# Black-Box Non-Interactive Non-Malleable Commitments

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

August 25, 2021

**Abstract**

There has been recent exciting progress on building non-interactive non-malleable commitments from judicious assumptions. All proposed approaches proceed in two steps. First, obtain simple "base" commitment schemes for very small tag/identity spaces based on a various sub-exponential hardness assumptions. Next, assuming sub-exponential non-interactive witness indistinguishable proofs (NIWIs), and variants of keyless collision resistant hash functions, construct non-interactive compilers that convert tag-based non-malleable commitments for a small tag space into tag-based non-malleable commitments for a larger tag space.

We propose the first black-box construction of non-interactive non-malleable commitments. Our key technical contribution is a novel way of implementing the non-interactive proof of consistency required by the tag amplification process. Prior to our work, the only known approach to tag amplification without setup and with black-box use of the base scheme (Goyal, Lee, Ostrovsky and Visconti, FOCS 2012) added multiple rounds of interaction.

Our construction satisfies the strongest known definition of non-malleability, i.e., CCA (chosen commitment attack) security. In addition to being black-box, our approach dispenses with the need for sub-exponential NIWIs, that was common to all prior work. Instead of NIWIs, we rely on sub-exponential hinting PRGs which can be obtained based on a broad set of assumptions such as sub-exponential CDH or LWE.

*UT Austin. Email:{rachg96, gclu}@cs.utexas.edu

†UIUC. Email: dakshita@illinois.edu.

‡UT Austin and NTT Research. Email: bwaters@cs.utexas.edu.

# Contents

**1 Introduction** **1**
1.1 Our Techniques . . . . . 5

**2 Preliminaries** **12**

**3 Computation Enabled CCA Commitments** **14**
3.1 Definition . . . . . 15
3.2 Connecting to Standard Security . . . . . 18

**4 Tag Amplification** **19**
4.1 Proof of Security . . . . . 22
4.2 Analysis. . . . . 27

**5 Removing the Same Tag Restriction** **34**
5.1 Proof of Security . . . . . 38
5.2 Analysis. . . . . 43

**6 Compiling our Transformations** **50**

**References** **57**

**A Proofs for Equivocal Commitments without Setup** **57**

# 1 Introduction

Non-malleable commitments have been a well studied primitive in cryptography since their introduction by Dolev, Dwork and Naor [DDN91]. They are an important component of nearly all multi-party protocols including multi-party computation, coin flipping and secure auctions. These commitments ensure security in the presence of "man in the middle" attacks. A man-in-the-middle adversary participates in two or more instantiations of a protocol, trying to use information obtained in one execution to breach security in the other protocol execution. A non-malleable protocol should ensure that such an adversary gains no advantage from such behavior.

**Non-Interactive Non-Malleable Commitments.** For several years, provably secure constructions of non-malleable commitments required several rounds of interaction. On the other hand, practical constructions need to be highly efficient and often non-interactive. For these reasons, in practice, we often heuristically assume that a family of (keyless) SHA-like hash functions is non-malleable. Our technique gives the first provably secure black-box construction of non-interactive non-malleable commitments, taking us a step closer to efficient realizations.

We will focus on computationally hiding and binding non-interactive commitments. For these commitments, the binding requirement will assert that it is computationally hard to produce a commitment that can be opened to two different messages $m \neq m'$. Specifically, no adversary can generate with non-negligible probability a malicious commitment string $c$, for which there exist two openings to messages $m$ and $m'$ such that $m \neq m'$. The (computational) hiding property asserts that for any two messages, $m$ and $m'$ (of the same length), the distributions of commitments $\mathsf{com}(m)$ and $\mathsf{com}(m')$ are computationally indistinguishable.

Loosely speaking, a commitment scheme is said to be non-malleable if no adversary, given a commitment $\mathsf{com}(m)$, can efficiently generate a commitment $\mathsf{com}(m')$, such that the message $m'$ is related to the original message $m$. This is equivalent (assuming the existence of one-way functions) to a tag-based notion where the commit algorithm obtains an additional input, a $\mathsf{tag} \in \{0,1\}^\kappa$, and where the adversary is restricted to using a tag, or identity, that is different from the tag used to generate its input commitment. We will rely on tag-based definitions throughout this paper. We will also model man-in-the-middle security as a CCA (chosen commitment attack) game between the adversary and a challenger.

Specifically, we will modify the hiding game to give the adversary oracle access to an inefficient value function $\mathsf{CCA.Val}$ where on input a string $c$, $\mathsf{CCA.Val}(\mathsf{tag}, c)$ will return $m$ if $\mathsf{CCA.Com}(\mathsf{tag}, m; r) \rightarrow c$ for some $r$. If the commitment is not well-formed, $\mathsf{CCA.Val}$ will return the unique message that the commitment can be opened to (and if no unique message exists, $\mathsf{CCA.Val}$ can behave arbitrarily – but by the binding condition, this event only occurs with negligible probability).

The game is as follows: the adversary must first specify a challenge tag $\mathsf{tag}^*$, along with messages $m_0^*, m_1^*$. He 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 com-

[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 from this construction by setting the tag as the verification key of a signature scheme, and signing the commitment string using the signining key.

mitment is a special case of this definition where the adversay is only allowed to make parallel oracle queries once it obtains the challenge commitment.

**Our Results, in a Nutshell.** In this work, we give the first black-box construction of CCA secure commitments, under weaker assumptions than prior work. In terms of assumptions, we substitute NIWIs with hinting PRGs [KW19] which can be instantiated under several standard assumptions like CDH and LWE. Additionally, while all prior work recursively applied NIWIs to prove cryptographic statements, making heavy non-black-box use of cryptography, our constructions are black-box. Combining this with base schemes due to [KK19], we obtain CCA secure commitments from black box use of the following assumptions: subexponential hinting PRGs, subexponential keyless collision-resistant hash functions, subexponential (injective) one-way functions/non-interactive commitments against quantum adversaries, and subexponential (injective) one-way functions/non-interactive commitments in BQP with hardness against classical adversaries. We note that subexponential hinting PRGs can be obtained based on black-box use of any group where CDH is subexponentially hard.

We believe this takes us one step closer to the goal of building provably secure and efficient non-interactive non-malleable commitments.

**Prior Work on Non-Malleable Commitments.** There has been a long line of work constructing non-malleable commitments in the plain model, without trusted setup. This research has been driven by two often competing goals: the first is to reduce the round complexity of commitment, which is important because it directly impacts the round complexity of applications like MPC. The second goal is to achieve non-malleable commitments under the weakest possible assumptions.

This research [DDN91, Bar02, PR05, PR08, LPV, PPV08, LP09, Wee10, PW10, LP, Goy11, GLOV12, GRRV14, GPR16, COSV17] culminated in three round stand-alone secure non-malleable commitments based on injective one-way functions [GR19] and concurrenct secure non-malleable commitments based on DDH/LWE [Khu17], or subexponential injective one-way functions [COSV16]. In the two round setting, we now have constructions based on sub-exponential time-lock puzzles [LPS17] and sub-exponential DDH/LWE/QR/NR [KS17].

Very recently, research in non-malleable commitments moved to a final frontier of achieving non-interactive non-malleable commitments from well-studied assumptions without leveraging trusted setup. In this non-interactive setting, Pandey, Pass and Vaikuntanathan [PPV08] first gave constructions of non-malleable commitments based on a strong non-falsifiable assumption. The primary research challenge has been to improve assumptions while realizing non-malleability in very resource constrained environments, which do not allow the use of tools like zero knowledge proof systems.

Nevertheless, the recent works of Bitansky and Lin [BL18] and Kalai and Khurana [KK19] made progress on improving these assumptions. All of these works [KS17, LPS17, BL18, KK19] proceed in two steps. First, they construct "base" commitment schemes that only support a constant-sized space of tags. Second, they give amplification techniques to convert commitments supporting a small space of tags into commitments that support a much larger tag space. Applying these amplification techniques to the base scheme helps generically increase the space of tags to $\{0,1\}^\kappa$.

We summarize known results in the non-interactive setting by splitting up contributions into base constructions and tag amplification results.

**Base Constructions.** Three recent works [LPS17, BL18, 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. These are typically only secure in a setting where the adversary is restricted to using the same tag in all its queries to the CCA.Val oracle.

This is primarily achieved by using families of assumptions, each of which is harder than the other along some axis of hardness. We list these assumptions below.

1. Lin, Pass and Soni [LPS17] assume a sub-exponential variant of the hardness of time-lock puzzles. Specifically, they define a two-dimensional variant of the Rivest, Shamir and Wagner (RSW) repeated squaring assumption there is a security parameter $n$ and another parameter $t$, and it is required that computing $h = g^{2^{2^t}}$ cannot be done by circuits of overall size $2^{n^\epsilon}$ and depth $2^{t^\delta}$, for constants $\epsilon$ and $\delta$.

2. Bitansky and Lin [BL18] rely on sub-exponentially hard injective one-way functions that admit a strong form of hardness amplification. Roughly speaking, they say that a one-way function $f$ is amplifiable, if there is a way to combine (e.g. XOR), say $\ell$ hardcore bits corresponding to $\ell$ independent images $f(x_1), \ldots, f(x_\ell)$ that are each hard against $T$-time adversaries, so that the combined bit is $2^{\ell^\epsilon}$-unpredicatable against $T'$-time adversaries; that is, the level of unpredictability increases at least subexponentially as more hardcore bits are combined (their assumption on unpredictability goes beyond the limit $\mathsf{poly}(\frac{T}{T'})$ that is commonly imposed by known provable results on hardness amplification).

3. Kalai and Khurana [KK19] assume classically sub-exponentially hard but quantum easy injective one-way functions (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.** Starting with non-malleable commitments for a tag space of size $c \log \log \kappa$ for a specific constant $c > 0$ (or sometimes even smaller), several works develop techniques to achieve non-malleable commitments for a tag space of $\{0,1\}^\kappa$. This is achieved by several applications of a tag-amplification compiler, that increases the tag space exponentially in each application. We also point out that these compilers often obtain as input base schemes that are secure against a restricted adversary; one that uses the same tag in all its queries to the CCA.Val oracle. The end goal, however, is to obtain security against a general adversary, that uses arbitrary tags in its oracle queries – as long as all tags in oracle queries are different from the challenge tag.

Such compilers were developed in [LPS17, BL18, KK19] based various assumptions, and we summarize these results below.

- Lin, Pass and Soni [LPS17] assume sub-exponential non-interactive witness indistinguishable (NIWI) proofs and keyless collision resistant hash functions against uniform adversaries. The resulting non-malleable commitments for larger tag space are also secure only against uniform adversaries.

- Bitansky and Lin [BL18] assume sub-exponential non-interactive witness indistinguishable (NIWI) proofs and keyless collision resistant hash functions with limited security against non-uniform adversaries. Such a hash function $H : \{0,1\}^{3\kappa} \to \{0,1\}^\kappa$ guarantees that no superpolynomial adversary with non-uniform description of polynomial size $S$ can find

more than $K(S)$ collisions in the underlying function. Here, $K$ is a fixed polynomial (e.g., quadratic). The resulting non-malleable commitments for larger tag space are secure against non-uniform adversaries.

- Kalai and Khurana [KK19] assume sub-exponential non-interactive witness indistinguishable (NIWI) proofs and obtain security against non-uniform adversaries. However their compiler, on input commitments that satisfy a weaker notion of non-malleability w.r.t. replacement generates commitments that are non-malleable w.r.t replacement for a larger tag space.

In [LPS17, BL18], NIWIs are combined with a hard-to-invert trapdoor statement to enable weak forms of NIZKs without setup. In contrast, [KK19] use NIWIs without associated trapdoors, but then only achieve weaker forms of non-malleability (that is, w.r.t. replacement).

But a common thread among the amplification techniques is that they all require the use of sub-exponential NIWI proofs. We remind that reader that NIWIs are one round proof systems with statistical soundness, for which no computationally bounded verifier can distinguish which witness in a relation was used to create the proof.

Reliance on NIWIs results in the following less than ideal consequences:

- Subexponential NIWIs are only known based on the hardness of the decisional linear problem over bilinear maps [GOS12], or based on derandomization assumptions, together with subexponential trapdoor permutations [BOV07].

- All these compilers use NIWIs to prove complex cryptographic statements, and therefore make non-black box use of the underlying non-malleable commitment for a smaller tag space. On the other hand, from the point of view of efficiency, it is desirable to have constructions that make black-box use of cryptography.

**Our Results.** In this work, we provide a new approach to non-interactive tag amplification for non-malleable commitments. This approach only makes black-box use of cryptography, and achieves provable security under a more diverse set of assumptions. Specifically, this compiler replaces the NIWI assumption with hinting PRGs, that were introduced by Koppula and Waters [KW19], and can be obtained based on CDH, LWE [KW19] and also $\phi$-hiding and DBDHI assumptions [GVW19]. (One can also alternatively execute the paradigm from any projective key-dependent secure symmetric key encryption scheme [KMT19] which is realizable from the LPN assumption).

We summarize (a simplification of) our results via the following informal theorems. Recall that base schemes are typically only secure in a setting where the adversary is restricted to using the same tag in all its queries to the oracle. In what follows, we refer to such a commitment scheme that is only secure against this limited class of adversaries as a same-tag CCA secure commitment. We also refer to CCA commitments where the adversary is only allowed to make parallel oracle queries after obtaining the challenge commitment, as non-malleable commitments.

**Informal Theorem 1.1.** (Removing the Same-Tag Restriction) Assuming the existence of sub-exponentially secure hinting PRGs and keyless hash functions that are collision-resistant against sub-exponential uniform adversaries, there exists a compiler that on input any same-tag CCA (respectively, non-malleable) non-interactive commitment for $N$ tags secure against *non-uniform*

*adversaries* where $N \leq \mathsf{poly}(\kappa)$, outputs a CCA (respectively, non-malleable) non-interactive commitment for $N$ tags secure against *uniform adversaries*.

**Informal Theorem 1.2.** (Tag-Amplification for CCA commitments) Assuming the existence of sub-exponentially hinting PRGs and keyless hash functions that are collision-resistant against sub-exponential uniform adversaries, there exists a compiler that on input any CCA (respectively, non-malleable) non-interactive commitment for $N$ tags secure against *non-uniform adversaries* where $N \leq \mathsf{poly}(\kappa)$, outputs a CCA (respectively, non-malleable) non-interactive commitment for $2^{N/2}$ tags secure against *uniform adversaries*.

Unfortunately, using these informal theorems to amplify tag space from $c \log \log n$ for a small constant $c > 0$ immediately encounters the following issue: the input scheme to the compiler is required to be *non-uniform secure*, whereas the output scheme is only *uniform secure*.

To enable recursion, we strengthen our CCA abstraction. Specifically, we modify the CCA security game to allow an adversary to submit a Turing Machine $P$ to the challenger, and obtain the evaluation of $P$ on an input of the adversary's choice. We say that a scheme is $e$-"computation enabled" if it is secure against all adversaries that submit programs that run in time polynomial in $2^{\kappa^e}$ for constant $e$. As such, we will substitute the *non-uniform security* requirement for the base CCA scheme and instead require it to be $e$-"computation enabled" for an appropriate constant $e$. The output of the compiler will be an $e'$-"computation enabled" commitment for an appropriate constant $e'$. We describe this abstraction, and our techniques, in additional detail in Section 1.1.

### 1.1 Our Techniques

We now provide our technical overview. Recall that the core technical goal of our work is to provide a method for amplifying from a commitment scheme for $O(N)$ sized tag space to a $2^N$ sized space. If the computational overhead associated with the amplification step is polynomial in $N$ and the security parameter $\kappa$, then the 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 $2^\kappa$. Here, we note that subexponential quantum hardness of LWE and subexponential hardness of DDH [KK19], or subexponential hardness amplifiable one-way functions [BL18], or appropriate subexponential variants of time-lock puzzles [LPS17] imply base schemes for tags in $(c \lg \lg \kappa)$ for a small constant $c > 0$, which means they imply schemes for tags in $(\lg \lg \lg \kappa)$.

Now the traditional way to amplify such a tag space can be traced back to [DDN91][2] They suggested 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 $T^j$.

A scheme for tags in $2^N$ will have an algorithm $\mathsf{CCA.Com}$ that commits to a message $m$ as $\mathsf{CCA.Com}(1^\kappa, \mathsf{tag}, m; r) \rightarrow \mathsf{com}$. To commit to $m$ under tag one first creates $N$ tags $t_1, \ldots t_N$ by applying the DDN encoding to tag. Next, these (smaller) tags are used to generate commitments of $m$ in the smaller tag scheme as $c_i = \mathsf{Small.Com}(1^\kappa, (t_i), \mathsf{msg} = m; r_i)$ for $i \in [N]$. Next, the committer attaches a zero knowledge (ZK) proof that all commitments are to the same message $m$ using the random coins as a witness. Since we are interested in non-interactive amplification,

[2]This was recently further optimized by [KS17] but in this paper, we use the [DDN91] technique for simplicity.

the ZK proof will need to be non-interactive. Additionally, we will require it 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.

CCA security of the scheme with larger tag space can be argued in two basic steps. Suppose the challenger commits to either $m_0^*$ or $m_1^*$ under tag $T^*$ (we denote the DDN encoding of $T^*$ by $t_1^*, \ldots t_N^*$). The adversary wins if it gets which out of $m_0^*$ and $m_1^*$ was committed. Recall that the adversary can request the CCA oracle to provide openings of commitment string with tags $\mathsf{tag} \neq \mathsf{tag}^* \in \{0,1\}^N$. This oracle generates a response as follows:

1. Verify the ZK proof in the commitment string. Return $\perp$ if verification does not accept.
2. Open the underlying commitment scheme with small tags at position $1$ with tag $t_1$.

We will assume, for simplicity, that the adversary makes a single oracle query in the CCA game, with tag $T$, whose DDN encoding is denoted by $t_1, \ldots t_N$. We will focus on the index $i$ in the adversary's oracle query, such that the tag $t_i \notin \{t_1^*, \ldots t_N^*\}$.

As a first step towards proving CCA security, one can modify the oracle to open the commitment string $c$ with small tag $t_i$, in Step 2. Because of the soundness of the ZK proof system, this change cannot be detected by the adversary, except with negligible probability.

At this point, the challenge commitment is modified so that the ZK proof is simulated and does not need the random coins used in the small tag commitments anymore. To argue indistinguishability, we will need to answer the adversary's oracle queries. This will be done by extracting, via brute-force, the value committed in the adversary's oracle query. As such, we will need to rely on ZK proofs where the ZK property holds even against machines that can (brute-force) break the small tag commitnents. Once this is done, we will change each of the small tag commitments in the challenge commitment from committing to the message $m_b^*$ to committing to the all $0$'s string, one by one. At the same time, the oracle will continue to open the commitment string $c$ with small tag $t_i$, in Step 2. Since $t_i \notin \{t_1^*, \ldots t_N^*\}$, we can rely on CCA security of the underlying small tag scheme and argue that the adversary will not be able to detect these changes. By the end of this process, all information about the bit $b$ will be erased.

Since non-interactive zero-knowledge proofs without setup are impossible, existing non-interactive tag amplification techniques [LPS17, KS17, BL18] rely on weaker variants of zero-knowledge proofs, such as ZK with super-polynomial simulation and weak soundness, to perform tag amplification via the afore-mentioned outline. These required variants of non-interactive ZK proofs are obtained by including a trapdoor statement $\mathsf{td}$. To prove that a statement $x$ is in an NP language $L$, one typically provides a NIWI to establish that $(x \in L) \vee (\mathsf{td}\text{ is true})$. The trapdoor statement helps perform simulation, whereas for soundness it is required that the adversary cannot prove the trapdoor statement. One exception is [KK19], which only relies on NIWIs and does not make use of on any trapdoor statements, but is limited to the weaker notion of replacement security. However, in addition to relying on NIWIs, the outline above makes non-black-box use of the underlying base commitment scheme.

