, and let ℓ be the size of randomness used by $\mathsf{Small.Com}$ on said input size. We set the block size of our hinting PRGscheme to be the maximum of $|m|, N \cdot \ell$. For ease of notation we assume that $\mathsf{HPRG.Eval}(\mathsf{pp}, s, 0) \in \{0, 1\}^{|m|}$ and $\forall i \in [n]$, $\mathsf{HPRG.Eval}(\mathsf{pp}, s, i) \in \{0, 1\}^{\ell \cdot N}$, i.e. we ignore any extra bits output by the $\mathsf{HPRG.Eval}$ algorithm.

Our transformation will produce three algorithms, $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ which we prove non-uniform 2^{κ^c}-subexponentially secure and $2^{\kappa^{v'}}$-efficient where $v' = \frac{v \cdot \tilde{v}}{\delta \cdot \gamma}$. The construction will call $\mathsf{AuxEquiv}$ on security parameter $\kappa' = \kappa^{\frac{v}{\delta}}$, HPRG on security parameter $\kappa'' = \kappa^{\frac{v}{\delta \cdot \gamma}}$

[12] Recall from Definition 3.2 that a 2^{κ^v}-efficient scheme with $v \geq 1$ implies that the runtime of $\mathsf{Small.Val}$ is polynomial in 2^{κ^v}.

and Same on security parameter κ. For simplicity, we assume that the message space of Same, $u(\kappa)$ is equal to the length of the decommitment string of the equivocal commitment called on κ'.

The proof proceeds similarly to Section 6 where we first switch the way we answer the CCA.Val oracle from brute force algorithm to CCA.ValAlt(tag*, com, $\mathcal{L}$) $\to m \cup \perp$. CCA.ValAlt has the property that it depends only on one tag, tag* (the exception being the check at the start which checks if tag* = com.tag and outputs $\perp$). The rest of the proof follows a similar sequence of steps as Section 6, where we introduce equivocations and reduce information about the seed. Utilizing the security of the hinting PRG with injective extension, we are able to remove the information in the block containing the message and thus hide the message.

CCA.FindSeed(aux)

Inputs: String aux = (HPRG.pp, aux$'$)
Output: $\tilde{s} \in \{0,1\}^n \cup \perp$

- Parse aux as (HPRG.pp, aux$'$)
- Iterate through all $\tilde{s} \in \{0,1\}^n$
  - If aux$'$ = HPRG.ExtEval(HPRG.pp, $\tilde{s}$), return s.
- Return $\perp$

Figure 8: Routine CCA.FindSeed

CCA.Check($\tilde{s}$, com)

Inputs: Seed candidate $\tilde{s} = \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$
Commitment com = $\left(\text{tag}, \text{aux}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N] \setminus \{\text{tag}\}})_{i \in [n]}\right)$
Output: $\{0,1\}$

- For $i \in [n]$
  1. Compute $(r_{1,i}, r_{2,i}, \ldots, r_{N,i})$ = HPRG.Eval(HPRG.pp, $\tilde{s}, i$)
  2. For $x \in [N] \setminus \{\text{tag}\}$
     (a) Let $\tilde{y}_i$ = Same.Recover($c_{x,i,\tilde{s}_i}, r_{x,i}$). If $\tilde{y}_i = \perp$, output 0
     (b) If $c_{x,i,\tilde{s}_i} \neq$ Same.Com(1^κ, (x, tag$_x$, $\tilde{s}_i$), $\tilde{y}_i$; $r_{x,i}$), output 0.
     (c) If $\tilde{s}_i \neq$ AuxEquiv.Decom(aux, $\sigma_i, \tilde{y}_i$), output 0.
- Parse aux as (HPRG.pp, aux$'$).
- If HPRG.CheckParams(HPRG.pp, n) = 0, output 0.
- If aux$'$ $\neq$ HPRG.ExtEval(HPRG.pp, s) output 0.
- If all the above checks have passed, output 1.

Figure 9: Routine CCA.Check

$\mathsf{CCA.FindAlt}(x', \mathsf{com}, \mathcal{L})$

Inputs: Index $x' \in [N]$

Commitment $\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N] \setminus \{\mathsf{tag}\}})_{i \in [n]}\right)$

Polynomial Size Non-Uniform Advice List $\mathcal{L}$

Output: $\tilde{s} \in \{0,1\}^n$

- If for some $\tilde{s} \in \{0,1\}^n$, $(\tilde{s}, \mathsf{com.aux}) \in \mathcal{L}$, where $\tilde{s}$ is the seed recorded from the advice. Output $\tilde{s}$.
- Else if $\mathsf{com.aux}$ is not recorded in $\mathcal{L}$,
  - For each $i \in [n]$
    1. Let $\tilde{y}_i = \mathsf{Same.Val}(c_{x',i,0})$
    2. Set $\tilde{z}_i = \mathsf{AuxEquiv.Decom}(\mathsf{aux}, \sigma_i, \tilde{y}_i)$. If $\tilde{z}_i = \perp$, set $\tilde{s}_i = 1$. Else, set $\tilde{s}_i = \tilde{z}_i$.
  - Output $\tilde{s} = \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$.

Figure 10: Routine CCA.FindAlt

$\mathsf{CCA.Equiv}(\mathsf{com})$

Inputs: Commitment $\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux} = (\mathsf{HPRG.pp}, \mathsf{aux}'), c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N] \setminus \{\mathsf{tag}\}})_{i \in [n]}\right)$

Output: Equivocation $(\mathsf{aux}, c, d_0, d_1) \cup \perp$

- For all $i \in [n], x \in [N] \setminus \{\mathsf{tag}\}, b \in \{0,1\}$,
  - If, $\mathsf{AuxEquiv.Decom}(\mathsf{Small.Val}(\mathsf{com}.c_{x,i,b})) = 0$, and $\mathsf{AuxEquiv.Decom}(\mathsf{Small.Val}(\mathsf{com}.c_{x,i,\bar{b}})) = 1$.
    Return $(\mathsf{aux}, \sigma_i, \mathsf{Small.Val}(\mathsf{com}.c_{x,i,b}), \mathsf{Small.Val}(\mathsf{com}.c_{x,i,\bar{b}}))$
- Return $\perp$

Figure 11: Routine CCA.Equiv

Transformation $\mathsf{OneToMany}(\mathsf{Same} = (\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover}), \mathsf{HPRG}, \mathsf{AuxEquiv}, w(\kappa), v') \to \mathsf{NM} = (\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$:

$\mathsf{CCA.Com}(1^\kappa, \mathsf{tag} \in [N], m \in \{0,1\}^{w(\kappa)}; r) \to \mathsf{com}$

1. Compute $\kappa' = \kappa^{\frac{v}{\delta}}$. Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
2. Sample $(\mathsf{HPRG.pp}, n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(|m|, N \cdot \ell)})$.
3. Sample $s = s_1 \ldots s_n \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
4. Set $\mathsf{aux} = (\mathsf{HPRG.pp}, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s))$.
5. For all $i \in [n]$ run $\mathsf{AuxEquiv.Com}(1^{\kappa'}, \mathsf{aux}, s_i) \to (\sigma_i, y_i)$.

6. Let for $x \in [N], i \in [n]$, $r_{x,i}, \tilde{r}_{x,i} \in \{0,1\}^\ell$ be defined as follows[13]:
7. For $i \in [n]$
   (a) Compute $(r_{1,i}, r_{2,i}, \ldots, r_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i)$
   (b) Sample $(\tilde{r}_{1,i}, \tilde{r}_{2,i}, \ldots, \tilde{r}_{N,i}) \xleftarrow{R} \{0,1\}^{N \cdot \ell}$
8. Compute $c = m \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$
9. For $i \in [n]$, $x \in [N] \setminus \{\mathsf{tag}\}$
   (a) If $s_i = 0$
      i. $c_{x,i,0} = \mathsf{Same.Com}(1^\kappa, (x,0), \mathsf{msg} = y_i; r_{x,i})$
      ii. $c_{x,i,1} = \mathsf{Same.Com}(1^\kappa, (x,1), \mathsf{msg} = y_i; \tilde{r}_{x,i})$
   (b) If $s_i = 1$
      i. $c_{x,i,0} = \mathsf{Same.Com}(1^\kappa, (x,0), \mathsf{msg} = y_i; \tilde{r}_{x,i})$
      ii. $c_{x,i,1} = \mathsf{Same.Com}(1^\kappa, (x,1), \mathsf{msg} = y_i; r_{x,i})$

Output $\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N] \setminus \{\mathsf{tag}\}})_{i \in [n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

$\mathsf{CCA.Val}(\mathsf{com}) \to m \cup \perp$

1. Set $\tilde{s} = \mathsf{CCA.FindSeed}(\mathsf{com.aux})$.
2. If $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 0$ output $\perp$.
3. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, 0)$.

$\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}) \to m \cup \perp$

1. If $\mathsf{com.tag} = \mathsf{tag}^*$, output $\perp$.
2. Set $\tilde{s} = \mathsf{CCA.FindAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$.
3. If $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 0$ output $\perp$.
4. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, 0)$.

$\mathsf{CCA.Recover}(\mathsf{com}, r) \to m \cup \perp$

1. From r, parse seed s of Hinting PRG.
2. If $\mathsf{CCA.Check}(\mathsf{com}, s) = 0$, output $\perp$.
3. From com, parse the commitment component c and the public parameter HPRG.pp.
4. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$

The procedures CCA.Exp (Figure 5), CCA.AdviceList (Figure 6) remain the exact same where the parameter settings and CCA.Equiv is run according to Section E.

Remark E.1. The randomness $r_{\mathsf{tag},i}$ and $\tilde{r}_{\mathsf{tag},i}$ are not used for all $i \in [n]$, but we generate it this way for notational simplicity.

[13] We will only use $(N-1) \cdot n$ random bits, but we generate extra $n \cdot \ell$ bits for simplifying notation.

Correctness and Efficiency.

Efficiency

Claim E.1. If $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$ is 2^{κ^v}-efficient CCA commitment scheme as per Definition 3.2 with tag space $N(\kappa) \in \mathsf{poly}(\kappa)$, $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is an efficient equivocal commitment scheme as per Definition 4.3, v, v' are constants where $v \geq 1$, then $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is $2^{\kappa^{v'}}$-efficient CCA commitment scheme. Moreover, there exists a *non-uniform* algorithm $\mathsf{CCA.ValAlt}$ that runs in time $\mathsf{poly}(|m|, 2^{\kappa^v})$.