**Eliminating NIWIs.** Our primary goal in this paper is to perform tag amplification without NIWIs, and while making black-box use of the underlying base commitments. Taking a step back, the reason ZK is required in the tag amplification argument discussed above, is that we can change the oracle to one that opens different underlying tags, without the adversary noticing. In other

words, we would like to establish a system where the adversary cannot submit a commitment such that its opening will be different under the original and new oracle functions.

Here, inspired by recent work in chosen ciphertext secure public key encryption [KW19], our construction will allow the oracle to recover a PRG seed $s$ that gives (a good part of) the randomness used to create the underlying commitments. Specifically, the oracle will use the commitment with a specific small tag to first recover a candidate PRG seed $s'$ and then check for consistency by re-evaluating the underlying commitment pieces, and checking them against the original.

These checks will intuitively serve as a substitution for ZK proofs. Interestingly, our checking algorithm will allow some partially malformed commitments to go through – allowing this is essential to our security argument. This is in contrast to a ZK proof which enforces that all must be commitments to the same message. While creating such partially malformed commitments is actually easy for the adversary, the adversary will still not be able to differentiate between different forms of decryption. (We note that in non-malleable encryption some systems [PSV06, CDSMW09] allow for somewhat malformed ciphertexts to be let through.) Importantly, unlike [KW19] that looked at two possible decryption strategies, we will need to ensure that up to polynomially many such strategies decrypt the same way. Furthermore, we will not be able rely on trusted setup to generate verification keys for a signature scheme. Instead, we will develop a new technique leveraging hinting PRGs, which we outline below.

We now describe our new tag amplification technique that converts CCA commitments with $4N$ tags to CCA commitments with $2^N$ tags. We point out that our technique also applies as is to converting non-malleable (i.e. parallel CCA) commitments with $4N$ tags to non-malleable commitments with $2^N$ tags. First, we summarize some of the tools we will use.

- **Hinting PRGs.** A hinting PRG, introduced in [KW19], satisfies the following property: for a uniformly random short seed $s$, the matrix $M$ obtained by first expanding $PRG(s) = z_0 z_1 z_2 \dots z_n$, sampling uniformly random $v_1 v_2 \dots v_n$, and setting for all $i \in [n]$, $M_{s_i,i} = z_i$ and $M_{1-s_i,i} = v_i$, should be indistinguishable from a uniformly random matrix. Hinting PRGs are known based on CDH, LWE [KW19] and more generally, any circular secure symmetric key encryption scheme [KMT19].

- **Statistically Equivocal Commitments without Setup.** We will rely on statistically hiding bit commitments without setup, that satisfy binding against uniform adversaries. Additionally, these commitments will be statistically equivocal, that is, with overwhelming probability, a randomly chosen commitment string can be opened to both a $0$ and a $1$. These can be 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.

**Outline of Our Tag Amplification Technique.** Let $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ be a non malleable commitment for $4N$ tags. 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 requires randomness of length $\ell(\kappa)$.

Our transformation will produce three algorithms, $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$. The $\mathsf{CCA.Com}$ algorithm on input a tag $\mathsf{tag}$ 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 $\widetilde{r}_{x,i}$ 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 $\mathsf{com}$, 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 $\sigma_i$. 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 $\mathsf{com}$ 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 differently depending on which small tag was used. If all these checks pass, the $\mathsf{CCA.Val}$ algorithm uses $c$ to recover and output $m$.

**Proving Security.** We will prove that the resulting scheme is CCA secure against uniform adversaries. To begin, we note that the set $\{(x, \mathsf{tag}_x)\}_{x\in[N]}$ is nothing but the DDN encoding of the tag $\mathsf{tag}$. Recall that this encoding has the property that for every $\mathsf{tag}, \mathsf{tag}^* \in 2^N$, there exists an index $x \in [N]$ such that $(x, \mathsf{tag}_x) \notin \{(x^*, \mathsf{tag}^*_{x^*})\}_{x^*\in[N]}$. In the scheme described above, the tag used for each set $\{c_{x,i,b}\}_{i\in[n]}$ is $(x, \mathsf{tag}_x, b)$. 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*.

Our first step towards proving security of the resulting commitment with large tags, will be to define an alternative $\mathsf{CCA.ValAlt}$ 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 $(x', \mathsf{tag}_{x'}, 1)$. As already alluded to earlier, this scheme is designed so that it is computationally infeasible for a uniform adversary to query the oracle on commitment strings for which $\mathsf{CCA.Val}$ and $\mathsf{CCA.ValAlt}$ lead to different outcomes. Formally, we will first switch to a hybrid that uses the $\mathsf{CCA.ValAlt}$ algorithm instead of $\mathsf{CCA.Val}$ to answer the adversary's oracle queries.

When making this change, because of the checks performed by the valuation algorithms, we can formally argue that any adversary that distinguishes these hybrids 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. Assuming that the equivocal commitment satisfies binding against uniform adversaries that run in subexponential time, one can brute-force extract these openings from $c_{x,i,0}$ and $c_{x,i,1}$ to contradict the binding property.

The next hybrid is an exponential time hybrid that samples 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.

In the next hybrid, inspired by [KW19] we modify the components $\{c^*_{x,i,b}\}_{x\in[N],i\in[n],b\in\{0,1\}}$ in the challenge commitment to "drown" out information about $s$ via noise. In particular, while in the real game, the values $c^*_{x,i,1}$ are always commitments to $y_{s_i,i}$, in the challenge commitment these values are modified to become commitments to $y^*_{i,1}$, irrespective of what $s_i$ is. In the next step, the values $c^*_{x,i,0}$ are modified to become commitments to $y^*_{i,0}$, irrespective of what $s_i$ is. We rely on CCA security of the underlying small tag scheme so that we can continue to run the $\mathsf{CCA.ValAlt}$ function to recover values committed under $(x', \mathsf{tag}_{x'}, 0)$ and $(x', \mathsf{tag}_{x'}, 1)$ while changing all the components $\{c^*_{x,i,b}\}_{x\in[N],i\in[n],b\in\{0,1\}}$ in the challenge commitment. 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*. Moreover, in spite of the fact that generating equivocal openings of $\{\sigma_i\}_{i\in[n]}$ takes exponential time, the proof of indistinguishability between this hybrid and the previous one does not need to rely on an exponential time reduction. Instead, we observe that the equivocal commitment strings $\{\sigma_i\}_{i\in[n]}$ together with their openings can be fixed non-uniformly and independently of the strings $c^*_{x,i,b}$, and therefore these hybrids can be proven indistinguishable based on non-malleability of the small tag commitment against non-uniform adversaries. Since we must carefully manipulate the randomness used for $c^*_{x,i,b}$ in both games, this hybrid requires a delicate argument.

At this point, we have eliminated all information about the PRG seed $s$, except from the randomness $r_{x,i}$ and $\widetilde{r}_{x,i}$. In the final hybrid, we rely on the security of the hinting PRG to switch to using uniform randomness everywhere. Note that we still need to answer the adversary's oracle queries, but this can be done by ensuring that the time required to run the $\mathsf{CCA.ValAlt}$ algorithm is much smaller than that needed to break hinting PRG security. At this point, we have eliminated all information about $s$, and therefore about the message being committed to in the challenge commitment.

**Issues with Recursion.** At this point, it may seem like we are done, but the careful reader may have noticed a problem. To prove security, we assumed an input scheme that was secure against *non-uniform* adversaries, but due to the use of equivocal commitments against uniform adversaries, the transformation yields a scheme that is only secure against *uniform* adversaries. This would be no problem if we say were only amplifying once from $\kappa$ to $2^\kappa$ tags. But unfortunately, the recursion will not work if our base scheme starts with $\lg\lg\lg(\kappa)$ size tags (which is the number of tags allowable by most existing base schemes), as we will need to recursively amplify multiple times.

It might seem that we are fundamentally stuck. The first hybrid in our argument requires the equivocal commitment scheme to be more secure than the underlying small tag commitment. Later hybrids require that the small tag commitment to satisfy CCA security even when equivocal commitments with openings to both ones and zeros are generated. If the small tag CCA scheme

is only uniformly secure, it seems impossible to satisfy this requirement without violating the previous one.

However, if we peel the recursion back further, there appears to be a glimmer of hope. Suppose we are applying our transformation to an underlying CCA commitment, which is itself the result of applying the transformation one or more times. When our proof arrives at the security of the underlying scheme, the underlying scheme's security will rely both on an equivocal commitment itself, and at the deepest level the non-uniform security of the base scheme. If the equivocal commitments in the underlying scheme use a larger security parameter than the current one, then the lower level scheme may still be secure (and lower level equivocal commitments may still be binding) even when equivocal openings are found at the current level.

**$e$-Computation Enabled Security.** We capture this intuition by expanding our abstraction to include what we call $e$-computation enabled CCA commitments. Here, we modify the security game to allow an adversary to submit a Turing Machine $P$ to the challenger. The adversary will receive the evaluation of $P$ on an input of its choice. We say that a scheme is $e$-computation enabled if it is secure against all adversaries that submit programs that run in time polynomial in $2^{\kappa^e}$ for constant $e$. (The program output size itself is required to be polynomially bounded.)

With this abstraction in place, when proving security, our reduction can pass the task of generating equivocal openings as an appropriate program $P$ to the enhanced CCA security game itself. Implicitly, this allows the equivocal opening requests to be satisfied in different ways depending on what stage the security proof of the lower scheme is at.

While this new property provides a useful tool for recursion, we also need to work a bit harder to prove $e$-computation enabled CCA security. Specifically, we prove in Section 4 that given a hinting PRG and an equivocal commitment scheme that are uniformly secure against $2^{\kappa^\delta}$ time adversaries for $\delta \in (0, 1)$, we can transform an $e$-computation enabled CCA scheme for small tags into one that is $e'$-computation enabled CCA secure for large tags, where $e' = e \cdot \delta$.

In our proof, at the stages where we use a reduction to find equivocal openings, the reduction will run in time $2^{\kappa^{e'}}$ to satisfy the adversary's program request. When contradicting the hinting PRG, the reduction will run in time $2^{\kappa^e}$ to to find equivocal openings, and $2^{\kappa^{e'}}$ to satisfy the adversary's program request. To ensure that this gives us a contradiction, we will set the security parameter of the hinting PRG to be large enough. Finally, when the reduction is to the underlying small tag CCA commitment, the program request of the large tag adversary will be passed by the reduction to the interface of the underlying small tag scheme, which is allowed since $e' < e$. In the base case, we note that we start with schemes secure against non-uniform adversaries (for $\lg \lg \lg \kappa$ tags). By definition, any scheme that is secure against non-uniform adversaries is trivially $e$-computation enabled secure for arbitrary $e$.

**Issues due to Same-Tag Restrictions.** The techniques described above capture our main ideas for tag amplification. Unfortunately, the base schemes that we start with may only be same-tag secure. On the other hand, we would like to end up with CCA schemes for $2^\kappa$ tags that do not have this restriction. This is because CCA commitments without such a restriction can be generically transformed, assuming signatures into schemes that do not use tags at all. We remedy the same-tag issue by applying a transformation that takes a scheme supporting a tag space of $N$ tags with same-tag only queries to one that supports $N$ tags without the same-tag restriction, for any $N \leq$

$\mathsf{poly}(\kappa)$. We highlight the technical ideas involved in this transformation below.

**Removing the Same-Tag Requirement.** We start with an underlying scheme that has the same-tag requirement, and modify it to remove this requirement as follows. To commit to a message with tag $\mathsf{tag}$ in the new scheme, commit to it with respect to all $N-1$ tags *except* $\mathsf{tag}$ in the underlying same-tag scheme. Similar to the previous construction, we use hinting PRGs and attach a bunch of checks to ensure that recovering the committed value from the adversary's queries using any one tag is computationally indistinguishable from recovering it using a different tag.

The overall mechanics and guarantees are similar to our prior transformation. Suppose an adversary were given a challenge commitment $\mathsf{tag}^*$ in the transformed scheme, and got to make queries to several *different* tags $\mathsf{tag} \neq \mathsf{tag}^*$. By our construction, the adversary's challenge does not contain an underlying commitment with tag $\mathsf{tag}^*$ whereas all of the adversary's oracle queries will contain an underlying commitment with tag $\mathsf{tag}^*$. We can therefore answer all of these queries by changing the oracle valuation function to one that uses only tag $\mathsf{tag}^*$ in underlying scheme.

We note that since the same-tag transformation incurs a blowup proportional to $N$, it is imperative to apply it early on in the sequence of transformations. If we first amplified the tag space to be of size $2^\kappa$ and then attempted to remove the same-tag restriction, the resulting scheme would have exponential sized commitments. Therefore, the order in which these transformations are applied is important. If we start with a base scheme that is same-tag secure and supports tags of size $\underbrace{\lg \lg \cdots \lg}_{c \text{ times}}(\kappa)$ for some constant $c$, we will first apply the same-tag to many-tag transformation. Next, we apply the tag amplification transformation $c+1$ times. We end up with a scheme that is polynomial sized and supports a tag space of size $2^\kappa$ with no same-tag restrictions.

This concludes our technical overview. We refer the reader to Section 3 for a formal definition of computation enabled CCA commitments, Section 4 for our tag amplifying transformation, and Section 5 for details on removing the same-tag requirement. We put things together and state our final results in Section 6.

**Non-uniform Security.** Our techniques, as described above, give a CCA commitment scheme secure against uniform adversaries. One might ask whether we could use similar techniques, perhaps combined with new assumptions such as non-uniformly secure keyless hash functions [BKP18, BL18] to obtain security against non-uniform adversaries. We address this in two parts.

First, taking a step back, a primary motivation for obtaining non-uniform security is that it is useful for protocol composition. For example, if we were using a cryptographic primitive like public key encryption as an end application say for encrypting email, then obtaining uniform security would arguably be just fine. As the uniform model captures attackers in the real world. However, the extra power of non-uniform security might be helpful if our commitment scheme were a component used in building a larger cryptosystem. Here, we observe that our transformation actually outputs a CCA scheme with properties that are stronger than (plain) uniform security. Specifically, the output scheme satisfies $e$-computation enabled CCA security.

While the initial motivation for this abstraction was that it helps with recursion; we note that it can actually be a useful property for a CCA scheme to have as well. In particular, it can actually be viewed as a more fine-grained or nuanced view of non-uniform computation. This abstraction essentially gives any adversary non-uniform advice so long as it can be computed in time of $2^{\kappa^e}$. If $e$ is set appropriately, then we expect this would suffice in many circumstances, including for

protocol composition. Indeed, this was true for the type of protocol composition that we needed to recursively amplify the tag space. Thus our amplification techniques and our abstraction can arguably deliver something that is the "best of both worlds": the outcome is as good as non-uniform security for many applications, but does not make any new non-uniform assumptions about the hash function.

Second, our techniques are also meaningful for constructing black-box two-message non-malleable commitments with (regular) non-uniform security. In our transformation, the primitive that requires uniform security is the keyless hash-based equivocal commitment scheme. In the two-message setting, it seems possible to slightly modify our scheme to have the receiver generate the key for a keyed (non-uniform secure) collision-resistant hash function. All of our other techniques appear to carry over to this setting, and it appears that one would be able to prove that the resulting scheme is a (regular) non-uniform secure non-malleable commitment that only makes black-box use of cryptography.

**The Subtle Issue of Over-Extraction.** In the discussion above, we implicitly assumed that the CCA decommitment oracle for small-tag/same-tag commitments behaves perfectly correctly on well-formed as well as malformed commitments: 1) whenever an adversary queries the oracle on a commitment that is valid, in the sense that there exists a value and randomness that would lead to an accepting decommitment, the oracle outputs exactly the committed value, otherwise, 2) when the commitment is invalid, and there is no value and randomness that would lead to an accepting decommitment, it outputs $\perp$.

However, the base scheme from time-lock puzzles in [LPS17] achieves a weaker notion of same-tag CCA security. In particular, the CCA decommitment oracle for this scheme will not satisfy property 2) above. When the commitment is invalid, the oracle may output arbitrary values - this is known as *over-extraction*. In a nutshell, as described in [LPS17] the time-lock puzzle based same-tag CCA commitments suffer from over-extraction because only honestly generated time-lock puzzles (i.e., in the domain of the puzzle generation algorithm) are guaranteed to be solvable by circuits of sufficient depth. There is no guarantee for ill-generated puzzles, and no depth-efficient procedure for deciding whether a puzzle is honestly generated or not. Eg., in the time-lock puzzles of Rivest, Shamir, and Wagner [RSW96], given a puzzle $(s + g^{2^{2^t}} \mathsf{mod} N, N)$ one can extract $s$ using $2^t$ squaring modular $N$, but cannot obtain a proof that $N$ is a valid RSA-modulus. As a result, the CCA decommitment oracle that extracts committed values via solving time-lock puzzles, provides no guarantees when commitments are invalid.

Nevertheless, our compiler removing the same-tag restriction boosts security so that even if the base CCA scheme suffers from over-extraction (i.e. satisfies property 1) only), the resulting scheme satisfies both properties 1) and 2). In a nutshell, the CCA check algorithm built into our construction prevents over-extraction by ensuring that whenever the oracle returns a value that is not $\perp$, there exists a decommitment to a valid value that would cause the decommitter to accept.

## 2 Preliminaries

In this section we will provide notions and security definitions for some of the building blocks that we will use in our constructions. We will use $\kappa$ to denote the security parameter. For our computational definitions, we will use $T(\cdot)$ security to indicate security against any attacker that runs in

time that is polynomial in $T(\cdot)$. In our constructions we will often utilize subexponential assumptions where $T(\kappa) = 2^{\kappa^\delta}$ for some constant $\delta \in (0,1)$. Finally, we will be explicit to whether we are describing security against uniform or non-uniform adversaries as our results will be sensitive to this nuance.

We will denote by $\mathsf{negl}(\kappa)$ a function that is asymptotically smaller than the inverse of every polynomial in $\kappa$.

## Hinting PRGs

We now provide the definition of hinting PRGs taken from [KW19]. Let $n(\cdot,\cdot)$ be a polynomial. A $n$-hinting PRG scheme consists of two PPT algorithms $\mathsf{Setup}, \mathsf{Eval}$ with the following syntax.

$\mathsf{Setup}(1^\kappa, 1^\ell)$: The setup algorithm takes as input the security parameter $\kappa$, and length parameter $\ell$, and outputs public parameters $\mathsf{pp}$ 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 $\mathsf{pp}$, an $n$ bit string $s$, an index $i \in [n] \cup \{0\}$ and outputs an $\ell$ bit string $y$.

**Definition 2.1.** A hinting PRG scheme $(\mathsf{Setup}, \mathsf{Eval})$ is said to be $T(\cdot)$ secure if for any polynomial $\ell(\cdot)$ and any adversary $\mathcal{A}$ running in time $p(T(\kappa))$ for some polynomial $p$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds:

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

## Equivocal Commitments without Setup

Equivocal commitments were proposed by DiCrescenzo, Ishai and Ostrovsky [CIO98] as a bit commitment scheme with a trusted setup algorithm. During normal setup, the bit commitment scheme is statistically binding. However, there exists an alternative setup which produces public parameters along with a trapdoor, that produces commitments which can be opened to either 0 or 1. Moreover, the public parameters of the normal and alternative setup are computationally indistinguishable.

Here we will define a similar primitive, but without utilizing a trusted setup algorithm. In order for such a notion to be meaningful, we will require the commitment scheme to be computationally binding for any *uniform* $T$-time attacker, but there will exist an algorithm running in time $\mathsf{poly}(2^\kappa)$ that can be opened to 0 or 1. Moreover, such a commitment with one of the openings should be statistically indistinguishable from a commitment created in the standard manner.