Proof.

- $\mathsf{CCA.Com}$ calls $\mathsf{Same.Com}$ $2 \cdot n \cdot (N-1)$ times on the output of $\mathsf{AuxEquiv.Com}$ in addition to some other poly-time computation. By Definition 3.2, $\mathsf{Same.Com}$ is $\mathsf{poly}(|m|, \kappa)$. Since $\mathsf{AuxEquiv.Com}$ runs in time $\mathsf{poly}(\kappa')$ by Definition 4.3, this bounds the message of $\mathsf{Small.Com}$ with $\mathsf{poly}(\kappa)$. Along with the fact that n is bounded by the security parameter, and N is bounded by the tag space which we assume is $\mathsf{poly}(\kappa)$, this is overall polynomial bounded in κ.
- $\mathsf{CCA.Recover}$ does a single $\oplus$, and since com and r are both bounded by $\mathsf{poly}(\kappa)$ by the runtime of $\mathsf{CCA.Com}$, $\mathsf{CCA.Recover}$ runs in $\mathsf{poly}(|m|, \kappa)$ as well.
- $\mathsf{CCA.Val}$ checks every possible seed in $\{0,1\}^n$, i.e. runs in time 2^n where n is in $\mathsf{poly}(\kappa'') = \Theta(\kappa''^{\tilde{v}}) = \Theta(\kappa^{v'})$. Thus the oracle is $2^{\kappa^{v'}}$-efficient from Definition 3.2.
- Additionally, $\mathsf{CCA.ValAlt}$ non-uniformly calls $\mathsf{CCA.FindAlt}$ that performs a check on a polynomial size list $\mathcal{L}$ and calls $\mathsf{Same.Val}$, $n = \mathsf{poly}(\kappa)$ times. As $\mathsf{Same.Val}$ runs in time 2^{κ^v}, thus the runtime is $\mathsf{poly}(|m|, 2^{\kappa^v})$ where $v \geq 1$.

□

Correctness.

Claim E.2. If $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$ is a correct CCA commitment scheme as per Definition 3.1 and $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom})$ is a correct equivocal commitment scheme as per Definition 4.2, hinting PRG satisfies the properties Definition 5.3, Definition 5.2 then $(\mathsf{CCA.Com}, \mathsf{CCA.Decom}, \mathsf{CCA.Val})$ is a correct CCA commitment scheme.

Proof. By the fact that the hinting PRG sets up parameters that are injective from Definition 5.3, Definition 5.2 implies that the same seed is output by $\mathsf{CCA.FindSeed}$ as was constructed by $\mathsf{CCA.Com}$.

Note that if base scheme is correct, then $\forall i \in [n], x \in [N] \setminus \{\mathsf{tag}\}, b \in \{0,1\}$,

$$\mathsf{Same.Val}(\mathsf{Same.Com}(1^\kappa, (x,b), y_i; r)) = y_i.$$

Also from the correctness of equivocal scheme, $\forall i \in [n]$,

$$\mathsf{AuxEquiv.Decom}(\mathsf{aux}, \mathsf{AuxEquiv.Com}(1^{\kappa'}, \mathsf{aux}, s_i)) = s_.,$$

On input s, $\forall i \in [n], x \in [N] \setminus \{\mathsf{tag}\}$, correctly sets the randomness along c_{x,i,s_i} and $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) = m$. We can observe that the scheme is correct. □

Below is an additional claim on the decryption oracle query of CCA.ValAlt.

Claim E.3. CCA.ValAlt alternate decryption oracle only queries Same.Val on the tag $(\mathsf{tag}^*, 0)$.

Proof. We can see CCA.ValAlt only ever calls Same.Val on commitments with tag $(\mathsf{tag}^*, 0)$. If $\mathsf{com.tag} = \mathsf{tag}^*$, we output $\perp$ and aborts without an oracle call to Same.Val. Otherwise, as com contains commitments under every tag except $(\mathsf{com.tag}, 0)$ and $(\mathsf{com.tag}, 1)$, CCA.ValAlt can query the oracle on tag $(\mathsf{tag}^*, 0)$. □

Recovery from Randomness. It is easy to see that the above scheme also satisfies the recovery from randomness property.

E.1 Proof of Security

We now prove security by showing a sequence of games that prove that we can transform a 2^{κ^c}-subexponentially "same tag" CCA secure 2^{κ^v}-efficient scheme on $2N$ tags to a 2^{κ^c}-subexponentially CCA secure $2^{\kappa^{v'}}$-efficient scheme on N tags.

The proof will follow the same techniques as in Section 6. The notable aspect different from the amplification section is based on Claim E.3, i.e. the security relies on the fact that the requests to the Same.Val oracle are made on the "same tag".