In Appendix A, we prove that any statistically hiding, computationally binding commitment also satisfies equivocality according to this definition. We can then instantiate commitments satisfying our definition using constructions of statistically hiding and computationally binding commitments due to [DPP93, HM96] and more recently [BKP18]. These constructions combine a pairwise independent hash (or more generally a strong extractor) with a keyless collision resistant hash function to obtain such a commitment scheme.
An equivocal commitment scheme without setup consists of three algorithms:

$\mathsf{Equiv.Com}(1^\kappa, b) \to (c, d)$ is a randomized PPT algorithm that takes in a bit and security parameter and outputs a commitment $c$, decommitment string $d$.

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

$\mathsf{Equiv.Equivocate}(1^\kappa) \to (c, d_0, d_1)$ is an (inefficient) randomized algorithm that takes in the security parameter, and outputs decommitment strings to both $0$ and $1$.

**Definition 2.2.** We say an equivocal commitment scheme is perfectly correct if for all $b \in \{0, 1\}$

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

**Definition 2.3.** We say an equivocal commitment scheme is efficient if $\mathsf{Equiv.Com}$ and $\mathsf{Equiv.Decom}$ run in $\mathsf{poly}(\kappa)$ time, and $\mathsf{Equiv.Equivocate}$ runs in time $2^\kappa$.

We now define the binding and equivocal properties.

**Definition 2.4.** An equivocal commitment without setup scheme ($\mathsf{Equiv.Com}$, $\mathsf{Equiv.Decom}$, $\mathsf{Equiv.Equivocate}$) is said to be $T(\cdot)$ binding secure if for any *uniform* adversary $\mathcal{A}$ running in time $p(T(\kappa))$ for some polynomial $p$, there exists a negligible function $\mathsf{negl}(\cdot)$,

$$\Pr\left[(c, d_0, d_1) \leftarrow \mathcal{A}(1^\kappa) : \mathsf{Equiv.Decom}(c, d_0) = 0 \wedge \mathsf{Equiv.Decom}(c, d_1) = 1\right] \leq \mathsf{negl}(\kappa).$$

**Definition 2.5.** We sat that a scheme is equivocal if for all $b \in \{0, 1\}$ the statistical difference between the following two distributions is negligible in $\kappa$.

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

We observe that our security definitions do not include an explicit hiding property of a committed bit. This property is actually implied by our equivocal hiding and will not be needed in explicit form in our proofs of security for non malleable commitments.

# 3 Computation Enabled CCA Commitments

We now define what we describe as "computation enabled" CCA secure commitments. Intuitively, these will be tagged commitments where a commitment to message $m$ under tag $\mathsf{tag}$ and randomness $r$ is created as $\mathsf{CCA.Com}(\mathsf{tag}, m; r) \to \mathsf{com}$. The scheme will be statistically binding if for all $\mathsf{tag}_0, \mathsf{tag}_1, r_0, r_1$ and $m_0 \neq m_1$ we have that $\mathsf{CCA.Com}(\mathsf{tag}_0, m_0; r_0) \neq \mathsf{CCA.Com}(\mathsf{tag}_1, m_1; r_1)$.

Our hiding property follows along the lines of chosen commitment security definitions [CLP10] where an attacker gives a challenge tag $\mathsf{tag}^*$ along with messages $m_0, m_1$ and receives a challenge commitment $\mathsf{com}^*$ to either $m_0$ or $m_1$ from the experiment. The attacker's job is to guess the message that was committed to with the aid of 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 $\mathsf{tag} \neq \mathsf{tag}^*$. The traditional notion of non-malleability (as

seen in [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].

The primary difference in our definition is that we also allow the attacker to submit a randomized turing machine program $P$ at the beginning of the game. The challenger will run the program $P$ and output the result for the attacker at the beginning of the game. This added property will allow us to successfully apply recursion for tag amplification later in our scheme.

In addition, we require a recover from randomness property, which allows one to open the commitment given all the randomness used to generate said commitment.

### 3.1 Definition

A computation enabled CCA secure commitment is parameterized by a tag space of size $N = N(\kappa)$ where tags are in [1,N]. 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 $\kappa$, a tag $\mathsf{tag} \in [N]$, a message $m \in \{0,1\}^*$ 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 $m \in \{0,1\}^*$ or a reject symbol $\perp$.

$\mathsf{CCA.Recover}(\mathsf{com}, r) \to m \cup \perp$ is a deterministic algorithm which takes a commitment com and the randomness $r$ used to generate com and outputs the underlying message $m$ (or $\perp$ indicating no message was recovered).

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

#### Correctness

**Definition 3.1.** We say that our computation enabled CCA secure commitment scheme is perfectly correct if the following holds. $\forall m \in \{0,1\}^*$, $\mathsf{tag} \in [N]$ 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 computation enabled CCA secure commitment scheme is efficient if CCA.Com, CCA.Recover run in time $\mathsf{poly}(|m|, \kappa)$, while CCA.Val runs in time $\mathsf{poly}(|m|, 2^\kappa)$.

#### Recovery From Randomness

**Definition 3.3.** We say that our CCA secure commitment scheme can be recovered from randomness if the following holds. For all $m \in \{0,1\}^*$, $\mathsf{tag} \in [N]$, and $r$ we have that

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

**Security**

**Binding Game.** We first define a binding game between a challenger and an attacker. The game is parameterized by a security parameter $\kappa$.

1. The attacker sends a randomized and inputless Turing Machine algorithm $P$. The challenger runs the program on random coins and sends the output to the attacker. If the program takes more than $2^{2^\kappa}$ time to halt, the challenger halts the evaluation and outputs the empty string.[3]
2. The attacker outputs a commitment string $c$. Let $v = \mathsf{CCA.Val}(c)$.
3. We define the attacker's advantage in this game to be

$$\Pr\left[\exists r \text{ s.t. } (\mathsf{CCA.Recover}(c,r) \neq \perp) \wedge (\mathsf{CCA.Recover}(c,r) \neq v)\right] + \Pr\left[\nexists r \text{ s.t. } \mathsf{CCA.Recover}(c,r) = v\right]$$

**Weak Binding Game.** We now define a weak binding game between a challenger and an attacker. The game is identical to the binding game above, except the way the adversary's advantage is defined. The game is parameterized by a security parameter $\kappa$.

1. The attacker sends a randomized and inputless Turing Machine algorithm $P$. The challenger runs the program on random coins and sends the output to the attacker. If the program takes more than $2^{2^\kappa}$ time to halt, the challenger halts the evaluation and outputs the empty string.[4]
2. The attacker outputs a commitment string $c$. Let $v = \mathsf{CCA.Val}(c)$.
3. We define the attacker's advantage in this game to be

$$\Pr\left[\exists r \text{ s.t. } (\mathsf{CCA.Recover}(c,r) \neq \perp) \wedge (\mathsf{CCA.Recover}(c,r) \neq v)\right]$$

**CCA Hiding Game.** We also define a CCA hiding game between a challenger and an attacker. The game is parameterized by a security parameter $\kappa$.

1. The attacker sends a randomized and inputless Turing Machine algorithm $P$. The challenger runs the program on random coins and sends the output to the attacker. If the program takes more than $2^{2^\kappa}$ time to halt, the challenger halts the evaluation and outputs the empty string.[5]
2. The attacker sends a "challenge tag" $\mathsf{tag}^* \in [N]$.
3. The attacker makes repeated commitment queries $\mathsf{com}$. If $\mathsf{com.tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as

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

4. For some $w$, the attacker sends two messages $m_0, m_1 \in \{0,1\}^w$.

[3] The choice of $2^{2^\kappa}$ is somewhat arbitrary as the condition is in place so that the game is well defined on all $P$.
[4] The choice of $2^{2^\kappa}$ is somewhat arbitrary as the condition is in place so that the game is well defined on all $P$.
[5] The choice of $2^{2^\kappa}$ is somewhat arbitrary as the condition is in place so that the game is well defined on all $P$.

5. The challenger flips a coin $b \in \{0,1\}$ and sends $\mathsf{com}^* = \mathsf{CCA.Com}(\mathsf{tag}^*, m_b; r)$ for randomly chosen $r$.

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

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

7. 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 3.4.** An attack algorithm $\mathcal{A}$ is said to be $e$-conforming for some real value $e > 0$ if:

1. $\mathcal{A}$ is a (randomized) uniform algorithm.
2. $\mathcal{A}$ runs in polynomial time.
3. The program $P$ output by $\mathcal{A}$ in Step 1 of the binding or the CCA game will always terminate in time $p(2^{\kappa^e})$ time and output at most $q(\kappa)$ bits for some polynomial functions $p, q$ (For all possible random tapes given to the program $P$).

**Definition 3.5.** A computation enabled CCA secure commitment scheme scheme given by algorithms $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be $e$-computation enabled CCA secure w.r.t. over-extraction if for any $e$-conforming adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathcal{A}$'s advantage in the weak binding and the CCA hiding game is $\mathsf{negl}(\kappa)$.

**Definition 3.6.** A computation enabled CCA secure commitment scheme scheme given by algorithms $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be $e$-computation enabled CCA secure if for any $e$-conforming adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathcal{A}$'s advantage in the binding and the CCA hiding 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 3.7.** A computation enabled CCA secure commitment scheme scheme given by algorithms $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be "same tag" $e$-computation enabled CCA secure w.r.t. over-extraction if for any $e$-conforming 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 weak binding game and the CCA game is $\mathsf{negl}(\kappa)$.

**Definition 3.8.** A computation enabled CCA secure commitment scheme scheme given by algorithms $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be "same tag" $e$-computation enabled CCA secure if for any $e$-conforming 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 binding game and the CCA game is $\mathsf{negl}(\kappa)$.

### 3.2 Connecting to Standard Security

We now connect our computation enabled definition of security to the standard notion of chosen commitment security. In particular, the standard notion of chosen commitment security is simply the computation enabled above, but removing the first step of submitting a program $P$. We prove two straightforward lemmas. The first showing that any computation enabled CCA secure commitment scheme is a standard secure one against uniform attackers. The second is that any non-uniformly secure standard scheme satisfies $e$-computation enabled security for any constant $e \geq 0$.

**Definition 3.9.** A commitment scheme $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be CCA secure against uniform/non-uniform attackers if for any poly-time uniform/non-uniform adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathcal{A}$'s advantage in the binding game, as well as the CCA hiding game *with Step 1 removed* is $\mathsf{negl}(\kappa)$.

**Definition 3.10.** A commitment scheme $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is said to be "same tag" CCA secure (resp. w.r.t. over-extraction) against uniform/non-uniform attackers if for any poly-time uniform/non-uniform adversary $\mathcal{A}$ such that all commitment queries submitted by $\mathcal{A}$ are on the same tag, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathcal{A}$'s advantage in the binding game (resp. weak binding game) and the CCA hiding game *with Step 1 removed* is $\mathsf{negl}(\kappa)$.

**Claim 3.1.** If $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is an $e$-computation enabled CCA secure commitment scheme for some $e$ as per Definition 3.6, then it is also a scheme that achieves standard CCA security against uniform poly-time attackers as per Definition 3.9.

*Proof.* This follows from the fact that any uniform attacker $\mathcal{A}$ in the standard security game with advantage $\epsilon(\kappa) = \epsilon$ immediately implies an $e$-conforming attacker $\mathcal{A}'$ with the same advantage where $\mathcal{A}'$ outputs a program $P$ that immediately halts and then runs $\mathcal{A}$. □

**Claim 3.2.** If $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ achieves standard CCA security against *non-uniform* poly-time attackers as per Definition 3.9, then it is an $e$-computation enabled CCA secure commitment scheme for any $e$ as per Definition 3.6.

*Proof.* Suppose $\mathcal{A}$ is an $e$-conforming attacker for some $e$ with some advantage $\epsilon = \epsilon(\kappa)$. Then our non-uniform attacker $\mathcal{A}'$ can fix the random coins of $\mathcal{A}$ and to maximize its probability of success. Since now $\mathcal{A}$ is deterministic save for randomness produced by the challenger in step 5, this deterministically fixes the $P$ $\mathcal{A}$ sends, so $\mathcal{A}'$ can fix the coins of $P$ to maximize success. Thus, $\mathcal{A}'$ can simulate $\mathcal{A}$ given the above aforementioned random coins of $\mathcal{A}$ and the output of $P$, both of which are poly-bounded by the fact that $\mathcal{A}$ is $e$-conforming. Since all non-challenger randomness was non-uniformly fixed to maximize success, $\mathcal{A}'$ has at least advantage $\epsilon$ as well. By our definition of standard security hiding, the advantage of $\mathcal{A}'$ must be negligible, so $\mathcal{A}$'s advantage must be as well. □

We remark that the above statements are also true for "same tag" conforming adversaries and for security w.r.t. over-extraction.

# 4 Tag Amplification

In this section we show a process from amplifying a computation enabled CCA commitment scheme for $N' = 4N$ tags to a scheme with $2^N$ tags. The amplification process imposes an overhead that is polynomial in $N$ and the size/time of the original commitment scheme. Thus it is important that $N$ be polynomially bounded in the security parameter.

Let $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ be an $e$-computation enabled CCA commitment scheme for $N'(\kappa) = N' = 4N$ tags. 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 $\ell(\kappa)$. In addition, for some constant $\delta \in (0,1)$[6] we assume a equivocal commitment without setup scheme $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ that is $T = 2^{\kappa^\delta}$ binding secure and statistically hiding.

We assume a hinting PRG scheme $(\mathsf{Setup}, \mathsf{Eval})$ that is $T = 2^{\kappa^\gamma}$ secure for some constant $\gamma \in (0,1)$ and has seed length $n(\kappa, |m|)$ (represented by $n$ for ease) and block output length of $\max(|m|, \ell \times N)$. For ease of notation we assume that $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) \in \{0,1\}^{|m|}$ and $\forall i \in [n]$, $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i) \in \{0,1\}^{\ell \cdot N}$.

Our transformation will produce three algorithms, $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ which we prove $e'$-computation enabled where we require $e' = e \cdot \delta \geq 1$. We will also present a fourth algorithm CCA.ValAlt, which is only used in the proof. The algorithms will make use of the auxiliary subroutines CCA.Find and CCA.Check described below. $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}) \to m \cup \perp$ is a deterministic inefficient algorithm that takes in a tag $\mathsf{tag}^*$ and a commitment com and outputs either a message $m \in \{0,1\}^*$ 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.

$\mathsf{CCA.Find}(x', \mathsf{com})$

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

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

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

- For each $i \in [n]$
  1. Let $\tilde{y}_i = \mathsf{Small.Val}(c_{x',i,0})$
  2. Set $\tilde{z}_i = \mathsf{Equiv.Decom}(\sigma_i, \tilde{y}_i)$. 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 1: Routine CCA.Find

We now describe our transformation.

Transformation $\mathsf{Amplify}(\mathsf{Small} = (\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover}), \mathsf{HPRG}, \mathsf{Equiv}, e') \to \mathsf{NM} = (\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ :

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

[6] The constant $\delta$ must be less than 1 in order to meet the requirement that the Equiv.Equivocate algorithm runs in time polynomial in $2^\kappa$.

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

**Inputs:** Seed candidate $\tilde{s} = \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$

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

**Output:** $\{0,1\}$

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

Figure 2: Routine CCA.Check

1. Compute $\kappa' = \kappa^{\frac{e'}{\delta}} = \kappa^e$. Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.[7]
2. Sample $(\mathsf{HPRG.pp}, 1^n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(|m|, N\cdot\ell)})$.
3. Sample $s = s_1 \ldots s_n \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
4. For all $i \in [n]$ run $\mathsf{Equiv.Com}(1^{\kappa'}, s_i) \to (\sigma_i, y_i)$.
5. Let $r_{x,i}, \tilde{r}_{x,i} \in \{0,1\}^\ell$ be defined as follows:
6. 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 $(\tilde{r}_{1,i}, \tilde{r}_{2,i}, \ldots, \tilde{r}_{N,i}) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$
7. Compute $c = m \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$
8. 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. $c_{x,i,1} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}_x, 1), \mathsf{msg} = y_i; r_{x,i})$
9. Output $\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]}\right)$ as the commitment. All of the randomness is used as the decommitment string.

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

1. Set $\tilde{s} = \mathsf{CCA.Find}(1, \mathsf{com})$.

[7] $\delta$ and $\gamma$ are known from the security guarantees of Equiv, HPRG respectively.

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

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

1. If $\mathsf{com.tag} = \mathsf{tag}^*$, output $\perp$.
2. Let $x^*$ be the smallest index where the bits of $\mathsf{tag}^*, \mathsf{tag}$ differ.
3. Set $\tilde{s} = \mathsf{CCA.Find}(x^*, \mathsf{com})$.
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. Parse $\mathsf{com} = \Big(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]}\Big)$.
2. Parse $r = (s, \{(r_{x,i,b})\}_{x\in[N],i\in[n],b\in\{0,1\}})$.
3. For each $i \in [n]$:
   (a) Let $\widetilde{y}_i = \mathsf{Small.Recover}(c_{1,i,0}, r_{1,i,0})$.
   (b) Set $\tilde{z}_i = \mathsf{Equiv.Decom}(\sigma_i, \tilde{y}_i)$. If $\tilde{y}_i = \perp$ or $\tilde{z}_i = \perp$, set $\tilde{s}_i = 1$. Else, set $\tilde{s}_i = \tilde{z}_i$.
4. If $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 0$, output $\perp$.
5. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$

**Efficiency**

**Claim 4.1.** If $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ is an efficient and correct $e$-computation enabled commitment w.r.t. over-extraction with recovery from randomness that satisfies the CCA hiding game with tag space $N(\kappa) \in \mathsf{poly}(\kappa)$, $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ is an efficient equivocal commitment scheme as per Definition 2.3, $e, e'$ are constants, then we have that $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is an efficient and correct $e'$-computation enabled CCA commitment scheme with recovery from randomness, and $\mathsf{CCA.ValAlt}$ runs in time $\mathsf{poly}(|m|, 2^\kappa)$.

*Proof.* $\mathsf{CCA.Com}$ calls $\mathsf{Small.Com}$ $2 \cdot n \cdot N$ times on the output of $\mathsf{Equiv.Com}(1^{\kappa^e}, \cdot)$ in addition to some other poly-time computation. By Definition 3.2, $\mathsf{Small.Com}$ is $\mathsf{poly}(|m|, \kappa)$. Since $\mathsf{Equiv.Com}$ runs in time $\mathsf{poly}(\kappa')$ by Definition 2.3, this bounds the $|m|$ from the input scheme with $\mathsf{poly}(\kappa^e) \in \mathsf{poly}(\kappa)$ as $e$ is a constant. 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 $\kappa$. $\mathsf{CCA.Val}$ and $\mathsf{CCA.ValAlt}$ both call $\mathsf{Small.Val}$ exactly once, in addition to some efficient computation, so must also be $\mathsf{poly}(|m|, 2^\kappa)$ as well. $\mathsf{CCA.Recover}$ 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. □

**Correctness**

**Claim 4.2.** If $(\mathsf{Small.Com}, \mathsf{Small.Val}, \mathsf{Small.Recover})$ is a correct computation enabled CCA commitment scheme as per Definition 3.1 and $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom})$ is a correct equivocal commitment scheme as per Definition 2.2, then $(\mathsf{CCA.Com}, \mathsf{CCA.Decom}, \mathsf{CCA.Val})$ is a correct computation enabled CCA commitment scheme.

*Proof.* 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, $\forall i \in [n]$,

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

Finally, the hinting PRG 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 therefore observe that the scheme is correct. □

**Recovery from Randomness** The recovery from randomness property follows from the correctness of CCA.Check, which follows because by randomness recovery of the base scheme, for all $i \in [n], x \in [N], b \in \{0,1\}$,

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

Moreover, by correctness of the equivocal scheme, $\forall i \in [n]$,

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

### 4.1 Proof of Security

We now prove security by showing that our transformation leads to an $e' = e \cdot \delta$-computation enabled CCA commitment scheme, assuming the base scheme is an $e$-computation enabled CCA commitment w.r.t. over-extraction. The rest of this section is devoted to proving the following theorem.