Theorem E.1. Let $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$ be a correct, polynomially efficient, weak binding "same tag" 2^{κ^c}-subexponentially secure (Definition 3.6, 2^{κ^v}-efficient CCA commitment scheme for $N'(\kappa) = N' = 2N \in \mathsf{poly}(\kappa)$ tags and message space $u(\kappa) \in \mathsf{poly}(\kappa)$ where $c < 1$ and $v \geq 1$. Let $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{ExtEval}, \mathsf{CheckParams})$ be $T = 2^{\kappa^\gamma}$ secure hinting PRG with injective extension for some constant $\gamma \in (0, 1)$ and $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ be $T = 2^{\kappa^\delta}$-binding secure and statistically hiding setupless equivocal commitment where $\delta \in (0, 1)$. Then the above commitment scheme $(\mathsf{CCA.Com}, \mathsf{CCA.Val})$ is a correct, polynomially efficient, binding, non-uniform 2^{κ^c}-subexponentially secure for N tags that is $2^{\kappa^{v'}}$-efficient and commits to messages of length $w(\kappa) \in \mathsf{poly}(\kappa)$ where $v' = \frac{v \cdot \tilde{v}}{\delta \cdot \gamma}$.

Proof. Correctness, polynomial efficiency, binding and randomness recovery follow along the lines of Section 6.

We first define our sequence of games. Then for each adjacent set of games we prove that the advantage of any non-uniform attacker $\mathcal{A}$ that runs in time $\mathsf{poly}(2^{\kappa^c})$ must be negligibly close.

Game 0. This is the original message hiding game between a challenger and a non-uniform attacker for 2^{κ^c}-subexponentially secure adversaries. The game is parameterized by a security parameter κ.

1. The attacker sends a "challenge tag" $\mathsf{tag}^* \in [N]$.
2. **Pre Challenge Phase:** The attacker makes repeated queries commitments

$$\mathsf{com} = \left(\mathsf{tag}, \mathsf{aux}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N] \setminus \{\mathsf{tag}\}})_{i \in [n]}\right).$$

   If $\mathsf{tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

$$\mathsf{CCA.Val}(\mathsf{com}).$$

3. **Challenge Phase**

   (a) The attacker sends two messages $m_0^*, m_1^* \in \{0,1\}^w$

   (b) **Part 1:**

   - Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
   - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Sample $(\mathsf{HPRG.pp}^*, n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(|m|, N\cdot\ell)})$.
   - Sample $s^* = s_1^* \dots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
   - Set $\mathsf{aux}^* = (\mathsf{HPRG.pp}^*, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}^*, s^*))$.
   - For all $i \in [n]$ run $\mathsf{AuxEquiv.Com}(1^{\kappa'}, \mathsf{aux}^*, s_i^*) \rightarrow (\sigma_i^*, y_i^*)$.
   - Let $r_{x,i}^*, \tilde{r}_{x,i}^* \in \{0,1\}^\ell$ be defined as follows:
   - For $i \in [n]$
     i. Compute $(r_{1,i}^*, r_{2,i}^*, \dots, r_{N,i}^*) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
     ii. Sample $(\tilde{r}_{1,i}^*, \tilde{r}_{2,i}^*, \dots, \tilde{r}_{N,i}^*) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$

   (c) **Part 2:**

   - It chooses a bit $b \in \{0,1\}$ and sets $c^* = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m_b^*$.
   - For $i \in [n]$, $x \in [N] \setminus \{\mathsf{tag}\}$
     i. If $s_i^* = 0$
       A. $c_{x,i,0}^* = \mathsf{Small.Com}(1^\kappa, (x,0), y_i^*; r_{x,i}^*)$
       B. $c_{x,i,1}^* = \mathsf{Small.Com}(1^\kappa, (x,1), y_i^*; \tilde{r}_{x,i}^*)$
     ii. If $s_i^* = 1$
       A. $c_{x,i,0}^* = \mathsf{Small.Com}(1^\kappa, (x,0), y_i^*; \tilde{r}_{x,i}^*)$
       B. $c_{x,i,1}^* = \mathsf{Small.Com}(1^\kappa, (x,1), y_i^*; r_{x,i}^*)$
   - Output $\mathsf{com} = \left(\mathsf{tag}^*, \mathsf{aux}^*, \mathsf{HPRG.pp}^*, c^*, (\sigma_i^*, \left(c_{x,i,0}^*, c_{x,i,1}^*\right)_{x\in[N]\setminus\{\mathsf{tag}\}})_{i\in[n]}\right)$ as the commitment. All of the randomness is used as the decommitment string.

4. **Post Challenge Phase:** The attacker again makes commitment queries com. If $\mathsf{tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

$$\mathsf{CCA.Val}(\mathsf{com}).$$

5. The attacker finally outputs a guess b'.

Game 1. This is same as Game 0, except that during the **Pre Challenge Phase** and **Post Challenge Phase**, challenger using CCA.ValAlt to answer queries. Let $\mathcal{A}$ be an adversary with non-uniform advice that tries to guess the difference between the two games. The Challenger uses $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$ to return queries where $\mathcal{L}$ is generated through the procedure CCA.AdviceList.

1. **Non-uniform Computation:** The challenger generates the list $\mathcal{L}^{(0)}, \mathcal{L}^{(1)}, \dots, \mathcal{L} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, (\mathcal{A}, \mathsf{advice}))$ by interacting with the attacker $\mathcal{A}$. It uses $\mathcal{L}$ to answer adversaries queries.

2. The attacker sends a "challenge tag" $\mathsf{tag}^* \in [N]$.

3. **Pre Challenge Phase:** The attacker makes repeated queries commitments com. If $\mathsf{com.tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

$$\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L}).$$

4. **Challenge Phase** ....

5. **Post Challenge Phase:** Same as **Pre Challenge Phase**.

6. The attacker finally outputs a guess b'.

Game 2. In this game in **Part 1** the (σ_i^*, y_i^*) are now generated from the AuxEquiv.Equivocate algorithm instead of the Equiv.Com algorithm.

- Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
- Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
- Sample $(\mathsf{HPRG.pp}^*, n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(|m|, N\cdot\ell)})$.
- Sample $s^* = s_1^* \dots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
- Set $\mathsf{aux}^* = (\mathsf{HPRG.pp}^*, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}^*, s^*))$.
- Let $r_{x,i}^*, \tilde{r}_{x,i}^* \in \{0,1\}^\ell$ be defined as follows:
- For $i \in [n]$
  1. Compute $(r_{1,i}^*, r_{2,i}^*, \dots, r_{N,i}^*) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
  2. Sample $(\tilde{r}_{1,i}^*, \tilde{r}_{2,i}^*, \dots, \tilde{r}_{N,i}^*) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$
- For all $i \in [n]$ run $\mathsf{AuxEquiv.Equivocate}(1^{\kappa'}, \mathsf{aux}^*) \to (\sigma_i^*, y_{i,0}^*, y_{i,1}^*)$.
- For all $i \in [n]$, set $y_i^* = y_{i,s_i^*}^*$.

Game 3 In this game in **Part 2** we move to $c_{x,i,0}^*$ committing to $y_{i,0}^*$ and $c_{x,i,1}^*$ committing to $y_{i,1}^*$ for all $x \in [N] \setminus \{\mathsf{tag}^*\}, i \in [n]$ independently of the value of s_i^*.

- It chooses a bit $b \in \{0,1\}$ and sets $c^* = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m_b^*$.
- For $i \in [n]$, $x \in [N] \setminus \{\mathsf{tag}^*\}$
  1. If $s_i^* = 0$
     (a) $c_{x,i,0}^* = \mathsf{Same.Com}(1^\kappa, (x,0), y_{i,0}^*; r_{x,i}^*)$
     (b) $c_{x,i,1}^* = \mathsf{Same.Com}(1^\kappa, (x,1), y_{i,1}^*; \tilde{r}_{x,i}^*)$
  2. If $s_i^* = 1$
     (a) $c_{x,i,0}^* = \mathsf{Same.Com}(1^\kappa, (x,0), y_{i,0}^*; \tilde{r}_{x,i}^*)$
     (b) $c_{x,i,1}^* = \mathsf{Same.Com}(1^\kappa, (x,1), y_{i,1}^*; r_{x,i}^*)$
- Finally, it sends $\mathsf{com}^* = \left(\mathsf{tag}^*, \mathsf{HPRG.pp}^*, c^*, (\sigma_i^*, \left(c_{x,i,0}^*, c_{x,i,1}^*\right)_{x\in[N]\setminus\{\mathsf{tag}^*\}})_{i\in[n]}\right)$ as the commitment. All of the randomness is used as the decommitment string.

Game 4. In this game c^* is chosen uniformly at random (instead of choosing $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m_b^*$).

E.2 Analysis.

Next, we show by a sequence of lemmas that no non-uniform with runtime $\mathsf{poly}(2^{\kappa^c})$ adversary where $c \in (0,1)$ can distinguish between any two adjacent games with non-negligible advantage. In the last game, we show that the advantage of any such adversary is negligible. We will let $\mathsf{adv}_{\mathcal{A}}^x$ denote the quantity $\Pr[b' = b] - \frac{1}{2}$ in Game x.

Lemma E.1. Suppose $(\mathsf{AuxEquiv.Com}, \mathsf{AuxEquiv.Decom}, \mathsf{AuxEquiv.Equivocate})$ is $(2^{\kappa^\delta}, \frac{|\mathsf{advice}|}{2n'})$ binding secure Definition 4.4 auxilary input equivocal commitment scheme. HPRG be an injectively extended hinting PRG and $\mathsf{Small.Val}$ is v-efficient. Then, for an non-uniform adversary $\mathcal{A}$ running in time $\mathsf{poly}(2^{\kappa^c})$ where $c \in (0,1)$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}_{\mathcal{A}}^0 - \mathsf{adv}_{\mathcal{A}}^1| \leq \mathsf{negl}(\kappa)$ where equivocal commitment is run on security parameter $\kappa' = \kappa^{\frac{v}{\delta}}$.

Proof. The proof is identical to Lemma 6.4 and the accompanying lemmas. The only change is the procedure $\mathsf{CCA.Equiv}$, where we check for equivocation for all tags except one. For brevity, we don't repeat the exact details.

□

Lemma E.2. Assuming that the equivocal commitment is statistically equivocal from Definition 4.5. For any adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}_{\mathcal{A}}^1 - \mathsf{adv}_{\mathcal{A}}^2| \leq \mathsf{negl}(\kappa)$ where equivocal commitment is run on security parameter $\kappa' = \kappa^{\frac{v}{\delta}}$.