**Theorem 4.1.** Let $(\mathsf{Small.Com}, \mathsf{Small.Val})$ be an $e$-computation enabled CCA commitment scheme w.r.t. over-extraction satisfying Definition 3.5, for $N'(\kappa) = N' = 4N \in \mathsf{poly}(\kappa)$ tags, $(\mathsf{Setup}, \mathsf{Eval})$ be a hinting PRG that is $T = 2^{\kappa^\gamma}$ secure where $\gamma \in (0,1)$, and $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ be an equivocal commitment without setup scheme that is $T = 2^{\kappa^\delta}$ binding secure and statistically hiding for some constant $\delta \in (0,1)$. Then the above commitment scheme $(\mathsf{CCA.Com}, \mathsf{CCA.Val})$ is an $e' = e \cdot \delta$-computation enabled CCA commitment scheme for $2^N$ tags satisfying Definition 3.6, if $e' \geq 1$.

**Proof of Binding.** First, we will prove that no $e'$-confirming adversary $\mathcal{A}$ will have advantage better than $\mathsf{negl}(\kappa)$ in the binding game, by relying on security of the equivocal commitment. Here, the attacker will be allowed to ask for a program $P$ that runs in time polynomial in $2^{\kappa^{e'}}$ where $e' = e \cdot \delta$. The equivocal commitment will use security parameter $\kappa' = \kappa^e$, i.e., it will be secure against attackers that run in time $\mathsf{poly}(2^{(\kappa')^\delta}) = \mathsf{poly}(2^{\kappa^{e\delta}}) = \mathsf{poly}(2^{\kappa^{e'}})$. Thus our reduction will be able to run $P$ by itself and still be a legitimate $2^{(\kappa')^\delta}$-time attacker.

Now suppose there is an adversary $\mathcal{A}$ with non-negligible advantage in the binding game. Then, there is a polynomial $p(\cdot)$ such that for com output by $\mathcal{A}$,

$$\Pr\left[\exists r \text{ s.t. } (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot) \wedge (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \mathsf{CCA.Val}(\mathsf{com}))\right] \geq \frac{1}{p(\kappa)}$$

or

$$\Pr\left[\nexists r \text{ s.t. } \mathsf{CCA.Recover}(\mathsf{com}, r) = \mathsf{CCA.Val}(\mathsf{com})\right] \geq \frac{1}{p(\kappa)}$$

We have the following claims that rule out this possibility.

**Claim 4.3.** Consider commitment $\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]})\right)$ output by $\mathcal{A}$ such that

$$\exists r \text{ s.t. } (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot) \wedge (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \mathsf{CCA.Val}(\mathsf{com}))\,.$$

Then

- Either there exists $i \in [n], r' \in \{0,1\}^*$ such that

  $$y_i^0 = \mathsf{Small.Val}(c_{1,i,0}) \wedge y_i^1 = \mathsf{Small.Recover}(c_{1,i,1}, r') \wedge \mathsf{Equiv.Decom}(\sigma_i, y_i^0) = 0 \wedge \mathsf{Equiv.Decom}(\sigma_i, y_i^1) = 1.$$

- Or $\exists r'$ such that $\mathsf{Small.Recover}(c_{1,i,0}, r') \neq \bot$ and $\mathsf{Small.Val}(c_{1,i,0}) \neq \mathsf{Small.Recover}(c_{1,i,0}, r')$.

*Proof.* CCA.Val and CCA.Recover differ only in their recovery of a candidate seed. Thus if they output different values, it implies that they recover two different seeds, $\tilde{s}^0$ and $\tilde{s}^1$ respectively, where $\tilde{s}^0 \neq \tilde{s}^1$.

Since $\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot$, we have that $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ cannot output $0$.

Let $i$ be any index where $\tilde{s}_i^0 \neq \tilde{s}_i^1$.

Let $\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{1,i,0}))$, $\tilde{z}_i^1 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{1,i,0}, r'))$, where $r' = r_{1,i,0}$ for $r$ parsed as $(\cdot, \{(r_{x,i,b})\}_{x\in[N],i\in[n],b\in\{0,1\}})$.

- Case 1: $\tilde{s}_i^0 = 0 \wedge \tilde{s}_i^1 = 1$.
  Note that $\tilde{s}_i^0$ is computed as a result of running $\mathsf{CCA.Find}(1, \mathsf{com})$. Thus,

  $$\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{1,i,0})) = 0$$

  (since if it was $\bot$ then $\tilde{s}_i^0$ would be set to 1).

  Moreover, since $\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot$, we have that $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com}) = 1$. Thus,

  $$\mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{1,i,1}, r_{1,i})) = \tilde{s}_i^1 = 1.$$

  where $(r_{1,i}, \ldots) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}^1, i)$.

- Case 2: $\tilde{s}_i^0 = 1 \wedge \tilde{s}_i^1 = 0$.
  Note that $\tilde{s}_i^1$ is computed as a result of running $\mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{1,i,0}, r'))$, where $r' = r_{1,i,0}$ for $r$ parsed as $(\cdot, \{(r_{x,i,b})\}_{x\in[N],i\in[n],b\in\{0,1\}})$. Thus, we get that

  $$(\mathsf{Small.Recover}(c_{1,i,0}, r') \neq \bot) \wedge (\tilde{z}_i^1 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{1,i,0}, r')) = 0)$$

  (if it was $\bot$ then $\tilde{s}_i^1$ would be set to 1).

  Moreover, if $\mathsf{Small.Val}(c_{1,i,0}) = \mathsf{Small.Recover}(c_{1,i,0}, r')$, $\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{1,i,0})) = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{1,i,0}, r')) = 0$. But this is impossible since $\tilde{s}_i^0 = 1$. This implies that $\mathsf{Small.Val}(c_{1,i,0}) \neq \mathsf{Small.Recover}(c_{1,i,0}, r')$. Thus, $\exists r'$ such that $\mathsf{Small.Recover}(c_{1,i,0}, r') \neq \bot$ and $\mathsf{Small.Val}(c_{1,i,0}) \neq \mathsf{Small.Recover}(c_{1,i,0}, r')$.

This proves the claim. □

Now, if there is a polynomial $p(\cdot)$ such that for com output by $\mathcal{A}$,

$$\Pr\left[\exists r \text{ s.t. } (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot) \wedge (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \mathsf{CCA.Val}(\mathsf{com}))\right] \geq \frac{1}{p(\kappa)}$$

then by the above claim, either

$$\Pr[\exists i, r' \text{ s.t. } \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{1,i,0})) = 0 \wedge \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{1,i,1}, r')) = 1] \geq \frac{1}{2p(\kappa)}$$

or

$$\Pr[\exists r' \text{ s.t. } \mathsf{Small.Recover}(c_{1,i,0}, r') \neq \bot, \mathsf{Small.Val}(c_{1,i,0}) \neq \mathsf{Small.Recover}(c_{1,i,0}, r')] \geq \frac{1}{2p(\kappa)}$$

Since the equivocal commitment is $2^{\kappa'^{\delta}} = 2^{\kappa^{e'}} > 2^{\kappa}$-secure, the former equation contradicts binding of the equivocal commitment (by finding $r'$ and running Small.Val in time at most $2^{\kappa}$). On the other hand the latter equation contradicts weak binding of the base commitment. Therefore, it only remains to prove that for com output by $\mathcal{A}$,

$$\Pr\left[\nexists r \text{ s.t. } \mathsf{CCA.Recover}(\mathsf{com}, r) = \mathsf{CCA.Val}(\mathsf{com})\right] = \mathsf{negl}(\kappa)$$

We will in fact prove the following stronger claim.

**Claim 4.4.** For every $\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]}\right)$, there exists $r$ such that

$$\mathsf{CCA.Recover}(\mathsf{com}, r) = \mathsf{CCA.Val}(\mathsf{com}).$$

*Proof.* First, we note that if $\mathsf{CCA.Val}(\mathsf{com}) = \bot$, the claim trivially follows. This is because for $r = \bot$, $\mathsf{CCA.Recover}(\mathsf{com}, r) = \bot$, proving the claim.

For the rest of this proof, we restrict ourselves to the case of $\mathsf{CCA.Val}(\mathsf{com}) = v \neq \bot$. This implies that there exists $\tilde{s}$ (output by $\mathsf{CCA.Find}(1, \mathsf{com})$) such that $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 1$. This implies that for $i \in [n], x \in [N]$, $\mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x,i,\tilde{s}_i}, r_{x,i})) = \tilde{s}_i$ and $c_{x,i,\tilde{s}_i} = \mathsf{Small.Com}(1^{\kappa}, (x, \mathsf{tag}_x, \tilde{s}_i), \mathsf{Small.Recover}(c_{x,i,\tilde{s}_i}, r_{x,i}); r_{x,i})$, where $(r_{1,i}, \ldots, r_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, i)$.

Set $\{r'_{x,i,b}\}_{i\in[n],x\in[N],b\in\{0,1\}}$ arbitrarily such that for $i \in [n], x \in [N]$, $r'_{x,i,\tilde{s}_i} = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, i)$, and for $i \in [n], x \in [N]$, $r'_{x,i,1-\tilde{s}_i}$ is set such that $\mathsf{CCA.Recover}(c_{x,i,1-s_i}, r'_{x,i,1-s_i}) = \bot$. Set $r' = (s', \{r'_{x,i,b}\}_{i\in[n],x\in[N],b\in\{0,1\}})$. Now, $\mathsf{CCA.Recover}(\mathsf{com}, r')$ computes $\tilde{y}_i = \mathsf{Small.Recover}(c_{1,i,0}, r_{1,i,0})$, $\tilde{z}_i = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{1,i,0}, r_{1,i,0}))$. For every $i$ where $\tilde{s}_i = 0$, by construction, we have that $\tilde{z}_i = 0$. Moreover, for every $i$ where $\tilde{s}_i = 1$, by construction $\tilde{y}_i = \bot$, which implies that $\tilde{z}_i = \bot$. This implies that the seed recovered by $\mathsf{CCA.Recover}$ matches $\tilde{s}$, which proves that $\mathsf{CCA.Recover}(\mathsf{com}, r') = \mathsf{CCA.Val}(\mathsf{com})$. □

This completes the proof of binding.

**Proof of CCA Hiding.** To prove security according to the CCA hiding game, we consider the following sequence of steps. In each proof step we will need to keep in mind that the attacker will be allowed to ask for a program $P$ that runs in time polynomial in $2^{\kappa^{e'}}$ where $e' = e \cdot \delta$. This will be satisfied in one of two ways. In the proof steps that rely on the hinting PRG security or the equivocal commitment without setup scheme we leverage the that that these are subexponentially secure primitives. For relying on security of the equivocal commitment without setup we use security parameter $\kappa' = \kappa^e$, it is secure against attackers that run in time polynomial in $2^{(\kappa')^\delta} = 2^{\kappa^{e\delta}} = 2^{\kappa^{e'}}$ time. Thus our reduction algorithm in these steps can satisfy the requirement by running $P$ itself and still be a legitimate $2^{(\kappa')^\delta}$ time attacker. For relying on security of the hinting PRG scheme, we use security parameter $\kappa'' = \kappa'^{\frac{1}{\gamma}}$, it is secure against attackers that run in time polynomial in $2^{\kappa'}$. Thus our reduction algorithm can run $P$ and the equivocate algorithm.

The second situation is when we rely on the security of the smaller tag space $e$-computation enabled scheme. In this case the reduction will need to be polynomial time so there is no way for it to directly run a program $P$ that takes $2^{\kappa^{e'}}$ time. However, in this case it can satisfy the requirement by creating a program $\tilde{P}$ and passing this onto the security game of the $e$-computation enabled challenger. The program $\tilde{P}$ will run $P$ as well as $n$ invocations of the $\mathsf{Equiv.Equivocate}$ algorithm.

We begin our formal analysis by defining a sequence of games. Then for each adjacent set of games we prove that the advantage of any $e' = e \cdot \delta$ conforming attacker $\mathcal{A}$ in the two games must be negligibly close.

**Game 0.** This is the original message hiding game between a challenger and an attacker for $e' = e \cdot \delta$ conforming attackers. The game is parameterized by a security parameter $\kappa$.

1. The attacker sends a randomized and inputless Turing Machine algorithm $P$. The challenger runs the program on random coins and sends the output to the attacker. If the program takes more than $2^{2^\kappa}$ time to halt, the outputs halts the evaluation and outputs the empty string.
2. The attacker sends a "challenge tag" $\mathsf{tag}^* \in \{0,1\}^N$.
3. **Pre Challenge Phase:** The attacker makes repeated queries commitments
$$\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, \left(\sigma_i, \left(c_{x,i,0}, c_{x,i,1}\right)_{x\in[N]}\right)_{i\in[n]}\right).$$
If $\mathsf{tag} = \mathsf{tag}^*$ the challenger responds with $\perp$. Otherwise it responds as
$$\mathsf{CCA.Val}(\mathsf{com}).$$
4. **Challenge Phase**
   (a) The attacker sends two messages $m_0^*, m_1^* \in \{0,1\}^w$
   (b) **Part 1:**
   - Compute $\kappa' = \kappa^e$.
   - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Sample $(\mathsf{HPRG.pp}^*, 1^n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(w, N\cdot\ell)})$.
   - Sample $s^* = s_1^* \ldots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.

- Let $r^*_{x,i}, \tilde{r}^*_{x,i} \in \{0,1\}^\ell$ be defined as follows:
- For $i \in [n]$
  i. Compute $(r^*_{1,i}, r^*_{2,i}, \ldots, r^*_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
  ii. Sample $(\tilde{r}^*_{1,i}, \tilde{r}^*_{2,i}, \ldots, \tilde{r}^*_{N,i}) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$
- For all $i \in [n]$ run $\mathsf{Equiv.Com}(1^{\kappa'}, s^*_i) \to (\sigma^*_i, y^*_i)$.

(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 $\mathsf{com}^* = \left(\mathsf{tag}^*, \mathsf{HPRG.pp}^*, c^*, (\sigma^*_i, \left(c^*_{x,i,0}, c^*_{x,i,1}\right)_{x\in[N]})_{i\in[n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

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

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

6. 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 uses $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com})$ to answer queries.

**Game 2.** In this game in **Part 1** the $(\sigma^*_i, y^*_i)$ are now generated from the Equiv.Equivocate algorithm instead of the Equiv.Com algorithm.

- Compute $\kappa' = \kappa^e$.
- Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
- Sample $(\mathsf{HPRG.pp}^*, 1^n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(w, N\cdot\ell)})$.
- Sample $s^* = s^*_1 \ldots s^*_n \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
- Let $r^*_{x,i}, \tilde{r}^*_{x,i} \in \{0,1\}^\ell$ be defined as follows:
- For $i \in [n]$
  1. Compute $(r^*_{1,i}, r^*_{2,i}, \ldots, r^*_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
  2. Sample $(\tilde{r}^*_{1,i}, \tilde{r}^*_{2,i}, \ldots, \tilde{r}^*_{N,i}) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$
- For all $i \in [n]$ run $\mathsf{Equiv.Equivocate}(1^{\kappa'}) \to (\sigma^*_i, y^*_{i,0}, y^*_{i,1})$.
- 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 $x \in [N], i \in [n]$ 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) $\underline{c^*_{x,i,1} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}^*_x, 1), y^*_{i,1}; \tilde{r}^*_{x,i})}$
  2. If $s^*_i = 1$
     (a) $\underline{c^*_{x,i,0} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}^*_x, 0), y^*_{i,0}; \tilde{r}^*_{x,i})}$
     (b) $c^*_{x,i,1} = \mathsf{Small.Com}(1^\kappa, (x, \mathsf{tag}^*_x, 1), y^*_{i,1}; r^*_{x,i})$
- Finally, it sends $\mathsf{com}^* = \left(\mathsf{tag}^*, \mathsf{HPRG.pp}^*, c^*, (\sigma^*_i, \left(c^*_{x,i,0}, c^*_{x,i,1}\right)_{x\in[N]})_{i\in[n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

**Game 4.** In all $r^*_{x,i}$ values are chosen uniformly at random (insted of choosing from $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$) and $c^*$ is also chosen uniformly at random (instead of choosing $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m^*_b$).

### 4.2 Analysis.

Next, we show by a sequence of lemmas that no $e' = e \cdot \delta$ adversary 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 $\mathsf{adv}^x_{\mathcal{A}}$ denote the quantity $\Pr[b' = b] - \frac{1}{2}$ in Game $x$.

**Lemma 4.1.** Assuming that the equivocal commitment is subexponentially binding secure for $T = 2^{\kappa^\delta}$ for $\delta \in (0,1)$ from Definition 2.4. For any $e'$-computation enabled adversary $\mathcal{A}$ where $e' \geq 1$, 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 $\kappa' = \kappa^e$.

*Proof.* Note that the two games differ in how we are answering pre and post challenge queries. In Game 0, we use $\mathsf{CCA.Val}$ and call the $\mathsf{CCA.Find}$ algorithm on tag $(1, \mathsf{tag}_1, 0)$. In Game 1, we use $\mathsf{CCA.ValAlt}$ and call the $\mathsf{CCA.Find}$ algorithm on a tag differing in the challenge tag.

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

$$|\mathsf{adv}^0_{\mathcal{A}} - \mathsf{adv}^1_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}.$$

We describe a uniform adversary $\mathcal{B}$ that runs in time $\mathsf{poly}(2^{(\kappa')^\delta}) = \mathsf{poly}(2^{\kappa^{e'}})$ and outputs $c, d_0, d_1$ such that both $\mathsf{Equiv.Decom}(c, d_0) = 0 \wedge \mathsf{Equiv.Decom}(c, d_1) = 1$ and hence breaks the binding security of the equivocal commitment.

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

1. $\mathcal{A}$ sends a randomized and inputless Turing Machine algorithm $P$. $\mathcal{B}$ runs the program on random coins and sends the output to the attacker.
2. $\mathcal{A}$ sends a "challenge tag" $\mathsf{tag}^* \in \{0,1\}^N$.
3. **Pre Challenge Phase:** $\mathcal{A}$ makes repeated queries on commitments

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

   If $\mathsf{tag} = \mathsf{tag}^*$, $\mathcal{B}$ responds with $\perp$. Otherwise it brute force opens the commitment scheme by running both $\mathsf{CCA.Val}, \mathsf{CCA.ValAlt}$.
   Let $x^* \in [N]$ be the smallest index where $\mathsf{tag}_{x^*} \neq \mathsf{tag}^*_{x^*}$, $\tilde{s}^0 = \mathsf{CCA.Find}(1, \mathsf{com})$, $\tilde{s}^1 = \mathsf{CCA.Find}(x^*, \mathsf{com})$ be the seeds computed when running $\mathsf{CCA.Val}$ and $\mathsf{CCA.ValAlt}$ respectively.

   - If $\mathsf{CCA.Val}(\mathsf{com}) = \mathsf{CCA.ValAlt}(\mathsf{com})$, return $\mathsf{CCA.Val}(\mathsf{com})$ to $\mathcal{A}$
   - Else, let $i^* \in [n]$ be the smallest index where $\tilde{s}^0_{i^*} \neq \tilde{s}^1_{i^*}$, output $(\perp, \perp, \perp)$ if no such $i^*$ exists,
     - If $\tilde{s}^0_{i^*} = 0$, output $(\sigma_{i^*}, \mathsf{Small.Val}(c_{1,i^*,0}), \mathsf{Small.Val}(c_{x^*,i^*,1}))$.
     - If $\tilde{s}^0_{i^*} = 1$, output $(\sigma_{i^*}, \mathsf{Small.Val}(c_{x^*,i^*,0}), \mathsf{Small.Val}(c_{1,i^*,1}))$.
4. Simulates **Challenge Phase** same as Game 0.
5. **Post Challenge Phase:** Exactly same as **Pre Challenge Phase**.
6. Output $(\perp, \perp, \perp)$.