Proof. From Definition 4.5, we know that the statistical distance between $(\mathsf{aux}^*, \sigma_i^*, y_i^*)$ in Games 1 and 2 is negligible. Since the rest of the inputs to the games are the same, this bounds the statistical distance of the output by a negligible function $\mathsf{negl}(\kappa)$ as well. □

Lemma E.3. Assuming that the base commitment scheme is 2^{κ^c}-subexponentially CCA secure, 2^{κ^v}-efficient CCA commitment. For any non-uniform adversary $\mathcal{A}$ that runs in time $\mathsf{poly}(2^{\kappa^c})$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}_{\mathcal{A}}^2 - \mathsf{adv}_{\mathcal{A}}^3| \leq \mathsf{negl}(\kappa)$.

Proof. Let $\mathcal{A}$ be an adversary given advice that has non-negligible advantage given by the polynomial $p(\cdot)$ in distingushing between the two games, i.e. for infinitely many $\kappa \in \mathbb{N}$,

$$|\mathsf{adv}_{\mathcal{A}}^2 - \mathsf{adv}_{\mathcal{A}}^3| \geq \frac{1}{p(\kappa)}.$$

Define Game 2_0 as Game 2. For all $j \in [N \cdot n]$, we define Game 2_j same as Game 2_{j-1}, with the following additional changes:
We can write $j = (i'-1) \cdot N + (x'-1)$ where $x' \in [N], i' \in [n]$ from Euclidean division, and we change the way $c^*_{x',i',\bar{s_i^*}}$[14] is generated from

$$c^*_{x',i',\bar{s}^*_{i'}} = \mathsf{Small.Com}(1^\kappa, (x', \bar{s}^*_{i'}), y^*_{i',s^*_{i'}}; \tilde{r}^*_{x',i'})$$

[14]There are n games where $x' = \mathsf{tag}^*$ for which $c^*_{x',i',\bar{s}^*_{i'}}$ don't exist. We will treat these as identical to previous games, but leave them in to avoid cluttering notation.

to

$$c^*_{x',i',\bar{s}^*_{i'}} = \mathsf{Small.Com}(1^\kappa, (x', \bar{s}^*_{i'}), y^*_{i',\bar{s}^*_{i'}}; \tilde{r}^*_{x',i'})$$

Observe that Game $2_{N \cdot n}$ is exactly Game 3.
Thus $\exists j = j(\kappa) \in [N \cdot n]$, for infinitely many $\kappa \in \mathbb{N}$,

$$|\mathsf{adv}^{2_{j-1}}_{\mathcal{A}} - \mathsf{adv}^{2_j}_{\mathcal{A}}| \geq \frac{1}{p(\kappa)(N \cdot n)}.$$

We will show a reduction $\mathcal{B}_j$ that achieves a non negligible advantage to the security of Same.Com.

Reduction $\mathcal{B}_j(1^\kappa)$:

Non-Uniform Computation:

- Run $\mathcal{L} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, (\mathcal{A}, \mathsf{advice}))$ non-uniformly.
- Compute equivocations.
  - Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
  - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
  - Let $(\mathsf{HPRG.pp}^*, n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(w, N \cdot \ell)})$.
  - Sample $s^* = s^*_1 \dots s^*_n \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
  - Set $\mathsf{aux}^* = (\mathsf{HPRG.pp}^*, \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}^*, s^*))$.
  - For all $i \in [n]$ run $\mathsf{AuxEquiv.Equivocate}(1^{\kappa'}, \mathsf{aux}^*) \to (\sigma^*_i, y^*_{i,0}, y^*_{i,1})$.
  - Let L be $(s^*, \mathsf{aux}^*, \{(\sigma^*_i, y^*_{i,0}, y^*_{i,1})\}_{i \in n})$.

1. $\mathcal{A}$ sends a challenge $\mathsf{tag}^* \in [N]$ to $\mathcal{B}_j$.
2. Let $c^*_{x',i',\bar{s}^*_{i'}}$ be the commitment changed in Game 2_j as described above.
3. Send challenge tag $(x', \bar{s}^*_{i'})$ to challenger.
4. **Pre Challenge Phase:**
   - For every com query,

     $$\mathsf{com} = \Big(\mathsf{tag}, \mathsf{aux}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N]})_{i \in [n]})\Big)$$

     to $\mathcal{B}_j$.
   - $\mathcal{B}_j$ answers by running $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$. This can be done efficiently using $\mathcal{B}_j$'s own Pre Challenge oracle access to Same.Val, $\mathcal{L}$ generated non-uniformly and runs CCA.FindAlt manually. By Claim E.3, these will all be to the same tag.
5. **Challenge Phase:**
   - $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^w$.
   - Select a random bit β.
6. **Part 1:**

- Recall $\kappa' = \kappa^{\frac{v}{\delta}}$, $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
- Now using L, we retreive the non-uniform equivocations, i.e. parse L as $(s^*, \mathsf{aux}^*, \{\sigma_i^*, y_{i,0}^*, y_{i,1}^*\}_{i\in n})$.
- Let $r_{x,i}, \tilde{r}_{x,i} \in \{0,1\}^\ell$ be defined as follows:
- For $i \in [n]$
  (a) Compute $(r_{1,i}, r_{2,i}, \ldots, r_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s^*, i)$
  (b) Sample $(\tilde{r}_{1,i}, \tilde{r}_{2,i}, \ldots, \tilde{r}_{N,i}) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$

(a) Submit $m_0^* = y_{i,s_{i'}^*}^*, m_1^* = y_{i',\bar{s}_{i'}^*}^*$ to challenger
(b) Receive $\mathsf{com}^* = \mathsf{Same.Com}((x', \bar{s_{i'}^*}), m_b^*; r)$ from challenger
(c) Set $c^*_{x',i',\bar{s_{i'}^*}} = \mathsf{com}^*$
(d) Run **Phase 2** of **Challenge Phase** using the message m_β with the exception that $c^*_{x',i',\bar{s_{i'}^*}}$ is computed as noted above and submit output to $\mathcal{A}$.