Consider any commitment sent by $\mathcal{A}$ and denoted by

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

We will define a set $\mathsf{BAD}_{\mathsf{COM}}$ of "bad" commitments queries as follows:

$$\mathsf{com} \in \mathsf{BAD}_{\mathsf{COM}} \iff \mathsf{CCA.Val}(\mathsf{com}) \neq \mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}).$$

**Claim 4.5.** Let $x^* \in [N]$ be the smallest index where $\mathsf{tag}_{x^*} \neq \mathsf{tag}^*_{x^*}$, $\tilde{s}^0 = \mathsf{CCA.Find}(1, \mathsf{com})$, $\tilde{s}^1 = \mathsf{CCA.Find}(x^*, \mathsf{com})$ be the seeds computed when running $\mathsf{CCA.Val}$ and $\mathsf{CCA.ValAlt}$ respectively.

$$\begin{aligned}\mathsf{CCA.Val}(\mathsf{com}) \neq \mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}) \implies & \exists i \in [n], x^0 \in [N], x^1 \in [N], \\ & \tilde{s}^0_i \neq \tilde{s}^1_i \wedge \\ & y^0_i = \mathsf{Small.Val}(c_{x^0,i,0}) \wedge y^1_i = \mathsf{Small.Val}(c_{x^1,i,1}) \wedge \\ & \mathsf{Equiv.Decom}(\sigma_i, y^0_i) = 0 \wedge \mathsf{Equiv.Decom}(\sigma_i, y^1_i) = 1\end{aligned}$$

*Proof.* $\mathsf{CCA.Val}$ and $\mathsf{CCA.ValAlt}$ oracles differ only in their calls to $\mathsf{CCA.Find}$. Thus if they output different values, it implies that $\tilde{s}^0 \neq \tilde{s}^1$. Clearly both $\mathsf{CCA.Check}(\tilde{s}^0, \mathsf{com})$ and $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ cannot output $0$ (as then both $\mathsf{CCA.Val}$ and $\mathsf{CCA.ValAlt}$ output $\perp$).
Let $i$ be any index where $\tilde{s}^0_i \neq \tilde{s}^1_i$, $\tilde{z}^0_i = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{1,i,0}))$, $\tilde{z}^1_i = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x^*,i,0}))$.

- Case 1: $\tilde{s}_i^0 = 0 \wedge \tilde{s}_i^1 = 1$.
  When running $\mathsf{CCA.Find}(1, \mathsf{com})$, we get that $\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{1,i,0})) = 0$ (if it was $\perp$ the seed at this position would be set to 1).

  – Since $\tilde{s}_i^1 = 1$, we have that $\tilde{z}_i^1 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x^*,i,0})) = 1$ or $\perp$. By weak binding of the base scheme, $\mathsf{CCA.Check}(\tilde{s}^0, \mathsf{com})$ outputs 0 as the check $2(c)$ in Figure 2 fails when checking $c_{x^*,i,0}$ as the output of $\mathsf{Equiv.Decom}(\sigma_i, \mathsf{CCA.Recover}(c_{x^*,i,0}, \tilde{r}))$ cannot be 0 for any $\tilde{r}$. Thus $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ must output 1 (both cannot be zero). Thus,

  $$\forall x \in [N], \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x,i,1})) = 1.$$

  Picking $x^0 = 1, x^1 \in [N]$, the above claim is satisfied.

- Case 2: $\tilde{s}_i^0 = 1 \wedge \tilde{s}_i^1 = 0$.
  When running $\mathsf{CCA.Find}(x^*, \mathsf{com})$, we get that $\tilde{z}_i^1 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x^*,i,0})) = 0$ (if it was $\perp$ the seed at this position would be set to 1).

  – Since $\tilde{s}_i^0 = 1$, we have that $\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{1,i,0})) = 1$ or $\perp$. By weak binding of the base scheme, $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ outputs 0 as the check $2(c)$ in Figure 2 fails when checking $c_{1,i,0}$ as the output of $\mathsf{Equiv.Decom}(\sigma_i, \mathsf{CCA.Recover}(c_{1,i,0}, \tilde{r}))$ cannot be 0 for any $\tilde{r}$. Thus $\mathsf{CCA.Check}(\tilde{s}^0, \mathsf{com})$ must output 1 (both cannot be zero). Thus,

  $$\forall x \in [N], \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x,i,1})) = 1.$$

  Picking $x^0 = x^*, x^1 \in [N]$, the above claim is satisfied.

□

**Claim 4.6.** Let $\mathbb{S}$ denote the commitment queries made by $\mathcal{A}$. Then for infinitely many $\kappa \in \mathbb{N}$,

$$\Pr[(\mathbb{S} \cap \mathsf{BAD}_{\mathsf{COM}}) \neq \emptyset] \geq \frac{1}{p(\kappa)}.$$

*Proof.* The two games differ only in the way the commitment queries are made by $\mathcal{A}$. If $\mathcal{A}$ doesn't output bad commitment queries, a distinction between the two games cannot be made. Thus with probability greater than equal to the distinguishing probability between the two games, they must have output a bad commitment. □

**Claim 4.7.** $\mathcal{B}$ runs in time $\mathsf{poly}(2^{(\kappa')^\delta}) = \mathsf{poly}(2^{\kappa^{e'}})$ and for infinitely many $\kappa \in \mathbb{N}$,

$$\Pr\left[(c, d_0, d_1) \leftarrow \mathcal{B}(1^{\kappa'}) : \mathsf{Equiv.Decom}(c, d_0) = 0 \wedge \mathsf{Equiv.Decom}(c, d_1) = 1\right] \geq \frac{1}{p(\kappa)}.$$

*Proof.* Runtime of $\mathcal{B}$ is bounded by running the Turing Machine Algorithm $P$. Since $\mathcal{A}$ is an $e'$-computation enabled adversary. Runtime of $\mathcal{B}$ in step 1 is polynomially bounded in $2^{\kappa^{e'}}$. Running $\mathsf{CCA.Val}, \mathsf{CCA.ValAlt}$ takes time $2^\kappa$ during the pre challenge and post challenge query phases. Assuming we set out parameters such that $e' \geq 1$. We satisfy the claim.

Since $\mathcal{A}$ outputs a bad commitment from Claim 4.6, this implies from Claim 4.5 that $\exists i \in [n], x^0 \in [N], x^1 \in [N]$ such that we can break the equivocal commitment scheme. Focusing on the analysis of Claim 4.5, we can observe that $\mathcal{B}$ picks the smallest index such that $\tilde{s}_i^0 \neq \tilde{s}_i^1$ and $x^0, x^1$ correctly depending on the value of $\tilde{s}_i^0$. □

The proof of the lemma follows immediately by contradiction from the above claims. □

**Lemma 4.2.** Assuming that the equivocal commitment is statistically equivocal from Definition 2.5. 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 $\kappa' = \kappa^e$.

*Proof.* From Definition 2.5, we know that the statistical distance between $(\sigma_i^*, y_i^*)$ 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 $\mathsf{negl}(\kappa)$ as well. □

**Lemma 4.3.** Assuming that the base commitment scheme is $e$-computation enabled from Definition 3.6. For any $e'$-computation enabled adversary $\mathcal{A}$, 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 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 $\mathsf{Small.Com}$.

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

1. Begin Running $\mathcal{A}(1^\kappa)$.
   - $\mathcal{A}$ sends $\mathcal{B}_j$ a randomized inputless TM $P$
2. Construct TM $\tilde{P}$ as follows and send to challenger:
   (a) Code for **Part 1** of **Challenge Phase**
   - Compute $\kappa' = \kappa^e$.
   - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Sample $(\mathsf{HPRG.pp}^*, 1^n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(w, N \cdot \ell)})$.

- Sample $s^* = s_1^* \dots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
- 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 $(\tilde{r}^*_{1,i}, \tilde{r}^*_{2,i}, \dots, \tilde{r}^*_{N,i}) \xleftarrow{R} \{0,1\}^{N \cdot n \cdot \ell}$
- For all $i \in [n]$ run $\mathsf{Equiv.Equivocate}(1^{\kappa'}) \to (\sigma_i^*, y^*_{i,0}, y^*_{i,1})$.
- For all $i \in [n]$, set $y_i^* = y^*_{i,s_i^*}$.
- Set $\mathsf{output}_1 = \left(s^*, \mathsf{HPRG.pp}^*, \{r^*_{x,i}, \tilde{r}^*_{x,i}\}_{x\in[N], i\in[n]}, \{\sigma_i^*, y^*_{i,0}, y^*_{i,1}\}_{i\in[n]}\right)$

(b) Run Programme $P$ on a random tape and let its output be $\mathsf{output}_2$.

(c) Output $(\mathsf{output}_1, \mathsf{output}_2)$.

- Return $\mathsf{output}_2$ to $\mathcal{A}$

3. $\mathcal{A}$ sends a challenge $\mathsf{tag}^* \in \{0,1\}^N$ to $\mathcal{B}_j$
4. Let $c^*_{x',i',\bar{s^*_{i'}}}$ be the commitment changed in Game $2_j$ as described above.
5. Send challenge tag $(x', \mathsf{tag}^*_{x'}, \bar{s^*_{i'}})$ to challenger.
6. **Pre Challenge Phase:**
   - $\mathcal{A}$ makes $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \cdot)$ queries
   $$\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]})\right)$$
   to $\mathcal{B}_j$.
   - $\mathcal{B}_j$ answers by running $\mathsf{CCA.ValAlt}$. This can be done efficiently using $\mathcal{B}_j$'s own Pre Challenge oracle access to $\mathsf{Small.Val}$ and runs $\mathsf{CCA.Find}$ manually. Since $\mathsf{CCA.Find}$ is only run on an index $x^*$ such that $\mathsf{tag}^*_{x^*} \neq \mathsf{tag}_{x^*}$, it will never call $\mathsf{Small.Val}$ on $(x', \mathsf{tag}^*_{x'}, \bar{s^*_{i'}})$.
7. **Challenge Phase:**
   - Select a random bit $\beta$
   - $\mathcal{A}$ sends two messages $m_0, m_1$

   (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 **Phase 2** of **Challenge Phase** using the message $m_\beta$ and $\mathsf{output}_2$, with the exception that $c^*_{x',i',\bar{s^*_{i'}}}$ is computed as noted above and submit output to $\mathcal{A}$.
8. **Post Challenge Phase:** Proceeds exactly as Pre Challenge Phase.
   - Receive bit guess $\beta'$ from $\mathcal{A}$
9. If $\beta = \beta'$, output $0$. Otherwise, output $1$.

**Claim 4.8.** $\mathcal{B}_j$ is an $e$-conforming adversary

*Proof.* We need to verify two main properties, that $\mathcal{B}_j$ is PPT, and $\tilde{P}$ runs in $\mathsf{poly}(2^{\kappa^e})$ time and outputs $\mathsf{poly}(\kappa)$ bits. The former is to verify, as $\mathcal{A}$ only makes use of a polynomial number of efficient functions and oracle queries. To gauge the runtime of $\tilde{P}$, we can see the only inefficient steps it does is $n$ calls to $\mathsf{Equiv.Equivocate}(1^{\kappa'})$ and the running of $P$. We can see the former takes time $n \cdot \mathsf{poly}(2^{\kappa'})$, and the latter is run in time $\mathsf{poly}(2^{\kappa^{e'}}) \ll \mathsf{poly}(2^{\kappa^e})$ by $e'$ conformity of $\mathcal{A}$. Finally, we can see the output of $\tilde{P}$ is $\mathsf{poly}(\kappa)$ length as $\mathsf{output}_1$ is the result of all polynomial time algorithms and the $\mathsf{poly}(\kappa)$ bound on the output of $P$. □

**Claim 4.9.** The advantage of $\mathcal{B}_j$ in winning the $e$-computation enabled game for the base commitment scheme $\mathsf{Small.Com}$ from Definition 3.6 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

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

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

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

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,

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

□

Since $\mathcal{B}_j$ is an $e$-conforming adversary with non-negligible advantage. The proof of the lemma follows immediately by contradiction from the above claims.

□

**Lemma 4.4.** Assuming that the hinting PRG is subexponentially secure with $T = 2^{\kappa^\gamma}$ where $\gamma \in (0,1)$ from Definition 2.1. For any $e'$-computation enabled 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 $\kappa'' = \kappa'^{\frac{1}{\gamma}} = \kappa^{\frac{e}{\gamma}}$.

*Proof.* Let $\mathcal{A}$ be an adversary 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 $\mathsf{poly}(2^{\kappa^e})$ time algorithm $\mathcal{A}'$ which has advantage $\frac{1}{2p(\kappa)}$ in the hinting PRG Game as per Definition 2.1 where inputs were called on security parameter $\kappa''$.

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

1. Choose a random bit $a \in \{0,1\}$
2. Run $\mathcal{A}$
   (a) When $\mathcal{A}$ sends their TM $P$, run it and return the result.
   (b) Receive challenge tag $\mathsf{tag}^*$ from $\mathcal{A}$
   (c) **Pre Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com})$.
   (d) **Challenge Phase:**
   - Compute $\kappa' = \kappa^e$.
   - 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 $\mathsf{Equiv.Equivocate}(1^{\kappa'}) \to (\sigma^*_i, y^*_{i,0}, y^*_{i,1})$.
   - For all $i \in [n]$, set $y^*_i = y^*_{i,s^*_i}$.
   (e) **Part 2:**
   - Set $c^* = \underline{r^\beta_0} \oplus m^*_a$.
   - 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 $\mathsf{com}^* = \left(\mathsf{tag}^*, \mathsf{HPRG.pp}^*, c^*, (\sigma^*_i, \left(c^*_{x,i,0}, c^*_{x,i,1}\right)_{x\in[N]})_{i\in[n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.
   (f) **Post Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com})$.
   (g) Receive $a'$ from $\mathcal{A}$.
3. If $a' = a$, then output $\beta' = 1$. Otherwise output $\beta' = 0$.

**Claim 4.10.** $\mathcal{A}'$ runs in time $\mathsf{poly}(2^{\kappa^e})$

*Proof.* The majority of $\mathcal{A}'$ involves simulating running $\mathcal{A}$. Since $\mathcal{A}$ is $e'$-computation enabled we can run $P$ in time $\mathsf{poly}(2^{\kappa^{e'}})$. The Pre and Post Challenge queries will invoke a polynomial number of calls to $\mathsf{CCA.ValAlt}$, which require time $\mathsf{poly}(2^\kappa) \leq \mathsf{poly}(2^{\kappa^e})$. Finally, the challenge phase is invokes $n$ calls to $\mathsf{Equiv.Equivocate}$, which has runtime $\mathsf{poly}(2^{\kappa'})$ by Definition 2.3. Since $N$ is poly bounded, the rest of the computation in the challenge phase is efficient, which bounds the total runtime with $\mathsf{poly}(2^{\kappa'}) = \mathsf{poly}(2^{\kappa^e})$ □

**Claim 4.11.** If $\mathcal{A}$ has advantage $|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}$, $\mathcal{A}'$ has advantage in the HPRG game in Definition 2.1 $\geq \frac{1}{2p(\kappa)}$

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

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

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

$$\left(r_0^0 \xleftarrow{R} \{0,1\}^\ell, \left\{r_{i,b}^0 \xleftarrow{R} \{0,1\}^\ell\right\}_{i\in[n],b\in\{0,1\}}\right)$$

$\mathcal{A}$ 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 $\mathcal{A}'$ 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 $\mathcal{A}'$ runs in time $\mathsf{poly}(2^{\kappa^e})$, its advantage must be negligible by Definition 2.1, a contradiction, which concludes our proof. □

**Lemma 4.5.** For any adversary $\mathcal{A}$, $\mathsf{adv}_{\mathcal{A}}^4 = 0$.

*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 $\mathsf{adv}_{\mathcal{A}}^0 = \mathsf{negl}(\kappa)$. This completes the proof of the theorem.

# 5 Removing the Same Tag Restriction

In this section we show a process from transforming from a "same-tag" computation enabled commitment scheme for $N' = 2N$ tags to a scheme with $N$ tags, but allows the attacker to make a query on any number of different tags not equal to the challenge $\mathsf{tag}^*$. The actual construction and proof is similar to the that of Section 4. Intuitively, to commit to message with tag $\mathsf{tag} \in [N]$ in the new scheme, one uses all tags in $[N] \setminus \{\mathsf{tag}\}$ in the underlying same tag scheme $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$. Due to our redundancy checks we can open using any of these underlying tags with the exception of tag. In particular, if we think of our security game with $\mathsf{tag}^*$ as the challenge tag in the new scheme, then any commitment with tag $\mathsf{tag} \neq \mathsf{tag}^*$ can be opened using $\mathsf{tag}^*$ in the underlying scheme.

Since our transformation produces $2(N-1)$ commitments from the underlying same-tag scheme, it is important that $N$ be polynomially bounded in the security parameter. Looking ahead, this means that when starting with a same-tag scheme of say $\lg\lg\lg(\kappa)$ tag size one should first apply the same tag to many tag transformation. And then apply a sequence of transformations to amplify the tag size.

Let $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$ be an same-tag $e$-computation enabled CCA commitment scheme for $N'(\kappa) = N' = 2N$ tags. We will assume tags take identities of the form $(i, \Gamma) \in [N] \times \{0,1\}$ and that the $\mathsf{Same.Com}$ algorithm take in random coins of length $\ell(\kappa)$. In addition, for some constant $\delta \in (0,1)$ we assume a equivocal commitment without setup scheme $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ that is $T = 2^{\kappa^\delta}$ binding secure and statistically hiding.

We assume a hinting PRG scheme $(\mathsf{Setup}, \mathsf{Eval})$ that is $T = 2^{\kappa^\gamma}$ secure for some constant $\gamma \in (0,1)$ and has seed length $n(\kappa, |m|)$ (represented by $n$ for ease) and block output length of $\max(|m|, \ell \times N)$. For ease of notation we assume that $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) \in \{0,1\}^{|m|}$ and $\forall i \in [n]$, $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, i) \in \{0,1\}^{\ell \cdot N}$.

As in the previous section our description will consist of three algorithms which consist of the defined $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ which we prove $e'$-computation enabled where we require $e' = e \cdot \delta \geq 1$. We will also present a fourth algorithm CCA.ValAlt, which is only used in the proof. The algorithms will make use of the auxiliary subroutines CCA.Find and CCA.Check described below.

$\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}) \to m \cup \perp$. A special feature of this algorithm is that its valuation process will depend only on $\mathsf{tag}^*$ and actually be independent of com.tag other than checking com.tag $\neq \mathsf{tag}^*$. Moreover, it will only call valuation on a single tag underneath.

$\mathsf{CCA.Find}(x', \mathsf{com})$

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

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

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

- If $x' = \mathsf{tag}$, output $\perp$
- For each $i \in [n]$
  1. Let $\tilde{y}_i = \mathsf{Same.Val}(c_{x',i,0})$
  2. Set $\tilde{z}_i = \mathsf{Equiv.Decom}(\sigma_i, \tilde{y}_i)$. 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.Find

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

**Inputs:** Seed candidate $\tilde{s} = \tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$

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

**Output:** $\{0,1\}$

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

Figure 4: Routine CCA.Check

Transformation OneToMany($\mathsf{Same} = (\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$, HPRG, Equiv, $e'$) $\rightarrow$ $\mathsf{NM} = (\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ :

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

1. Compute $\kappa' = \kappa^{\frac{e'}{\delta}} = \kappa^e$. Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
2. Sample $(\mathsf{HPRG.pp}, 1^n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(|m|, N\cdot\ell)})$.
3. Sample $s = s_1 \dots s_n \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
4. For all $i \in [n]$ run $\mathsf{Equiv.Com}(1^{\kappa'}, s_i) \rightarrow (\sigma_i, y_i)$.
5. Let $r_{x,i}, \tilde{r}_{x,i} \in \{0,1\}^\ell$ be defined as follows:
6. 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 $(\tilde{r}_{1,i}, \tilde{r}_{2,i}, \dots, \tilde{r}_{N,i}) \xleftarrow{R} \{0,1\}^{N\cdot\ell}$
7. Compute $c = m \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$
8. 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 $\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]\setminus\{\mathsf{tag}\}})_{i\in[n]}\right)$ as the commitment. All of the randomness is used as the decommitment string.