7. **Post Challenge Phase:** Proceeds exactly as Pre Challenge Phase.
8. Receive bit guess β' from $\mathcal{A}$.
9. If $\beta = \beta'$, output 0. Otherwise, output 1.

Claim E.4. $\mathcal{B}_j$ is a non-uniform algorithm that outputs polynomial size advice and runs in time $\mathsf{poly}(2^{\kappa^c})$.

Proof. Since the procedure $\mathsf{CCA.AdviceList}$ outputs a polynomial size list $\mathcal{L}$ and since n and output of $\mathsf{AuxEquiv.Equivocate}$ is polynomial in κ, L is polynomial in κ and $\mathcal{B}_j$ gets polynomial size advice.

The runtime of $\mathcal{B}_j$ includes running the algorithm $\mathcal{A}$ that runs in time $\mathsf{poly}(2^{\kappa^c})$ and other algorithms that run in $\mathsf{poly}(\kappa)$. Note that it doesn't need to run $\mathsf{Small.Val}$ as it uses the oracle from the security of the small commitment scheme and it gets equivocations non-uniformly. □

Claim E.5. The advantage of $\mathcal{B}_j$ in winning the "same tag" message hiding game for the base commitment scheme $\mathsf{Same.Com}$ from Definition 3.6 is $\geq \frac{1}{2p(\kappa)(N\cdot n)}$ for infinitely many $\kappa \in \mathbb{N}$.

Proof. First we observe that from Claim E.3, the queries made by $\mathcal{B}_j$ to the challenger preserve the "same tag" condition.

Secondly, observe that if $\beta = 0$, then

$$c^*_{x',i',\bar{s_{i'}^*}} = \mathsf{Same.Com}((x', \bar{s_{i'}^*}), y^*_{i',s_{i'}^*}; r)$$

which is exactly what it is in Game 2_{j-1}, and similarly, if $\beta = 1$

$$c^*_{x',i',\bar{s_{i'}^*}} = \mathsf{Same.Com}((x', \bar{s_{i'}^*}), y^*_{i',\bar{s}_{i'}^*}; r)$$

which is what it is in Game 2_j.

Let q be the probability $\mathcal{A}$ wins Game 2_j and $\mathcal{A}$ wins Game 2_{j-1} with probability $q \pm \frac{1}{p(\kappa)\cdot N\cdot n}$. $\mathcal{B}_j$ wins if $\beta = \beta'$ and $b = 0$ - i.e. $\mathcal{A}$ wins Game 2_{j-1} or if $\beta \neq \beta'$ and $b = 1$ - i.e. $\mathcal{A}$ loses Game 2_j. Thus for infinitely many $\kappa \in \mathbb{N}$, the probability of $\mathcal{B}_j$ winning is given by,

$$\frac{1}{2}\left(q \pm \frac{1}{p(\kappa)\cdot N\cdot n}\right) + \frac{1}{2}(1-q) = \frac{1}{2} \pm \frac{1}{2\cdot p(\kappa)\cdot N\cdot n}.$$

□

As Same.Com is a "same tag" secure scheme, the proof of the lemma follows immediately by contradiction from the above claims.

□

Lemma E.4. Assuming that the hinting PRG is subexponentially secure with $T = 2^{\kappa^\gamma}$ where $\gamma \in (0,1)$ from Definition 5.1. For any non-uniform 2^{κ^c}-subexponentially secure adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, $|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \leq \mathsf{negl}(\kappa)$ where hinting PRG is run on security parameter $\kappa'' = \kappa'^{\frac{1}{\gamma}} = \kappa^{v'} = \kappa^{\frac{v}{\delta \cdot \gamma}}$.

Proof. Let $\mathcal{A}$ be a non-uniform adversary given advice that has non-negligible advantage given by the polynomial $p(\cdot)$ in distingushing between the two games, i.e. for infinitely many $\kappa \in \mathbb{N}$,

$$|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}.$$

We will construct a $\mathsf{poly}(2^{\kappa^c})$ time non-uniform adversary $\mathcal{B}$ which has advantage $\frac{1}{2p(\kappa)}$ in the hinting PRG Game as per Definition 5.1 where inputs were called on security parameter κ''.

Reduction $\mathcal{B}\left(\mathsf{HPRG.pp}, \left(r^\beta_0, r^\beta_{\mathsf{ext}}, \left\{r^\beta_{i,b}\right\}_{i\in[n],b\in\{0,1\}}\right)\right)$:

Non-Uniform Computation:

- Run $\mathcal{L} \leftarrow \mathsf{CCA.AdviceList}(1^\kappa, (\mathcal{A}, \mathsf{advice}))$ non-uniformly.