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

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

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

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

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

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

3. Set $x' = (\mathsf{com.tag} \mod N)$.
4. For each $i \in [n]$:
   (a) Let $\widetilde{y}_i = \mathsf{Small.Recover}(c_{x',i,0}, r_{x',i,0})$.
   (b) Set $\tilde{z}_i = \mathsf{Equiv.Decom}(\sigma_i, \tilde{y}_i)$. If $\tilde{y}_i = \perp$ or $\tilde{z}_i = \perp$, set $\tilde{s}_i = 1$. Else, set $\tilde{s}_i = \tilde{z}_i$.
5. If $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 0$, output $\perp$.
6. Output $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0)$

**Remark 5.1.** The randomness $r_{\mathsf{tag},i}$ and $\tilde{r}_{\mathsf{tag},i}$ are not used for all $i$, but we generate it this way for notational simplicity.

**Remark 5.2.** The choice of $(\mathsf{com.tag} \mod N) + 1$ is somewhat arbitrary. We simply use the fact that this quantity is both always in $[1, N]$ and never equal to com.tag.

**Correctness and Efficiency.**

**Efficiency**

**Claim 5.1.** If $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$ is an efficient $e$-computation enabled CCA commitment scheme as per Definition 3.2 with tag space $N(\kappa) \in \mathsf{poly}(\kappa)$, $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ is an efficient equivocal commitment scheme as per Definition 2.3, $e, e'$ are constants then $(\mathsf{CCA.Com}, \mathsf{CCA.Val}, \mathsf{CCA.Recover})$ is an efficient $e'$-computation enabled CCA commitment scheme, and CCA.ValAlt runs in time $\mathsf{poly}(|m|, 2^\kappa)$

*Proof.* CCA.Com calls Same.Com $2 \cdot n \cdot (N-1)$ times on the output of Equiv.Com in addition to some other poly-time computation. By Definition 3.2, Same.Com is $\mathsf{poly}(|m|, \kappa)$. Since Equiv.Com runs in time $\mathsf{poly}(\kappa')$ by Definition 2.3, this bounds $|m|$ from the input schemewith $\mathsf{poly}(\kappa^e) \in \mathsf{poly}(\kappa)$ as $e$ a constant. 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 $\kappa$. CCA.Val and CCA.ValAlt both call Same.Val exactly once, in addition to some efficient computation, so must also be $\mathsf{poly}(|m|, 2^\kappa)$ as well. CCA.Recover does a single $\oplus$, and since com and $r$ are both bounded by $\mathsf{poly}(\kappa)$ by the runtime of CCA.Com, CCA.Recover runs in $\mathsf{poly}(|m|, \kappa)$ as well. □

**Correctness.**

**Claim 5.2.** If $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$ is a correct computation enabled CCA commitment scheme as per Definition 3.1 and $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom})$ is a correct equivocal commitment scheme as per Definition 2.2, then $(\mathsf{CCA.Com}, \mathsf{CCA.Decom}, \mathsf{CCA.Val})$ is a correct computation enabled CCA commitment scheme.

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

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

Notably, this is true for $x \in [N] = (\mathsf{com.tag} \mod N) + 1 \neq \mathsf{com.tag}$, which means, along with the correctness of equivocal scheme, $\forall i \in [n]$,

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

the CCA.Find will return the correct seed candidate. Using this and the fact that the hinting PRG on input $s$, $\forall i \in [n], x \in [N] \setminus \{\mathsf{tag}\}$ tells us it correctly sets the randomness along $c_{x,i,s_i}$ and $c \oplus \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, s, 0) = m$, meaning the scheme is correct. □

Below is an additional claim on the correctness of CCA.ValAlt.

**Claim 5.3.** CCA.ValAlt alternate decryption oracle only uses one tag in its oracle calls to Same.Val that is dependent on the challenge tag $\mathsf{tag}^*$.

*Proof.* We can see CCA.ValAlt only ever calls Same.Val with on commitments with tag $(\mathsf{tag}^*, 0)$. Since com.tag contains commitments under every tag except $(\mathsf{tag}, 0)$ and $(\mathsf{tag}, 1)$, CCA.ValAlt will only return $\perp$ when $\mathsf{tag}^* = \mathsf{tag}$, in which case the oracle should return $\perp$ anyway. □

**Recovery from Randomness.** The recovery from randomness property follows from the correctness of CCA.Check, which follows because by randomness recovery of the base scheme, for all $i \in [n], x \in [N], b \in \{0,1\}$,

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

Moreover, by correctness of the equivocal scheme, $\forall i \in [n]$,

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

## 5.1 Proof of Security

We now prove security by showing that our transformation leads to an $e' = e \cdot \delta$-computation enabled CCA commitment scheme. We do so in a sequence of security games.

**Theorem 5.1.** Let $(\mathsf{Same.Com}, \mathsf{Same.Val}, \mathsf{Same.Recover})$ be a same-tag $e$-computation enabled CCA commitment w.r.t. over-extraction for $N'(\kappa) = N' = 2N$ tags, $(\mathsf{Setup}, \mathsf{Eval})$ be a hinting PRG scheme that is $T = 2^{\kappa^\gamma}$ secure for $\gamma \in (0,1)$, and $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ be an equivocal commitment without setup scheme that is $T = 2^{\kappa^\delta}$ binding secure and statistically hiding for some constant $\delta \in (0,1)$. Then the above commitment scheme $(\mathsf{CCA.Com}, \mathsf{CCA.Val})$ is an $e' = e \cdot \delta$-computation enabled CCA commitment scheme (without any same tag restriction) for $N$ tags if $e' \geq 1$.

**Proof of Binding.** First, we will prove that no $e'$-confirming adversary $\mathcal{A}$ will have advantage better than $\mathsf{negl}(\kappa)$ in the binding game, by relying on security of the equivocal commitment. Here, the attacker will be allowed to ask for a program $P$ that runs in time polynomial in $2^{\kappa^{e'}}$ where $e' = e \cdot \delta$. The equivocal commitment will use security parameter $\kappa' = \kappa^e$, i.e., it will be secure against attackers that run in time $\mathsf{poly}(2^{(\kappa')^\delta}) = \mathsf{poly}(2^{\kappa^{e\delta}}) = \mathsf{poly}(2^{\kappa^{e'}})$. Thus our reduction will be able to run $P$ by itself and still be a legitimate $2^{(\kappa')^\delta}$-time attacker.

Now suppose there is an adversary $\mathcal{A}$ with non-negligible advantage in the binding game. Then, there is a polynomial $p(\cdot)$ such that for com output by $\mathcal{A}$,

$$\Pr\left[\exists r \text{ s.t. } (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \perp) \wedge (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \mathsf{CCA.Val}(\mathsf{com}))\right] \geq \frac{1}{p(\kappa)}$$

or

$$\Pr\left[\nexists r \text{ s.t. } \mathsf{CCA.Recover}(\mathsf{com}, r) = \mathsf{CCA.Val}(\mathsf{com})\right] \geq \frac{1}{p(\kappa)}$$

We have the following claims that rule out this possibility.

**Claim 5.4.** Consider commitment $\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]})\right)$ output by $\mathcal{A}$ such that

$$\exists r \text{ s.t. } (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot) \wedge (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \mathsf{CCA.Val}(\mathsf{com}))\,.$$

Then

- Either there exists $x' \in [N], i \in [n], r' \in \{0,1\}^*$ such that

  $$y_i^0 = \mathsf{Small.Val}(c_{x',i,0}) \wedge y_i^1 = \mathsf{Small.Recover}(c_{x',i,1}, r') \wedge \mathsf{Equiv.Decom}(\sigma_i, y_i^0) = 0 \wedge \mathsf{Equiv.Decom}(\sigma_i, y_i^1) = 1.$$

- Or $\exists r'$ such that $\mathsf{Small.Recover}(c_{x',i,0}, r') \neq \bot$ and $\mathsf{Small.Val}(c_{x',i,0}) \neq \mathsf{Small.Recover}(c_{x',i,0}, r')$.

*Proof.* CCA.Val and CCA.Recover differ only in their recovery of a candidate seed. Thus if they output different values, it implies that they recover two different seeds, $\tilde{s}^0$ and $\tilde{s}^1$ respectively, where $\tilde{s}^0 \neq \tilde{s}^1$.

Since $\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot$, we have that $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ cannot output $0$.

Let $i$ be any index where $\tilde{s}_i^0 \neq \tilde{s}_i^1$.

Let $x' = (\mathsf{com.tag} \mod N)$, $\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x',i,0}))$, $\tilde{z}_i^1 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x',i,0},$ where $r' = r_{x',i,0}$ for $r$ parsed as $(\cdot, \{(r_{x,i,b})\}_{x\in[N],i\in[n],b\in\{0,1\}})$.

- Case 1: $\tilde{s}_i^0 = 0 \wedge \tilde{s}_i^1 = 1$.
  Note that $\tilde{s}_i^0$ is computed as a result of running $\mathsf{CCA.Find}(x', \mathsf{com})$. Thus,

  $$\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x',i,0})) = 0$$

  (since if it was $\bot$ then $\tilde{s}_i^0$ would be set to 1).

  Moreover, since $\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot$, we have that $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com}) = 1$. Thus,

  $$\mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x',i,1}, r_{1,i})) = \tilde{s}_i^1 = 1.$$

  where $(r_{1,i}, \ldots) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}^1, i)$.

- Case 2: $\tilde{s}_i^0 = 1 \wedge \tilde{s}_i^1 = 0$.
  Note that $\tilde{s}_i^1$ is computed as a result of running $\mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x',i,0}, r'))$, where $r' = r_{x',i,0}$ for $r$ parsed as $(\cdot, \{(r_{x,i,b})\}_{x\in[N],i\in[n],b\in\{0,1\}})$. Thus, we get that

  $$(\mathsf{Small.Recover}(c_{x',i,0}, r') \neq \bot) \wedge (\tilde{z}_i^1 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x',i,0}, r')) = 0)$$

  (if it was $\bot$ then $\tilde{s}_i^1$ would be set to 1).

  Moreover, if $\mathsf{Small.Val}(c_{x',i,0}) = \mathsf{Small.Recover}(c_{x',i,0}, r')$, $\tilde{z}_i^0 = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x',i,0})) =$ $\mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x',i,0}, r')) = 0$. But this is impossible since $\tilde{s}_i^0 = 1$. This implies that $\mathsf{Small.Val}(c_{x',i,0}) \neq \mathsf{Small.Recover}(c_{x',i,0}, r')$. Thus, $\exists r'$ such that $\mathsf{Small.Recover}(c_{x',i,0}, r') \neq \bot$ and $\mathsf{Small.Val}(c_{x',i,0}) \neq \mathsf{Small.Recover}(c_{x',i,0}, r')$.

This proves the claim. □

Now, if there is a polynomial $p(\cdot)$ such that for com output by $\mathcal{A}$,

$$\Pr\left[\exists r \text{ s.t. } (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \bot) \wedge (\mathsf{CCA.Recover}(\mathsf{com}, r) \neq \mathsf{CCA.Val}(\mathsf{com}))\right] \geq \frac{1}{p(\kappa)}$$

then by the above claim, either

$$\Pr[\exists i, r' \text{ s.t. } \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Val}(c_{x',i,0})) = 0 \wedge \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x',i,1}, r')) = 1] \geq \frac{1}{2p(\kappa)}$$

or

$$\Pr[\exists r' \text{ s.t. } \mathsf{Small.Recover}(c_{x',i,0}, r') \neq \bot, \mathsf{Small.Val}(c_{x',i,0}) \neq \mathsf{Small.Recover}(c_{x',i,0}, r')] \geq \frac{1}{2p(\kappa)}$$

Since the equivocal commitment is $2^{\kappa'^{\delta}} = 2^{\kappa^{e'}} > 2^{\kappa}$-secure, the former equation contradicts binding of the equivocal commitment (by finding $r'$ and running Small.Val in time at most $2^{\kappa}$). On the other hand the latter equation contradicts weak binding of the base commitment. Therefore, it only remains to prove that for com output by $\mathcal{A}$,

$$\Pr\left[\nexists r \text{ s.t. } \mathsf{CCA.Recover}(\mathsf{com}, r) = \mathsf{CCA.Val}(\mathsf{com})\right] = \mathsf{negl}(\kappa)$$

We will in fact prove the following stronger claim.

**Claim 5.5.** For every $\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x\in[N]})_{i\in[n]}\right)$, there exists $r$ such that

$$\mathsf{CCA.Recover}(\mathsf{com}, r) = \mathsf{CCA.Val}(\mathsf{com}).$$

*Proof.* First, we note that if $\mathsf{CCA.Val}(\mathsf{com}) = \bot$, the claim trivially follows. This is because for $r = \bot$, $\mathsf{CCA.Recover}(\mathsf{com}, r) = \bot$, proving the claim.

For the rest of this proof, we restrict ourselves to the case of $\mathsf{CCA.Val}(\mathsf{com}) = v \neq \bot$. This implies that there exists $\tilde{s}$ (output by $\mathsf{CCA.Find}(1, \mathsf{com})$) such that $\mathsf{CCA.Check}(\tilde{s}, \mathsf{com}) = 1$. This implies that for $i \in [n], x \in [N]$, $\mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x,i,\tilde{s}_i}, r_{x,i})) = \tilde{s}_i$ and $c_{x,i,\tilde{s}_i} = \mathsf{Small.Com}(1^{\kappa}, (x, \mathsf{tag}_x, \tilde{s}_i), \mathsf{Small.Recover}(c_{x,i,\tilde{s}_i}, r_{x,i}); r_{x,i})$, where $(r_{1,i}, \ldots, r_{N,i}) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, i)$.

Set $\{r'_{x,i,b}\}_{i\in[n],x\in[N],b\in\{0,1\}}$ arbitrarily such that for $i \in [n], x \in [N]$, $r'_{x,i,\tilde{s}_i} = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}, \tilde{s}, i)$, and for $i \in [n], x \in [N]$, $r'_{x,i,1-\tilde{s}_i}$ is set such that $\mathsf{CCA.Recover}(c_{x,i,1-s_i}, r'_{x,i,1-s_i}) = \bot$. Set $r' = (s', \{r'_{x,i,b}\}_{i\in[n],x\in[N],b\in\{0,1\}})$. Now, $\mathsf{CCA.Recover}(\mathsf{com}, r')$ computes $\tilde{y}_i = \mathsf{Small.Recover}(c_{x',i,0}, r_{x',i,0})$, $\tilde{z}_i = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Small.Recover}(c_{x',i,0}, r_{x',i,0}))$. For every $i$ where $\tilde{s}_i = 0$, by construction, we have that $\tilde{z}_i = 0$. Moreover, for every $i$ where $\tilde{s}_i = 1$, by construction $\tilde{y}_i = \bot$, which implies that $\tilde{z}_i = \bot$. This implies that the seed recovered by $\mathsf{CCA.Recover}$ matches $\tilde{s}$, which proves that $\mathsf{CCA.Recover}(\mathsf{com}, r') = \mathsf{CCA.Val}(\mathsf{com})$. □

This completes the proof of binding.

**Proof of CCA Hiding.** The proof will proceed in a similar direction as in Section 4 where we again need to be mindful of the reduction algorithm fulfilling the request to run program $P$. The other aspect is that when the algorithm reduces to the underlying scheme that is secure only for "same-tag" requests, we must argue that all requests are indeed to the same tag. Otherwise, the proof proceeds largely as before.

We now begin our formal analysis by defining a sequence of games. Then for each adjacent set of games we prove that the advantage of any $e' = e \cdot \delta$ conforming attacker $\mathcal{A}$ in the two games must be negligibly close.

**Game 0.** This is the original message hiding game between a challenger and an attacker for $e' = e \cdot \delta$ conforming attackers. The game is parameterized by a security parameter $\kappa$.

1. The attacker sends a randomized and inputless Turing Machine algorithm $P$. The challenger runs the program on random coins and sends the output to the attacker. If the program takes more than $2^{2^\kappa}$ time to halt, the outputs halts the evaluation and outputs the empty string.
2. The attacker sends a "challenge tag" $\mathsf{tag}^* \in [N]$.
3. **Pre Challenge Phase:** The attacker makes repeated queries commitments

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

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

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

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

   (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. $c_{x,i,0} = \mathsf{Same.Com}(1^\kappa, (x, 0), \mathsf{msg} = y_i; r_{x,i})$

B. $c_{x,i,1} = \mathsf{Same.Com}(1^\kappa, (x,1), \mathsf{msg} = y_i; \tilde{r}_{x,i})$

ii. If $s_i = 1$

A. $c_{x,i,0} = \mathsf{Same.Com}(1^\kappa, (x,0), \mathsf{msg} = y_i; \tilde{r}_{x,i})$

B. $c_{x,i,1} = \mathsf{Same.Com}(1^\kappa, (x,1), \mathsf{msg} = y_i; r_{x,i})$

- Output $\mathsf{com} = \left(\mathsf{tag}, \mathsf{HPRG.pp}, c, (\sigma_i, (c_{x,i,0}, c_{x,i,1})_{x \in [N] \setminus \{\mathsf{tag}\}})_{i \in [n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

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

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

6. 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 uses $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com})$ to answer queries.

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

- Compute $\kappa' = \kappa^e$.
- Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
- Sample $(\mathsf{HPRG.pp}^*, 1^n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(w, N \cdot \ell)})$.
- Sample $s^* = s_1^* \ldots s_n^* \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
- Let $r_{x,i}^*, \tilde{r}_{x,i}^* \in \{0,1\}^\ell$ be defined as follows:
- For $i \in [n]$
  1. Compute $(r_{1,i}^*, r_{2,i}^*, \ldots, r_{N,i}^*) = \mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$
  2. Sample $(\tilde{r}_{1,i}^*, \tilde{r}_{2,i}^*, \ldots, \tilde{r}_{N,i}^*) \xleftarrow{R} \{0,1\}^{N \cdot \ell}$
- For all $i \in [n]$ run $\mathsf{Equiv.Equivocate}(1^{\kappa'}) \rightarrow (\sigma_i^*, y_{i,0}^*, y_{i,1}^*)$.
- 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 $x \in [N] \setminus \{\mathsf{tag}^*\}, i \in [n]$ 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 all $r^*_{x,i}$ values are chosen uniformly at random (insted of choosing from $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, i)$) and $c^*$ is also chosen uniformly at random (instead of choosing $\mathsf{HPRG.Eval}(\mathsf{HPRG.pp}^*, s^*, 0) \oplus m^*_b$).

## 5.2 Analysis.

Next, we show by a sequence of lemmas that no $e' = e \cdot \delta$ adversary 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 $\mathsf{adv}^x_{\mathcal{A}}$ denote the quantity $\Pr[b' = b] - \frac{1}{2}$ in Game $x$.

**Lemma 5.1.** Assuming that the equivocal commitment is subexponentially binding secure for $T = 2^{\kappa^\delta}$ for $\delta \in (0,1)$ from Definition 2.4. For any $e'$-computation enabled adversary $\mathcal{A}$ where $e' \geq 1$, 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 $\kappa' = \kappa^e$.

*Proof.* Note that the two games differ in how we are answering pre and post challenge queries. In Game $0$, we use $\mathsf{CCA.Val}$ and call the $\mathsf{CCA.Find}$ algorithm on tag $(1, \mathsf{tag}_1, 0)$. In Game $1$, we use $\mathsf{CCA.ValAlt}$ and call the $\mathsf{CCA.Find}$ algorithm on a tag differing in the challenge tag.

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

$$|\mathsf{adv}^0_{\mathcal{A}} - \mathsf{adv}^1_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}.$$

We describe a uniform adversary $\mathcal{B}$ that runs in time $\mathsf{poly}(2^{(\kappa')^\delta}) = \mathsf{poly}(2^{\kappa^{e'}})$ and outputs $c, d_0, d_1$ such that both $\mathsf{Equiv.Decom}(c, d_0) = 0 \wedge \mathsf{Equiv.Decom}(c, d_1) = 1$ and hence breaks the binding security of the equivocal commitment.

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

1. $\mathcal{A}$ sends a randomized and inputless Turing Machine algorithm $P$. $\mathcal{B}$ runs the program on random coins and sends the output to the attacker.
2. $\mathcal{A}$ sends a "challenge tag" $\mathsf{tag}^* \in [N]$.
3. **Pre Challenge Phase:** $\mathcal{A}$ makes repeated queries on commitments

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

If $\mathsf{tag} = \mathsf{tag}^*$, $\mathcal{B}$ responds with $\perp$. Otherwise it brute force opens the commitment scheme by running both $\mathsf{CCA.Val}, \mathsf{CCA.ValAlt}$.

- Let $\tilde{s}^0 = \mathsf{CCA.Find}((\mathsf{tag} \mod N) + 1, \mathsf{com})$

- Let $\tilde{s}^1 = \mathsf{CCA.Find}(\mathsf{tag}^*, \mathsf{com})$
- If $\mathsf{CCA.Val}(\mathsf{com}) = \mathsf{CCA.ValAlt}(\mathsf{com})$, return $\mathsf{CCA.Val}(\mathsf{com})$ to $\mathcal{A}$
- Else, let $i^* \in [n]$ be the smallest index where $\tilde{s}^0_{i^*} \neq \tilde{s}^1_{i^*}$, output $(\perp, \perp, \perp)$ if no such $i^*$ exists,
  - If $\tilde{s}^0_{i^*} = 0$, output $(\sigma_{i^*}, \mathsf{Same.Val}(c_{1,i^*,0}), \mathsf{Same.Val}(c_{\mathsf{tag}^*,i^*,1}))$.
  - If $\tilde{s}^0_{i^*} = 1$, output $(\sigma_{i^*}, \mathsf{Same.Val}(c_{\mathsf{tag}^*,i^*,0}), \mathsf{Same.Val}(c_{1,i^*,1}))$.

4. Simulates **Challenge Phase** same as Game 0.
5. **Post Challenge Phase:** Exactly same as **Pre Challenge Phase**.
6. Output $(\perp, \perp, \perp)$.

Consider any commitment sent by $\mathcal{A}$ and denoted by

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

. We will define a set $\mathsf{BAD}_{\mathsf{COM}}$ of "bad" commitments queries as follows:

$$\mathsf{com} \in \mathsf{BAD}_{\mathsf{COM}} \iff \mathsf{CCA.Val}(\mathsf{com}) \neq \mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}).$$

**Claim 5.6.** Let $\tilde{s}^0 = \mathsf{CCA.Find}((\mathsf{tag} \mod N) + 1, \mathsf{com})$, $\tilde{s}^1 = \mathsf{CCA.Find}(\mathsf{tag}^*, \mathsf{com})$ be the seeds computed when running CCA.Val and CCA.ValAlt respectively.

$$\begin{aligned}\mathsf{CCA.Val}(\mathsf{com}) \neq \mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com}) \implies & \exists i \in [n], x^0 \in [N], x^1 \in [N], \\ & \tilde{s}^0_i \neq \tilde{s}^1_i \wedge \\ & y^0_i = \mathsf{Same.Val}(c_{x^0,i,0}) \wedge y^1_i = \mathsf{Same.Val}(c_{x^1,i,1}) \wedge \\ & \mathsf{Equiv.Decom}(\sigma_i, y^0_i) = 0 \wedge \mathsf{Equiv.Decom}(\sigma_i, y^1_i) = 1\end{aligned}$$

*Proof.* CCA.Val and CCA.ValAlt oracles differ only in their calls to CCA.Find. Thus if they output different values, it implies that $\tilde{s}^0 \neq \tilde{s}^1$. Clearly both $\mathsf{CCA.Check}(\tilde{s}^0, \mathsf{com})$ and $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ cannot output 0 (as then both CCA.Val and CCA.ValAlt output $\perp$).
Let $i$ be any index where $\tilde{s}^0_i \neq \tilde{s}^1_i$,

$$\tilde{z}^0_i = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Same.Val}(c_{(\mathsf{tag} \mod N)+1,i,0})),$$

$$\tilde{z}^1_i = \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Same.Val}(c_{\mathsf{tag}^*,i,0})).$$

- Case 1: $\tilde{s}^0_i = 0 \wedge \tilde{s}^1_i = 1$.
  When running $\mathsf{CCA.Find}((\mathsf{tag} \mod N) + 1, \mathsf{com})$, we get that $\tilde{z}^0_i = 0$ (if it was $\perp$ the seed at this position would be set to 1).
  - If $\tilde{z}^1_i = 1$ or $\perp$. $\mathsf{CCA.Check}(\tilde{s}^0, \mathsf{com})$ outputs 0 as the check $2(c)$ in figure (4) fails when checking $c_{\mathsf{tag}^*,i,0}$ as the output is either 1 or $\perp$. Thus $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ must output 1 (both cannot be zero). Thus,

    $$\forall x \in [N] \setminus \{\mathsf{tag}\}, \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Same.Val}(c_{x,i,1})) = 1.$$

    Picking $x^0 = (\mathsf{tag} \mod N + 1), x^1 \in [N] \setminus \{\mathsf{tag}\}$, the above claim is satisfied.

- Case 2: $\tilde{s}_i^0 = 1 \wedge \tilde{s}_i^1 = 0$.
  When running $\mathsf{CCA.Find}(\mathsf{tag}^*, \mathsf{com})$, we get that $\tilde{z}_i^1 = 0$ (if it was $\perp$ the seed at this position would be set to 1).

  – If $\tilde{z}_i^0 = 1$ or $\perp$. $\mathsf{CCA.Check}(\tilde{s}^1, \mathsf{com})$ outputs 0 as the check $2(c)$ in figure (4) fails when checking $c_{(\mathsf{tag} \mod N)+1,i,0}$ as the output is either 1 or $\perp$. Thus $\mathsf{CCA.Check}(\tilde{s}^0, \mathsf{com})$ must output 1 (both cannot be zero). Thus,

  $$\forall x \in [N] \setminus \{\mathsf{tag}\}, \mathsf{Equiv.Decom}(\sigma_i, \mathsf{Same.Val}(c_{x,i,1})) = 1.$$

  Picking $x^0 \in \mathsf{tag}^*, x^1 \in [N] \setminus \{\mathsf{tag}\}$, the above claim is satisfied.

□

**Claim 5.7.** Let $\mathbb{S}$ denote the commitment queries made by $\mathcal{A}$. Then for infinitely many $\kappa \in \mathbb{N}$,

$$\Pr[(\mathbb{S} \cap \mathsf{BAD}_{\mathsf{COM}}) \neq \emptyset] \geq \frac{1}{p(\kappa)}.$$

*Proof.* The two games differ only in the way the commitment queries are made by $\mathcal{A}$. If $\mathcal{A}$ doesn't output bad commitment queries, a distinction between the two games cannot be made. Thus with probability greater than equal to the distinguishing probability between the two games, they must have output a bad commitment. □

**Claim 5.8.** $\mathcal{B}$ runs in time $\mathsf{poly}(2^{(\kappa')^\delta}) = \mathsf{poly}(2^{\kappa^{e'}})$ and for infinitely many $\kappa \in \mathbb{N}$,

$$\Pr\left[(c, d_0, d_1) \leftarrow \mathcal{B}(1^{\kappa'}) : \mathsf{Equiv.Decom}(c, d_0) = 0 \wedge \mathsf{Equiv.Decom}(c, d_1) = 1\right] \geq \frac{1}{p(\kappa)}.$$

*Proof.* Runtime of $\mathcal{B}$ is bounded by running the Turing Machine Algorithm $P$. Since $\mathcal{A}$ is an $e'$-computation enabled adversary. Runtime of $\mathcal{B}$ in step 1 is polynomially bounded in $2^{\kappa^{e'}}$. Running $\mathsf{CCA.Val}, \mathsf{CCA.ValAlt}$ takes time $2^\kappa$ during the pre challenge and post challenge query phases. Assuming we set out parameters such that $e' \geq 1$. We satisfy the claim.

Since $\mathcal{A}$ outputs a bad commitment from Claim 5.7, this implies from Claim 5.6 that $\exists i \in [n], x^0 \in [N], x^1 \in [N]$ such that we can break the equivocal commitment scheme. Focusing on the analysis of Claim 5.6, we can observe that $\mathcal{B}$ picks the smallest index such that $\tilde{s}_i^0 \neq \tilde{s}_i^1$ and $x^0, x^1$ correctly depending on the value of $\tilde{s}_i^0$. □

The proof of the lemma follows immediately by contradiction from the above claims. □

**Lemma 5.2.** Assuming that the equivocal commitment is statistically equivocal from Definition 2.5. 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 $\kappa' = \kappa^e$.

*Proof.* From Definition 2.5, we know that the statistical distance between $(\sigma_i^*, y_i^*)$ 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 $\mathsf{negl}(\kappa)$ as well. □

**Lemma 5.3.** Assuming that the base commitment scheme is $e$-computation enabled from Definition 3.6. For any $e'$-computation enabled adversary $\mathcal{A}$, 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 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 $c^*_{x',i',\bar{s^*_i}}$[8] is generated from

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

to

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

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

1. Begin Running $\mathcal{A}(1^\kappa)$.
   - $\mathcal{A}$ sends $\mathcal{B}_j$ a randomized inputless TM $P$
2. Construct TM $\tilde{P}$ as follows and send to challenger:
   (a) Code for **Part 1** of **Challenge Phase**
   - Compute $\kappa' = \kappa^e$.
   - Compute $\kappa'' = \kappa'^{\frac{1}{\gamma}}$.
   - Sample $(\mathsf{HPRG.pp}^*, 1^n) \leftarrow \mathsf{HPRG.Setup}(\kappa'', 1^{\max(w, N \cdot \ell)})$.
   - Sample $s^* = s^*_1 \dots s^*_n \xleftarrow{R} \{0,1\}^n$ as the seed of the hinting PRG.
   - 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 $(\tilde{r}^*_{1,i}, \tilde{r}^*_{2,i}, \dots, \tilde{r}^*_{N,i}) \xleftarrow{R} \{0,1\}^{N \cdot \ell}$
   - For all $i \in [n]$ run $\mathsf{Equiv.Equivocate}(1^{\kappa'}) \to (\sigma^*_i, y^*_{i,0}, y^*_{i,1})$.
   - For all $i \in [n]$, set $y^*_i = y^*_{i,s^*_i}$.
   - Set $\mathsf{output}_1 = \left(s^*, \mathsf{HPRG.pp}^*, \{r^*_{x,i}, \tilde{r}^*_{x,i}\}_{x \in [N], i \in [n]}, \{\sigma^*_i, y^*_{i,0}, y^*_{i,1}\}_{i \in [n]}\right)$
   (b) Run Programme $P$ on a random tape and let its output be $\mathsf{output}_2$.

[8] 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.

(c) Output $(\mathsf{output}_1, \mathsf{output}_2)$.

- Return $\mathsf{output}_2$ to $\mathcal{A}$

3. $\mathcal{A}$ sends a challenge $\mathsf{tag}^* \in [N]$ to $\mathcal{B}_j$
4. Let $c^*_{x',i',\bar{s^*_{i'}}}$ be the commitment changed in Game $2_j$ as described above.
5. Send challenge tag $(x', \bar{s^*_{i'}})$ to challenger.
6. **Pre Challenge Phase:**
   - $\mathcal{A}$ makes $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \cdot)$ queries

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

     to $\mathcal{B}_j$.
   - $\mathcal{B}_j$ answers by running $\mathsf{CCA.ValAlt}$. This can be done efficiently using $\mathcal{B}_j$'s own Pre Challenge oracle access to $\mathsf{Same.Val}$ and runs $\mathsf{CCA.Find}$ manually. By Claim 5.3, these will all be to the same tag.
7. **Challenge Phase:**
   - Select a random bit $\beta$
   - $\mathcal{A}$ sends two messages $m_0, m_1$

   (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 $m_\beta$ and $\mathsf{output}_2$, with the exception that $c^*_{x',i',\bar{s^*_{i'}}}$ is computed as noted above and submit output to $\mathcal{A}$.
8. **Post Challenge Phase:** Proceeds exactly as Pre Challenge Phase.
   - Receive bit guess $\beta'$ from $\mathcal{A}$
9. If $\beta = \beta'$, output $0$. Otherwise, output $1$.

**Claim 5.9.** $\mathcal{B}_j$ is an $e$-conforming "same tag" adversary

*Proof.* The same tag property of $\mathcal{B}_j$ follows directly from Claim 5.3, as the only oracle queries it makes are through $\mathsf{CCA.ValAlt}$ on a fixed tag $\mathsf{tag}^*$. We need to verify two main properties, that $\mathcal{B}_j$ is PPT, and $\tilde{P}$ runs in $\mathsf{poly}(2^{\kappa^e})$ time and outputs $\mathsf{poly}(\kappa)$ bits. The former is to verify, as $\mathcal{A}$ only makes use of a polynomial number of efficient functions and oracle queries. To gauge the runtime of $\tilde{P}$, we can see the only inefficient steps it does is $n$ calls to $\mathsf{Equiv.Equivocate}(1^{\kappa'})$ and the running of $P$. We can see the former takes time $n \cdot \mathsf{poly}(2^{\kappa'})$, and the latter is run in time $\mathsf{poly}(2^{\kappa^{e'}}) \ll \mathsf{poly}(2^{\kappa^e})$ by $e'$ conformity of $\mathcal{A}$. Finally, we can see the output of $\tilde{P}$ is $\mathsf{poly}(\kappa)$ length as $\mathsf{output}_1$ is the result of all polynomial time algorithms and the $\mathsf{poly}(\kappa)$ bound on the output of $P$. □

**Claim 5.10.** The advantage of $\mathcal{B}_j$ in winning the $e$-computation enabled 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 note 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,

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

□

Since $\mathcal{B}_j$ is an $e$-conforming adversary with non-negligible advantage. The proof of the lemma follows immediately by contradiction from the above claims.

□

**Lemma 5.4.** Assuming that the hinting PRG is exponentially secure with $T = 2^{\kappa^\gamma}$ where $\gamma \in (0,1)$ from Definition 2.1. For any $e'$-computation enabled 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 $\kappa'' = \kappa'^{\frac{1}{\gamma}} = \kappa^{\frac{e}{\gamma}}$.

*Proof.* Let $\mathcal{A}$ be an adversary 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 $\mathsf{poly}(2^{\kappa^e})$ time algorithm $\mathcal{A}'$ which has advantage $\frac{1}{2p(\kappa)}$ in the hinting PRG Game as per Definition 2.1 where inputs were called on security parameter $\kappa''$.

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

1. Choose a random bit $a \in \{0,1\}$
2. Run $\mathcal{A}$
   (a) When $\mathcal{A}$ sends their TM $P$, run it and return the result.
   (b) Receive challenge tag $\mathsf{tag}^*$ from $\mathcal{A}$
   (c) **Pre Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com})$.
   (d) **Challenge Phase:**

- Compute $\kappa' = \kappa^e$.
- 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 $\mathsf{Equiv.Equivocate}(1^{\kappa'}) \to (\sigma^*_i, y^*_{i,0}, y^*_{i,1})$.
- For all $i \in [n]$, set $y^*_i = y^*_{i,s^*_i}$.

(e) **Part 2:**

- Set $c^* = \underline{r^\beta_0} \oplus m^*_a$.
- 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 $\mathsf{com}^* = \left(\mathsf{tag}^*, \mathsf{HPRG.pp}^*, c^*, (\sigma^*_i, \left(c^*_{x,i,0}, c^*_{x,i,1}\right)_{x \in [N] \setminus \{\mathsf{tag}^*\}})_{i \in [n]})\right)$ as the commitment. All of the randomness is used as the decommitment string.

(f) **Post Challenge Phase:** Receive challenge commitments com from $\mathcal{A}$ and respond with $\mathsf{CCA.ValAlt}(\mathsf{tag}^*, \mathsf{com})$.

(g) Receive $a'$ from $\mathcal{A}$.

3. If $a' = a$, then output $\beta' = 1$. Otherwise output $\beta' = 0$.

**Claim 5.11.** $\mathcal{A}'$ runs in time $\mathsf{poly}(2^{\kappa^e})$

*Proof.* The majority of $\mathcal{A}'$ involves simulating running $\mathcal{A}$. Since $\mathcal{A}$ is $e'$-computation enabled we can run $P$ in time $\mathsf{poly}(2^{\kappa^{e'}})$. The Pre and Post Challenge queries will invoke a polynomial number of calls to CCA.ValAlt, which require time $\mathsf{poly}(2^\kappa) \leq \mathsf{poly}(2^{\kappa^e})$. Finally, the challenge phase is invokes $n$ calls to Equiv.Equivocate, which has runtime $\mathsf{poly}(2^{\kappa'})$ by Definition 2.3. Since $N$ is poly bounded, the rest of the computation in the challenge phase is efficient, which bounds the total runtime with $\mathsf{poly}(2^{\kappa'}) = \mathsf{poly}(2^{\kappa^e})$ □

**Claim 5.12.** If $\mathcal{A}$ has advantage $|\mathsf{adv}^3_{\mathcal{A}} - \mathsf{adv}^4_{\mathcal{A}}| \geq \frac{1}{p(\kappa)}$, $\mathcal{A}'$ has advantage in the HPRG game in Definition 2.1 $\geq \frac{1}{2p(\kappa)}$

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

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

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

$$\left(r^0_0 \xleftarrow{R} \{0,1\}^\ell, \left\{r^0_{i,b} \xleftarrow{R} \{0,1\}^\ell\right\}_{i \in [n], b \in \{0,1\}}\right)$$

$\mathcal{A}$ 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 $\mathcal{A}'$ 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 $\mathcal{A}'$ runs in time $\mathsf{poly}(2^{\kappa^e})$, its advantage must be negligible by Definition 2.1, a contradiction, which concludes our proof. □

**Lemma 5.5.** 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.

# 6 Compiling our Transformations

We conclude by showing how to compile our transformations. Suppose that we begin with a base scheme supporting $32 \cdot \mathsf{ilog}(c, \kappa)$[9] tags for some constant $c$ that is secure against non-uniform attackers that make same tag queries, and supports randomness recovery. We will compile this into a scheme supporting $16 \cdot 2^{\kappa}$ space against uniform attackers with no same tag restriction.

We begin with applying the transformation of Section 5 to the base scheme which divides the tag space supported by 2 to get a scheme with $16 \cdot \mathsf{ilog}(c, \kappa)$ sized tag space, but removes the same-tag restriction. The we apply the Section 4 tag amplification process $c + 1$ times. Recall the transformation takes a $N' = 4N$ scheme to a scheme supporting $2^N$ tags. Since $16/4 = 4$ and $2^4 = 16$ the effect is of each application is to remove one of the $\lg$ iterations and keep the factor of 16. Since the transformation imposes a polynomial blowup in $N$ on the underlying scheme and since it is applied a constant number of times, the size of the resulting scheme will also be polynomial.