1. Choose a random bit $a \in \{0,1\}$.
2. Run $\mathcal{A}$
   (a) **Pre Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$ using $\mathcal{L}$.
   (b) $\mathcal{A}$ sends two messages $m_0, m_1 \in \{0,1\}^w$.
   (c) **Challenge Phase:**
   - Compute $\kappa' = \kappa^{\frac{v}{\delta}}$.
   - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Let $r^*_{x,i,b} \in \{0,1\}^\ell$ be defined as follows:
   - For $i \in [n]$, $b \in \{0,1\}$
     i. Split up $(r^*_{1,i,b}, r^*_{2,i,b}, \ldots, r^*_{N,i,b}) = \underline{r^\beta_{i,b}}$
   - For all $i \in [n]$ run $\mathsf{AuxEquiv.Equivocate}(1^{\kappa'}, r^\beta_{\mathsf{ext}}) \to (\sigma^*_i, y^*_{i,0}, y^*_{i,1})$.
   (d) **Part 2:**
   - Set $c^* = r^\beta_0 \oplus m^*_a$.
   - For $i \in [n], x \in [N] \setminus \{\mathsf{tag}^*\}$, $b \in \{0,1\}$
     i. $c^*_{x,i,b} = \mathsf{Same.Com}(1^\kappa, (x,b), y^*_{i,b}; \underline{r^*_{x,i,b}})$
   - Finally, it sends $\mathsf{com}^* = \left(\mathsf{tag}^*, r^\beta_{\mathsf{ext}}, \mathsf{HPRG.pp}^*, c^*, (\sigma^*_i, \left(c^*_{x,i,0}, c^*_{x,i,1}\right)_{x\in[N]\setminus\{\mathsf{tag}^*\}})_{i\in[n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

(e) **Post Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}, \mathcal{L})$.

(f) Receive a' from $\mathcal{A}$.

3. If $a' = a$, then output $\beta' = 1$. Otherwise output $\beta' = 0$.

Claim E.6. $\mathcal{B}$ is a non-uniform algorithm that outputs polynomial size advice and runs in time $\mathsf{poly}(2^{\kappa''^{\gamma}}) = \mathsf{poly}(2^{\kappa'}) = \mathsf{poly}(2^{\kappa^{\frac{v}{\delta}}})$.

Proof. $\mathcal{B}$ runs CCA.ValAlt that runs in time $\mathsf{poly}(|m|, 2^{\kappa^v})$ from Claim E.1. Additionally, it runs AuxEquiv.Equivocate n times that runs in time $2^{\kappa'}$. Since message lengths are polynomial in κ and N, ℓ are polynomial in κ implying n is $\mathsf{poly}(\kappa)$. The whole algorithm runs in time $2^{\kappa''^{\gamma}}$. □

Claim E.7. If $\mathcal{A}$ has advantage $|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}$, $\mathcal{B}$ has advantage in the HPRG game in Definition 5.1 $\geq \frac{1}{2p(\kappa)}$

Proof. We observe that when $\beta = 1$ in the HPRG Game - when $\mathcal{B}$ receives

$$\Bigg(r^1_0 = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0), r^1_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s),$$
$$\left\{r^1_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i), r^1_{i,\bar{s}_i} \xleftarrow{R} \{0,1\}^{\ell}\right\}_{i\in[n]}\Bigg)$$

$\mathcal{A}$ is run on exactly Game 3, and when $\beta = 0$ - i.e. when $\mathcal{B}$ receives

$$\Bigg(r^0_0 \xleftarrow{R} \{0,1\}^{\ell}, r^0_{\mathsf{ext}} = \mathsf{HPRG.ExtEval}(\mathsf{HPRG.pp}, s),$$
$$\left\{r^0_{i,s_i} = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i), r^0_{i,\bar{s}_i} \xleftarrow{R} \{0,1\}^{\ell}\right\}_{i\in[n]}\Bigg)$$

$\mathcal{A}$ is run is identical to Game 4 (barring the fact that we are replacing c^* with $m^*_\beta \oplus r^0_0$ rather than just $c^* \xleftarrow{R} \{0,1\}^{\ell}$, but these are identically distributed). So suppose $\mathcal{A}$ has probability p of winning Game 4. Then we can see that $\mathcal{B}$ wins the HPRG ($\beta' = \beta$) game either when $\mathcal{A}$ is run on Game 3 and wins, or when $\mathcal{A}$ is run on Game 4 and loses. These events happen with probabilities

$$\frac{1}{2}\left(p \pm \frac{1}{p(\kappa)}\right) + \frac{1}{2}(1-p) = \frac{1}{2} \pm \frac{1}{2 \cdot p(\kappa)}$$

for infinitely many $\kappa \in \mathbb{N}$. □

Since $\mathcal{B}$'s advantage must be negligible by Definition 5.1, a contradiction, which concludes our proof. □

Lemma E.5. For any adversary $\mathcal{A}$, $\mathsf{adv}^4_{\mathcal{A}} = 0$.

Proof. The challenge commitment is independent of the message. Thus the probability of any adversary guessing an independent random bit is $\frac{1}{2}$. □

From the above lemmas we can conclude that $\mathsf{adv}^0_{\mathcal{A}} = \mathsf{negl}(\kappa)$. This completes the proof of the theorem. □