Below we give a formal construction utilizing the transformations $\mathsf{OneToMany}(\cdot)$ presented in Section 5, and $\mathsf{Amplify}(\cdot)$ presented in Section 4. Since we are transforming a scheme that takes $32 \cdot \mathsf{ilog}(c, \kappa)$ tags to $16 \cdot 2^{\kappa}$ tags, we need to use the amplification transformation $c + 1$ times. $\mathsf{OneToMany}(\cdot), \mathsf{Amplify}(\cdot)$ transformations take in a $e$-computation enabled scheme and output a $e' = e \cdot \delta$-computation enabled scheme where $e' \geq 1$ and $\delta \in (0, 1)$ and the equivocal commitment scheme is $2^{\kappa^{\delta}}$ hiding secure. We set $\mathsf{OneToMany}(\cdot)$ to take a $e \cdot \delta^{-c-2}$-computation enabled and output a $e \cdot \delta^{-c-1}$-computation enabled scheme. $\mathsf{Amplify}(\cdot)$ takes a $e \cdot \delta^{-c-1}$-computation enabled scheme and outputs a $e$-computation enabled scheme after $c + 1$ tranformations.

$\mathsf{CompiledAmplify}(\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val}), \mathsf{HPRG}, \mathsf{Equiv}, e)$

1. Let $\delta$ be the constant so that $\mathsf{Equiv}$ is $2^{\kappa^{\delta}}$ binding secure and $c$ be the constant such that the base scheme takes $32 \cdot \mathsf{ilog}(c, \kappa)$.

[9] For brevity, $\mathsf{ilog}(c, \kappa)$ denotes $\underbrace{\lg \lg \cdots \lg}_{c \text{ times}}(\kappa)$.

2. $\mathsf{AmplifiedCCA}^0 \leftarrow \mathsf{OneToMany}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{Equiv}, e \cdot \delta^{-c-1})$.
3. For $i \in [c+1]$
   (a) $\mathsf{AmplifiedCCA}^i \leftarrow \mathsf{Amplify}(\mathsf{AmplifiedCCA}^{i-1}, \mathsf{HPRG}, \mathsf{Equiv}, e \cdot \delta^{i-c-1})$
4. Output $(\mathsf{AmplifiedCCA}^{c+1}.\mathsf{Com}, \mathsf{AmplifiedCCA}^{c+1}.\mathsf{Val})$

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

**Theorem 6.1.** For every $\kappa \in \mathbb{N}$, let $\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val})$ be a perfectly correct CCA commitment scheme by Definition 3.1. Let $\mathsf{Equiv} = (\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ be a perfectly correct equivocal commitment scheme by Definition 2.2. Then, we have that the scheme $\mathsf{CompiledAmplify}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{Equiv}, e)$ is a perfectly correct CCA commitment scheme.

*Proof.* By the assumption that BaseCCA is a correct CCA commitment scheme and correctness of Equiv, we can apply Claim 5.2 to conclude $\mathsf{AmplifiedCCA}^0$ is a correct CCA commitment scheme. Using that again with the correctness of Equiv, we can conclude inductively that $\mathsf{AmplifiedCCA}^i$ is a perfectly correct CCA commitment scheme $\forall i \in [c]$ via Claim 4.2, including our final output $\mathsf{AmplifiedCCA}^{c+1}$. □

**Theorem 6.2.** For every $\kappa \in \mathbb{N}$, let $\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val})$ be an efficient CCA commitment scheme with randomness recovery satisfying Definition 3.2 with tag space $32 \cdot \mathsf{ilog}(c, \kappa)$. Let $\mathsf{Equiv} = (\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ be an efficient equivocal commitment scheme by Definition 2.3. Then, $\mathsf{CompiledAmplify}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{Equiv}, e)$ is an efficient CCA commitment scheme.

*Proof.* By the assumption that BaseCCA is an efficient CCA commitment scheme and the efficiency of Equiv, we can apply Claim 5.1 with BaseCCA being $e \cdot \delta^{-c-2}$-computation enabled, and $N = 16 \cdot \mathsf{ilog}(c, \kappa)$, to conclude $\mathsf{AmplifiedCCA}^0$ is an efficient CCA commitment scheme. Using that again with the correctness of Equiv, we can inductively apply Claim 4.1 to $\mathsf{AmplifiedCCA}^i$ with $\mathsf{AmplifiedCCA}^{i-1}$ being $e \cdot \delta^{i-c-2}$-computation enabled and $N = 4 \cdot \mathsf{ilog}(c - i, \kappa)$ (same as in Claim 6.2). Since $i - 1$ is at most $c$, we can see that $N$ will always be polynomial in $\kappa$, so Claim 4.1 can be applied $c+1$ times. Since $c+1$ is a constant, we use the fact that a constant number of compositions of polynomials is still polynomial and the final expression is going to be polynomially bounded in parameters $|m|, \kappa$ for Com, Recover and parameters $|m|, 2^\kappa$ for Val.

Consider algorithm Com. Let $\mathsf{poly}_0$ be the polynomial that denotes the efficiency for $\mathsf{AmplifiedCCA}^0$ after application of Claim 5.1 i.e. Com runs in $\mathsf{poly}_0(|m|, \kappa)$. After applying Amplify once, from Claim 4.1, $\mathsf{AmplifiedCCA}^1$ is a scheme with efficiency polynomial in $|m|, \kappa$ and efficiency of Com for $\mathsf{AmplifiedCCA}^0$. Thus we can represent runtime of Com for $\mathsf{AmplifiedCCA}^1$ by $\mathsf{poly}_1(\mathsf{poly}'_0(|m|, \kappa))$ where $\mathsf{poly}'_0$ includes the call of $\mathsf{poly}_0$ on message length equal to the length of decommitment strings produced by the equivocal commitment scheme on security parameter $\kappa^{e \cdot \delta^{-c-1}}$.

Simiarly, after applying Amplify on $\mathsf{AmplifiedCCA}^{i-1}$, from Claim 4.1 we can denote efficiency for $\mathsf{AmplifiedCCA}^i$ by $\mathsf{poly}_i(\mathsf{poly}'_{i-1}(\ldots \mathsf{poly}'_0(|m|, \kappa))$ where the message length for $\mathsf{AmplifiedCCA}^{i-1}$ is equal to the length of decommitment strings produced by the equivocal commitment scheme on security parameter $\kappa^{e \cdot \delta^{i-c-2}}$. This gives us that for Com the efficiency is given by

$$\mathsf{poly}_{c+1}(\mathsf{poly}'_c(\ldots \mathsf{poly}'_0(|m|, \kappa)) \in \mathsf{poly}(|m|, \kappa).$$

A similar analysis shows that $\mathsf{Recover} \in \mathsf{poly}(|m|, \kappa)$ and $\mathsf{Val} \in \mathsf{poly}(|m|, 2^\kappa)$. This concludes that our final scheme $\mathsf{AmplifiedCCA}^{c+1}$ is an efficient CCA commitment scheme.

□

**Theorem 6.3.** For every $\kappa \in \mathbb{N}$, let $\mathsf{BaseCCA} = (\mathsf{BaseCCA.Com}, \mathsf{BaseCCA.Val})$ be a CCA commitment scheme with randomness recovery that is hiding against non-uniform "same tag" adversaries according to Definition 3.10 for tag space $32 \cdot \mathsf{ilog}(c, \kappa)$. $\mathsf{HPRG} = (\mathsf{HPRG.Setup}, \mathsf{HPRG.Eval})$ be a hinting PRG scheme that is $T = 2^{\kappa^\gamma}$ secure by Definition 2.1 for $\gamma \in (0, 1)$. $\mathsf{Equiv} = (\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ be an equivocal commitment without setup scheme that is $T = 2^{\kappa^\delta}$ binding secure and statistically hiding for some constant $\delta \in (0, 1)$.

Then, $\mathsf{CompiledAmplify}(\mathsf{BaseCCA}, \mathsf{HPRG}, \mathsf{Equiv}, e)$ is a $e$-computation enabled CCA commitment scheme that is hiding against uniform adversaries according to Definition 3.9 for tag space $16 \cdot 2^\kappa$.

*Proof.* Consider the following sequence of claims.

**Claim 6.1.** $\mathsf{AmplifiedCCA}^0$ is a CCA commitment scheme secure against $e \cdot \delta^{-c-1}$ computation enabled adversaries with the recover from randomness property on tag space $16 \cdot \mathsf{ilog}(c, \kappa)$

*Proof.* By assumption in Theorem 6.3, $\mathsf{HPRG}$ is a $2^{\kappa^\gamma}$ secure hinting PRG scheme, $\mathsf{Equiv}$ is a $2^{\kappa^\delta}$ secure equivocal commitment without setup scheme, $\mathsf{BaseCCA}$ is a CCA commitment scheme that is hiding against non-uniform "same tag" adversaries with the recover from randomness property on tag space $32 \cdot \mathsf{ilog}(c, \kappa)$. Apply Claim 3.2 to get that $\mathsf{RandomBaseCCA}$ is secure against $e \cdot \delta^{-c-2}$-computation enabled "same tag" adversaries. Apply Theorem 5.1 with $N = 16 \cdot \mathsf{ilog}(c, \kappa)$ to get the result. □

**Claim 6.2.** For all $i \in [c+1]$, $\mathsf{AmplifiedCCA}^i$ is a CCA commitment scheme secure against $e \cdot \delta^{i-c-1}$-computation enabled adversaries with the recover from randomness property on tag space $16 \cdot \mathsf{ilog}(c - i, \kappa)$

*Proof.* We will proceed with induction on $i \in [0, c+1]$. The base case is true by Claim 6.1. By assumption in Theorem 6.3, $\mathsf{HPRG}$ is a $2^{\kappa^\gamma}$ secure hinting PRG scheme and $\mathsf{Equiv}$ is a $2^{\kappa^\delta}$ secure equivocal commitment without setup scheme. By our induction hypothesis, $\mathsf{AmplifiedCCA}^{i-1}$ a CCA commitment scheme secure against $e \cdot \delta^{i-c-2}$-computation enabled adversaries with the recover from randomness property on tag space $16 \cdot \mathsf{ilog}(c - i + 1, \kappa)$. We apply Theorem 4.1 with $N = 4 \cdot \mathsf{ilog}(c + 1 - i, \kappa)$. Since $i \leq c + 1$, 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 $\delta \cdot e \cdot \delta^{i-c-2} = e \cdot \delta^{i-c-1}$-computation enabled adversaries with the recover from randomness property on tag space $2^{4 \cdot \mathsf{ilog}(c-i+1,\kappa)} = 16 \cdot \mathsf{ilog}(c - i, \kappa)$, which proves the induction step. □

By applying Claim 6.2 on $i = c + 1$, we conclude $\mathsf{AmplifiedCCA}^{c+1}$ is a $e$-computation enabled CCA commitment with tag space $16 \cdot 2^\kappa$. By Claim 3.1, this is standard secure against uniform adversaries.

□

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

**Theorem 6.4.** [KK19] For every constant $c > 0$, there exist same-tag CCA secure commitments with randomness recovery satisfying Definition 3.10 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 $\mathsf{BQP}$, both with randomness recovery.

**Theorem 6.5.** [LPS17] For every constant $c > 0$, there exist CCA secure commitments with randomness recovery satisfying same-tag CCA security w.r.t. over-extraction according to Definition 3.10 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 w.r.t. over-extraction. The [KK19] scheme does not suffer from over-extraction and achieves (standard) same-tag CCA security. 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 these theorem with Theorem 6.3, we obtain the following corollaries.

**Corollary 6.1.** There exists a constant $e > 0$ for which there exists a perfectly correct and polynomially efficient $e$-computation enabled CCA secure commitment satisfying Definition 3.6 against uniform adversaries, with tag space $2^\kappa$ for security parameter $\kappa$, that makes black-box use of subexponential quantum hard non-interactive commitments with randomness recovery, subexponential classically hard non-interactive commitments in $\mathsf{BQP}$ with randomness recovery, subexponential hinting PRGs and subexponential keyless collision-resistant hash functions.

**Corollary 6.2.** There exists a constant $e > 0$ for which there exists a perfectly correct and polynomially efficient $e$-computation enabled CCA secure commitment satisfying Definition 3.6 against uniform adversaries, with tag space $2^\kappa$ for security parameter $\kappa$, 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 $2^\kappa$, without the same tag restriction.

## References


[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.

[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.

[BL18] 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.

[CDSMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, Black-Box Constructions of Adaptively Secure Protocols. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009*, pages 387–402, 2009.

[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.

[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.

[GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. New constructions of hinting prgs, owfs with encryption, and more. *IACR Cryptology ePrint Archive*, 2019.

[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.

[KK19] Yael Tauman Kalai and Dakshita Khurana. Non-interactive non-malleability from quantum supremacy. 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 III*, volume 11694 of *Lecture Notes in Computer Science*, pages 552–582. Springer, 2019.

[KMT19] Fuyuki Kitagawa, Takahiro Matsuda, and Keisuke Tanaka. CCA security and trapdoor functions via key-dependent-message security. 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 III*, volume 11694 of *Lecture Notes in Computer Science*, pages 33–64. Springer, 2019.

[KS17] Dakshita Khurana and Amit Sahai. How to achieve non-malleability in one or two rounds. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 564–575. IEEE Computer Society, 2017.

[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 Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 576–587. IEEE Computer Society, 2017.

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

[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.

[PSV06] Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Construction of a non-malleable encryption scheme from any semantically secure one. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 271–289. Springer, 2006.

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

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, 1996.

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

# A Proofs for Equivocal Commitments without Setup

In this section, we will show that any efficient, statistically hiding, $T(\kappa)$ binding commitment scheme $(\mathsf{Com}, \mathsf{Decom})$ [DPP93, HM96, BKP18] without setup is also an efficient $T(\kappa^{1/c})$ binding equivocal bit commitment without setup for some constant $c$.[10]

**Definition A.1.** We say a bit commitment scheme is statistically hiding if the statistical distance between the following two distributions is negligible.

- $\mathcal{D}_0 = c$ where $(c, d) \leftarrow \mathsf{Com}(1^\kappa, 0)$.
- $\mathcal{D}_1 = c$ where $(c, d) \leftarrow \mathsf{Com}(1^\kappa, 1)$.

Let $r(\kappa)$ be the amount of randomness used by $\mathsf{Com}(1^\kappa, \cdot)$. Since we know $\mathsf{Com}$ is efficient, we can bound length of $r$ by some polynomial $O(\kappa^c)$ (assuming $r \in \{0,1\}^{\kappa^c}$ for simpler notation). We set $\kappa'$ to be $\kappa^{1/c}$ and call $\mathsf{Com}$ with $\kappa'$.

$\mathsf{Equiv.Com}(1^\kappa, b)$

- Set $\kappa' = \kappa^{1/c}$
- Output $(c, d) \leftarrow \mathsf{Com}(1^{\kappa'}, b)$

$\mathsf{Equiv.Decom}(c, d)$

- Output $\mathsf{Decom}(c, d)$

$\mathsf{Equiv.Equivocate}(1^\kappa)$

- Set $\kappa' = \kappa^{1/c}$, $\mathsf{D} = \emptyset$
- Let $(c_0, d_0) = \mathsf{Com}(1^{\kappa'}, 0)$
- For randomness $r \in \{0,1\}^\kappa$
  - Run $(c_1, d_1)\mathsf{Com}(1^{\kappa'}, 1; r)$
  - If $c_1 = c_0$, add $(r, d_1)$ to a set $\mathsf{D}$.
- If $\mathsf{D} = \emptyset$, output $(\perp, \perp, \perp)$
- Otherwise, select a unformly random $(r', d_1') \in \mathsf{D}$
- Output $(c_0, d_0, d_1')$.

[10] Note that in our construction we require $T(\kappa^{1/c})$ to be subexponential, so it suffices for $T(\kappa)$ to be subexponential as well.

**Lemma A.1.** If $(\mathsf{Com}, \mathsf{Decom})$ is a correct commitment scheme, $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ is a correct equivocal commitment scheme as per Definition 2.2.

*Proof.* Since $\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}$ simply call and output $\mathsf{Com}, \mathsf{Decom}$, correctness directly follows from the commitment of the underlying scheme. □

**Lemma A.2.** If $(\mathsf{Com}, \mathsf{Decom})$ is an efficient commitment scheme, $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ is an efficient equivocal commitment scheme as per Definition 2.3.

*Proof.* By the efficiency of $\mathsf{Com}, \mathsf{Decom}$, we know that their runtime can be bound by some polynomial $p(\cdot)$, so their invocations in $\mathsf{Com}, \mathsf{Decom}$ are bounded by $p(\kappa') = p(\kappa^{1/c})$, which is of course still polynomial in $\kappa$. Finally, we note the runtime of $\mathsf{Equiv.Equivocate}$ is dominated by invocations of $\mathsf{Com}$ iterating over $r \in \{0,1\}^\kappa$, which is $\mathsf{poly}(\kappa)2^\kappa \in \mathsf{poly}(2^\kappa)$. We also remark that the existence of a $c$ which a polynomially bounds the amount of randomness $\mathsf{Com}$ uses also relies on the efficiency of $\mathsf{Com}$. □

**Lemma A.3.** If $(\mathsf{Com}, \mathsf{Decom})$ is a $T(\kappa)$ binding secure commitment scheme, $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ a $T(\kappa^{1/c})$ binding secure commitment scheme as per Definition 2.4.

*Proof.* Since $\mathsf{Equiv.Com}$ simply calls $\mathsf{Com}$ on security parameter $\kappa' = \kappa^{1/c}$, $T(\kappa')$ binding follows directly from the $T(\kappa)$ binding of $\mathsf{Com}$. □

**Lemma A.4.** If $(\mathsf{Com}, \mathsf{Decom})$ is statistically hiding as per Definition A.1, then $(\mathsf{Equiv.Com}, \mathsf{Equiv.Decom}, \mathsf{Equiv.Equivocate})$ is equivocal as per Definition 2.5.

*Proof.* First, we define the following distributions

- $\mathcal{D}_0 = (c, d)$ where $\mathsf{Equiv.Com}(1^\kappa, 0) \to (c, d)$
- $\mathcal{D}_1 = (c, d)$ where $\mathsf{Equiv.Com}(1^\kappa, 1) \to (c, d)$
- $\mathcal{D}'_b = (c, d_b)$ where $\mathsf{Equiv.Equivocate}(1^\kappa) \to (c, d_0, d_1)$

**Claim A.1.** $\mathcal{D}_0$ and $\mathcal{D}'_0$ are statistically close.

*Proof.* We observe that these distributions are generated identically except $\mathsf{Equiv.Equivocate}$ has a possibility of aborting when $\mathsf{D} = \emptyset$. But the event $\mathsf{D} = \emptyset$ means that the $c_0$ generated by $\mathsf{Equiv.Equivocate}(\cdot)$ was not in the support of $\mathsf{Com}(1^{\kappa'}, 1)$, lower bounding the statistical distance between $c \in \mathcal{D}_0$ and $c \in \mathcal{D}_1$. Since this is negligible by statistical hiding $\mathsf{D} = \emptyset$ can only happen with negligible probability, so the claim holds. □

**Claim A.2.** $\mathcal{D}_1$ and $\mathcal{D}'_1$ are statistically close.

*Proof.* We examine the marginal distribution of $d \in \mathcal{D}_1$ and $d_1 \in \mathcal{D}'_1$ conditioned on a fixed $c'$. We can observe that in $\mathcal{D}_1$, $d$ is generated by a uniformly random $r$ which over the subset of $r : (c, d) \leftarrow \mathsf{Com}(1^{\kappa'}, 1; r) \wedge c = c'$. But this is exactly the set $\mathsf{D}$ from which $d_1$ is drawn, so these marginal distributions are identical.

Observe from the facts that $c \in \mathcal{D}'_0$ is statistically close to $c \in \mathcal{D}_0$ (by Claim A.1), and $c \in \mathcal{D}_0$ is close to $c \in \mathcal{D}_1$ (by statistical hiding), we can conclude that $c \in \mathcal{D}_1$ is statistically close to $c \in \mathcal{D}'_1 = c \in \mathcal{D}'_0$. Our claim follows directly from the combination of these two observations. □

That this satisfies the definition of equivocal follows from Claim A.1 and Claim A.2. □